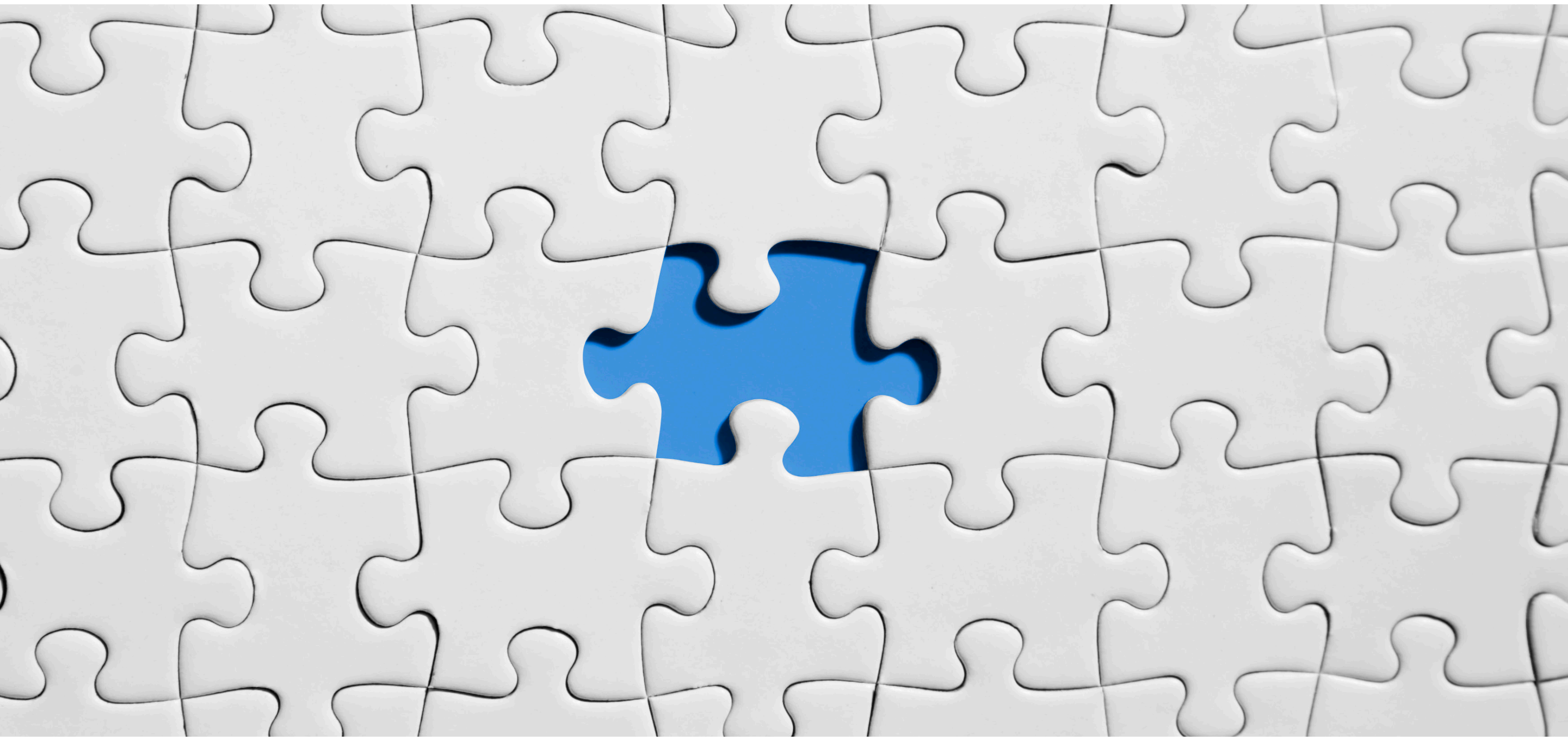
# AN OBJECTIVE USE OF BLOCKCHAIN IN OBJECT STORAGE



## Anthony Dutra
Systems Engineer
Dell Technologies
Anthony.Dutra@dell.com

## Varshith AnilKumar
Senior Software Engineer
Wayfair LLC
Varshithanilkumar@gmail.com

## William Lupo
Senior Software Engineer
Thomson Reuters
Williamdelmontlupo@gmail.com

## Table of Contents

## Abstract

Each day we post, snap, stream, and download our lives onto the Internet. This is drastically changing the way our data is formatted and stored. If data is the new oil, then this unstructured data – everything from PDFs to MP3s – is rocket fuel. Thriving in the age of digital transformation are those businesses technically savvy enough to skillfully deploy object storage solutions centered around this growing unstructured data problem. Built on top of these object-based infrastructures, organizations are integrating insightful applications like Hadoop to transform those blobs of data into meaningful information for the business. However, these implementations often are not cost effective or fast for organizations to deploy. Object-based storage solutions are difficult to start at a smaller scale. Hardware options commonly require a steep upfront cost and heavily rely on the performance of the solution in order to produce timely value back to the business. Meanwhile, software implementations struggle with security and data consistency across heterogenous hardware. The resolution to the issues facing software-based object storage solutions is simple for companies like Storj. The answer is blockchain.

This article begins with an introduction to unstructured data, what industries it is growing in and why. Next, the article will focus on the current limitation and inefficiencies. During this section, the article will educate readers by delving into the intricacies Dell Technologies flagship object-storage platform, Elastic Cloud Storage (ECS), describing its functions and internal mechanisms to store, manage, and protect a businesses unstructured data. This is followed by a section describing Hyperledger blockchain, an open-source permissioned blockchain developed and maintained by the Linux Foundation and IBM. These introductions lead to an explorative piece on the concept of "blockchain storage", an amalgam of blockchain and object-storage technology which can be employed to further drive transformation of business applications.

Next, the article dives into what companies like Storj and Sia are doing in this space to enhance software-based object storage solutions with blockchain technology. Then the article introduces a theoretical blockchain-enhanced object-storage cluster- a conceptual architecture using ECS and Hyperledger blockchain as the distributed ledger technology (DLT) across each node. The article will explore how this integration would vastly improve overall system performance, security, and availability while pointing out drawbacks of incorporating Hyperledger blockchain into the ECS platform. The article concludes by offering a step-forward in solving those technical drawbacks by incorporating machine learning into the design before concluding with a summary of key takeaways from the article.

Readers can expect to learn a breadth of topics in unstructured data, learning more about how object storage solutions can change their business, the inner workings of such solutions and how they are evolving businesses who use them. Readers can also expect to develop a strong understanding in what blockchain is, how it functions in its many different forms, and how it – with other emerging technologies such as machine learning – can be used to improve these types of deployments.

## What is an Object?

Scroll through the photo library on your phone. Each picture, while it represents a personal memory of its own, is a series of 0’s and 1’s to your phone and any server the application it is living on persists. This is the same for your music, PDFs of your tax returns, and any other media on that device. This type of data is categorized as “unstructured data” and it differs significantly from its structured counterpart. It is up to you, as an application developer or any IT professional working to balance these different types of data structures to best optimize IT for the business.

A good rule of thumb is to remember that structured data, such as databases, neatly fit in rows and columns (i.e. SQL database or Excel spreadsheet). Everything else can broadly fall under the umbrella that is unstructured data. As these two categories of data are unique, they must be stored on physical media in different formats.

Both structured and unstructured data is stored on a physical media (i.e. hard disk or solid-state drive) as blocks (those 1’s and 0’s mentioned earlier). The applications we interact with on our phones or computers write the data we post, update, append, and so forth in this structured format to keep track of the address (location) of each block of data. For data constructs like SQL databases, there is no need to further transform this data from the block format. However, those pictures and music files are more complex – *unstructured*-data – constructs which require another level of abstraction to be stored.

The difficulty is that not all unstructured data can be organized into this hierarchical format. Consider the thousands of home listings on Zillow.com or the thousands of pictures streaming from social media sites. If the IT organizations of those companies tried to store all that information in such a rigid format, it would suffer in performance if each document and their metadata were stored in a hierarchical format. This is what is known as a second type of unstructured data – object data. Instead of storing data in a hierarchy, object data is stored in a single flat namespace. This is stored as a clump of data with its metadata and an additional unique identifier. Unlike its block counterparts, object storage does not have many indexing-related limitations when it comes to writing data in this schema. This means object storage solutions can scale to petabytes (PB) of data.

To illustrate the key differences of block and object, let’s look at an example; the social media website Facebook, which provides its users on their application an ‘On This Day’ post. The application will send the user a notification with a post that they made on the website on the same date at some random year in the past and ask if you want to post it to “reshare that memory” on your feed. Everything that makes up that post – the picture, text/comments, total number of likes, and all the metadata and unique identifiers associated with that post – can be considered an object.

### How Objects are Organized in the Data Center

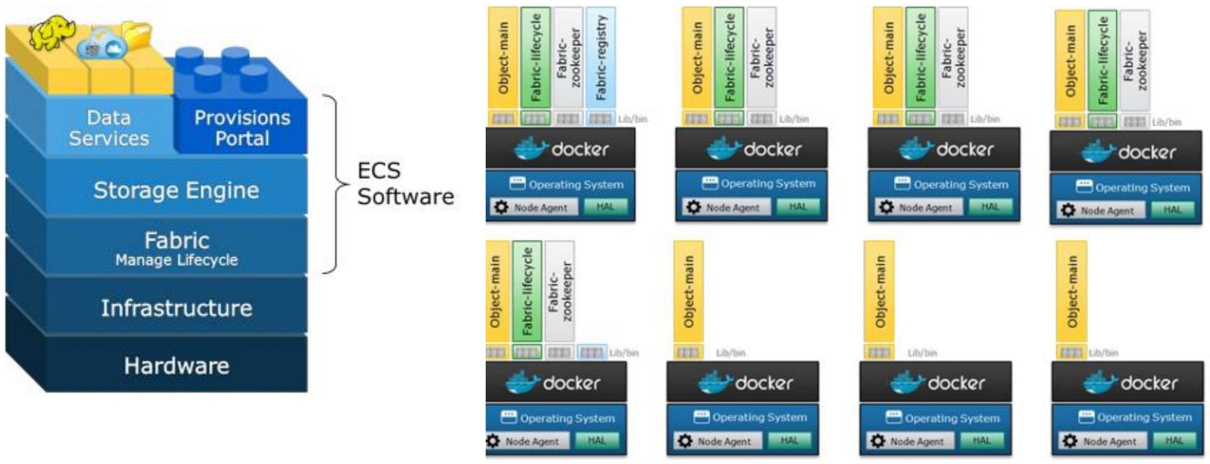
In the data center, those thousands upon thousands of Facebook posts are split up and written to “nodes”. Think of nodes simply as a logical cluster of compute resources (i.e. servers) running a software-defined storage application which is transacting with user applications. These objects are organized into “buckets”, similar to a folder on a Windows file system. As these groups of objects grow, nodes are added to the object storage cluster. Depending on the implementation of the object storage solution, this may simply be adding another physical node or in a software-defined manner by installing a virtual version of the platform on commodity hardware and connecting the clusters over a network.

Regardless of the implementation choice, whichever nodes are a part of the that object storage cluster must be able to handle tracking of the metadata catalog (the data about the object) and must be able to respond independently to requests from applications seeking to access that data. It is important to have this work as efficiently as possible since retrieving data from an object store requires the application to call for its object ID in that flat namespace. No matter where each piece of that object lives, each of the nodes must be in sync in order to serve user applications efficiently.

Let’s explore how all this works through Dell Technologies’ flagship object storage solution – Elastic Cloud Storage (ECS).

In both physical turnkey appliance and software-defined implementations, ECS is designed to be an industry standard object repository which allows for application interactivity with multiple object (S3, Atmos, Swift, and CAS) and file (HDFS and NFS) protocols(1). Applications can GET, POST, PUT, and DELETE via HTTP or HTTPS calls to objects in buckets.

The magic behind the technology is the ECS software which is architected in layers of code (Docker containers) that work together atop the validated hardware and SUSE Linux-based operating system. This is best illustrated and described in the September 2019 *ECS Overview and Architecture Whitepaper*(2):



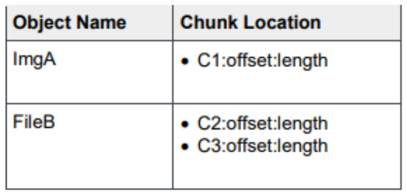
Illustrated above is a graphical view of the ECS Software layers and a diagram of how the software and Docker containers are deployed (in an 8-node configuration). The start of the ECS Software layers is the Fabric. This layer is responsible for health, configuration, upgrade management, Docker image Registry, event library and alerting for cluster(2). This is crucial for managing the lifecycle of the cluster. The next layer up is also the core layer of the ECS Software. The Storage Service is responsible for handling data availability – ensuring it is stored, retrieved and protected against data corruption and hardware failures by replicating data (and metadata) locally within the cluster or across sites of clusters.

The Storage Service is core to the ECS Software stack as it is responsible for managing and processing all data transactions flowing through the ECS cluster. These services are encapsulated within a Docker container that runs on every ECS node to provide a distributed and shared service[3]. All incoming data is fed to these services via the Data Services layer. This layer is responsible for taking client requests, extracting required information, and passing it back to the Storage Service layer for processing[1]. It is this layer that is interacting with the application on the details of the object to store.

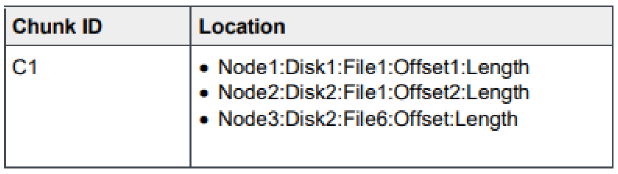
It is important to understand how the Storage Service layer does this in a fair bit of technical detail to show not only how this process works but why it works in this fashion. The value of processing object data in this format allows for such geographical access and scalability of object storage solutions. This means applications – referencing back to our ‘On This Day’ Facebook example – can create interesting functionalities that are of value for users. Supplementing the explanation of this process and the its illustrations will be adapted mainly from the September 2019 *ECS Overview Whitepaper and Architecture*.

Let’s recall that data (i.e. images, Facebook posts, music files, etc.) are stored on object storage solutions as data. This data is not only made up of that object but a plethora of other metadata about it. When written to the ECS platform, this data is broken down into “chunks”. These chunks of data live across all the nodes on the ECS cluster and are made up of all different objects stored on that cluster. Again, the benefit of object storage solutions like ECS is that the bundle of metadata created alongside the data is used to index where all these chunks of objects live in the cluster. Now what the system gives in terms of data dispersion and protection, it makes up for in resource consumption which ultimately dilutes performance.

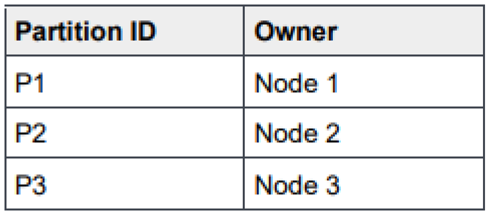
Applications can access any of the objects on the cluster without having to worry about changes made to an object as chunks are written in an append-only format. The append-only behavior means that an application’s request to modify or update an existing object will not modify or delete the previously written data within a chunk, but rather the new modifications or updates will be written in a new chunk[3]. All these different chunks are going to have a lot of metadata associated with each, making it quite the task to manage. In order to do so, ECS internally leverages logical tables to keep track of the location of data and metadata chunks on disk. The table stores this information into key-value pairs. These unique identifiers use a hash function to efficiently search and retrieve values associated with the key. These key-value pairs are eventually stored in B+ tree for fast indexing of data locations[3]. A B+ tree is an internal algorithm which measures and orders the capacity of nodes and balances each key-value pair to further enhance the query performance of these logical tables. In order to ensure metadata redundancy ECS objects and their metadata is stored into two tree-like structures where a smaller tree is in memory (memory table) and the main B+ tree resides on disks[4]. As data is written to the cluster, these logical tables use a journal to log the information of each record to disk in triple-mirrored chunks[4]. Triple-mirrored simply means that the data is written three times to disk to ensure that if a disk fails, the data is still available (up to three times). This means that after each transaction is fed into the system, any changes happening to the in-memory table are secure as the journal is using physical space on the system to track changes not yet committed to the B+ tree. When the memory table fills up or after a certain period, the table is eventually merged, sorted, or dumped into B+ tree on disk.



| Object Name | Chunk Location |
|---|---|
| ImgA | • C1:offset:length |
| FileB | • C2:offset:length<br>• C3:offset:length |



| Chunk ID | Location |
|---|---|
| C1 | • Node1:Disk1:File1:Offset1:Length<br>• Node2:Disk2:File1:Offset2:Length<br>• Node3:Disk2:File6:Offset:Length |



| Partition ID | Owner |
|---|---|
| P1 | Node 1 |
| P2 | Node 2 |
| P3 | Node 3 |

These writes to and from memory to disk are organized into three different tables: the Object table (OB), positioned top left, the chunk table (CT), top right, and the Partition Records table (PR), on the bottom of the above illustration(4). The details of what each table is tracking is beyond the scope of this article. What is important to demonstrate is that in order to ensure high availability, geographical scalability, and data protection these object storage platforms must use a series of tables (multiple *structured* data concepts) to track the data and metadata associated with an object.

### The Impact

As those tables upon tables of tracking metadata begin to grow, there eventually becomes a performance impact when writing data into resource saturated buckets. As the whitepaper notes, the impact to operations increases as the number of indexed fields increase(1). Impact to performance needs careful consideration on choosing if to index metadata in a bucket, and if so, how many indexes to maintain(1). The result is slowed client responses and an overall negative impact to end user experience. Not to mention, most hardware-based implementations of these object storage solutions require a large capital expenditure to procure. For example, to ensure strong, consistent delivery of an object to an application, many entry-level hardware implementations are at least five (5) physical servers. Many enterprise investment organizations rely on these platforms to interact with technologies such as Hadoop to stream things like historical financial data. This data – while generally infrequently accessed – holds immense value to algorithms that help dictate the pulse of the global economy. Use cases such as these is why it is important that object storage solutions can enable strong consistency of the data stored on the cluster as cost effectively as possible. This idea becomes particularly true when solution designers seek to mix hardware and software implementations as it is difficult to ensure consistency of data across different systems.

## What is Blockchain?

Blockchain is a distributed ledger technology that is much more than a cryptocurrency. At a high-level, blockchain is packaged code of containerized processes (usually in a cluster of containers or virtual machines) running on nodes which store a very small amount of data onto a highly encrypted and distributed ledger service across a shared peer to peer network. Blockchain offers a secure and synchronized record of transactions to all nodes involved on a shared network. As each transaction occurs, it is stored in a block. Each block is connected to the one before and after it. Groups of transactions are blocked together, and a fingerprint of each block is added to the next, thus creating an irreversible chain. The blockchain ledger records every sequence of transactions from beginning to end. On top of this, many blockchains allow for a “smart contract” functionality, automating much of the application’s business logic without needing to code outside the blockchain application being developed.

Blockchain solves many of the inefficiencies prevalent in seemingly daily processes. For example, buying a car. The purchase and procurement of a new vehicle requires many third-parties to validate the identity, financial information, and credibility of the individual purchasing a car. Think about how many organizations are involved and how much of that information must remain consistent and valid throughout the purchasing process. Processes such as these using traditional data storage schemas (like SQL) are inefficient, expensive and, if centralized, vulnerable. Many centralized database applications are inefficient because interactions are managed point to point. Business rules are implemented by all those who are part of the process individually. This leads to data inconsistencies which leads to larger IT expenses not to mention the risk of being a single attack zone for hackers (i.e. SQL injections). Other issues such as disputes over data validity result in high reconciliation costs by third-parties.

In summary, blockchain is a distributed ledger technology that stores data in a ledger replicated across all nodes involved in the transaction of processing whatever request an application makes. The technology balances security with efficiency, unlike it’s centralized – or structured – data storage schema counterparts which can be inefficient, expensive, and vulnerable to several outside factors. These factors are why this article proposes that blockchain technology should be considered as a solution to many of the inefficiencies presented in object storage solution services as they read/write objects for business applications.

Other organizations agree with this sentiment. The next section explores the emerging market of decentralized cloud services driven by the concept of decentralized cloud storage on the blockchain.

## Centralized Cloud Storage

Cloud storage is a sector traditionally dominated by cloud storage companies like AWS, GCP and Azure. As these large public cloud organizations can purchase server resources at scale, their object storage managed services (AWS-Simple Storage Service, Azure-Blob Storage, and GCP-Google Cloud Storage) are cost effective to use as they eliminate the need to physically build and provision object storage solutions; you can rent a bucket instead. These large organizations can provide users a simple interface and ability to purchase a certain amount of storage that is generally sold by the gigabyte (GB) consumed per month or at a flat price, usually by the terabyte (TB).

While these services are fantastic to leverage as a public cloud customer, it is important to understand that these managed services are still subject to hardware outages hosting them. Many of these services run on a very similar object storage solution that functions generally the same as ECS does. Therefore, in order to offset any downtime, public cloud providers offer a service level agreement (SLA) to ensure durability and availability of the object stored in the managed service bucket. This is difficult to promise as when these services go down, it has a large-scale impact.

A 2019 blog post related to this topic points out the many cloud outage incidents in 2018 from all three public cloud providers. The July 2018 Google Cloud outage stalled platforms like Snapchat, Spotify, and Pokémon Go from 12:17 to 12:55 PST. AWS outage silenced Alexa and highly popular online platforms like Slack and Twilio[5]. As much of these public cloud data centers are centralized locations in a specific geography, security can be considered another point of concern. While encryption techniques (at rest or in transit) can be deployed on public managed service buckets, vulnerabilities persist as they would in any data center. Resourceful IT organizations seeking to avoid both issues are taking notice of blockchain to solve these problems.

## Decentralizing Cloud Storage with Blockchain

Blockchain technology can augment object storage in the cloud as it gives users a means to store information as part of a peer to peer network. As described in previous sections, much of the data protection functionalities to read/write the information across a cluster is facilitated mainly by transactions. Imagine instead of running redundant tables to track metadata, we offload that process to a blockchain service which not only replicates and secures metadata written to it across the nodes, but also enables willing nodes to give up their storage to become part of a chain of computers to collectively create a bucket for objects. As blockchain is an append-only database, nodes that share a network can expect to more efficiently write and read object data across the cluster in a secured, immutable, and most importantly, decentralized (good for fault tolerance and efficiency) fashion.

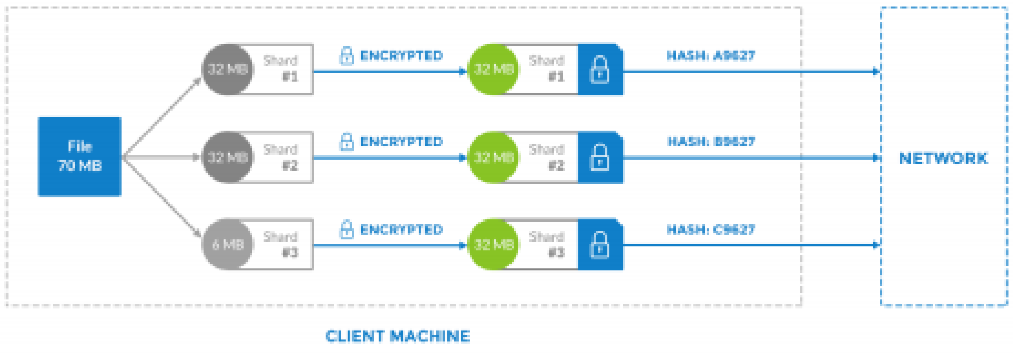
Theoretically, each object (or just an object’s metadata) would be split up and hashed and replicated across all nodes active on the network to collectively create a bucket. This ensures that even if one bad actor was able to decrypt a piece of data, they would not have access to the entire file as no node would be able to identify who the data belongs to so that the true user remains anonymous[6]. This allows for an extra layer of security that is inherently built into the blockchain service. If something does get flagged indicating that a node attempted to manipulate or alter data, that user will be immediately removed[6]. These inherent security features and decentralized data management structure of blockchain technology solves many of the issues prevalent in object storage solutions today.

The proceeding section explores a startup IT organization pioneering these blockchain implementations of object storage solutions in more detail.

### Storj-Farmers & File Shards

Storj Labs is a new startup exploring the use of blockchain to create a decentralized object storage solution. Over seven rounds of funding, they raised a total of $35.4 million after their 2017 initial coin offering (ICO)[7] .

Storj operates by encrypting user data initially written to the application onto all client machines as a standard transaction and associated metadata. Leveraging their File Sharding algorithm[8], the object is split into shards to protect data in transit as they are distributed. A shard is an encrypted portion of a file that a user would like to store on the network. These shards are managed by “farmers” – a user that is leasing his or her hard drive space to the network in a standardized shard size as a byte multiple such as 8 MB or 32 MB[8]. The benefit of this sharding allows large files to be significantly more controllable as they are distributed throughout the network.



The diagram above illustrates the sharding process on each client adapted from the *Storj - A Peer-to-Peer Cloud Storage Network Whitepaper.* Each client is holding secured information which is auditable across the network through smart contracts. This is able to be achieved through two proprietary protocols: Proof-of-Storage and Proof-of-Redundancy(8).

Proof-of-Storage is used to ensure integrity and accessibility of a shard by having a farmer prove cryptographically that they can validate it. This is done by running a batch of services which are summarized below:

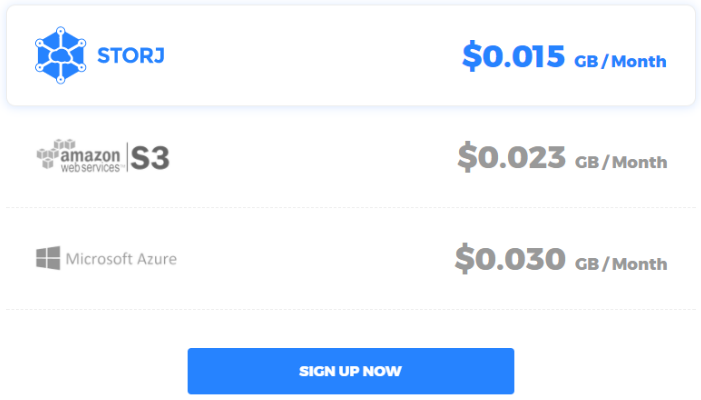
- *Proof-of-Storage via Merkle Audits*
  - Uses Merkle trees and proofs to implement a trustless data storage network. This provides a method for a client to audit that the data he or she has stored on the network is available and unmodified(8).

- *Proof-of-Storage via Pre-generated Audits*
  - An alternative method for auditing the chain of shards through a hash challenge(9). This ensures that no farmer can modify a file.

- *Full, Cycle, and Deterministic Heartbeats*
  - Methods for verifying hash challenges in order to guarantee data integrity(9) throughout the chain of nodes.

Proof-of-Redundancy, subverts the issues of needing so much capacity in order to ensure data protection in case of a component failure in a node (i.e. triple mirroring of data and metadata described earlier in the article). Storj guarantees physical redundancy by storing shards using a K-of-M erasure encoding scheme with multiple farmers(10). K-of-M erasure encoding scheme is what ensures shard availability across the active nodes through a “RAID-like” data protection schema. This means that even if a farmer decides to turn off their node from the system, the decentralized bucket will still be able to host the object data stored on it.

Farmers are compensated with the cryptocurrency Storjcoin X (SJCX) for their bandwidth and storage space they have provided to the network by the client after the completion of a heartbeat(11). Unlike the fixed service cost as public cloud providers do today, Storj uses this cryptocurrency to balance the relationship between clients requiring buckets and farmers giving up their resources. Monitoring is done by using “gates” which track and ensure a SLA for the service(12). Operating in this format, the cost of object storage bucket space is reduced for clients creating buckets. Reference the below table from the website *BlockApps*, claiming a $0.008 GB/month(13):



| | |
|---|---|
| STORJ | $0.015 GB / Month |
| amazon web services S3 | $0.023 GB / Month |
| Microsoft Azure | $0.030 GB / Month |

While low GB/month costs and incentives seem like a great new format, there are quite a few drawbacks. The first issue that arises is the reliance on those to give up their computational resources to create buckets (i.e. farmers). The incentive to keep providing resources to the managed service is based on a cryptocurrency that has its own complexities. To ensure uptime of objects stored, Storj must rely heavily on the (most likely) commodity hardware operating these decentralized buckets. While their erasure coding methodologies keep data redundant throughout each of the disparate nodes, the physical integrity that they've applied this strong software data protection methodology makes it seemingly obsolete.

What if we were able to leverage blockchain efficiencies in an object storage solution, but with proven hardware to support it?

## Evolving ECS with Permissioned Blockchain

The emergence of blockchain-based object storage solutions like Storj create an interesting dynamic in how buckets are created, how data is stored/validated, and the cost structure designed around it. In this new decentralized model of object storage, it removes much of the risk prevalent in centralized data schemas. Blockchain provides much of the data protection through its cryptographically secured and redundant nature inherent to the software (i.e. Proof-of-Redundancy). As metadata is stored in specialized format more efficiently, failure (or removal) of nodes doesn't interrupt application availability to buckets. New nodes onboarded to the cluster are done so securely and seamlessly, as the blockchain services chain together all components automatically. This blockchain-based object storage solutions can be provided at a low variable cost back to clients. However, as we've learned, much of this strongly consistent object data stored with blockchain technology is being done on unreliable (most likely commodity) hardware. This means blockchain object storage solutions must rely on their incentive business model to maximize the node count validating object storage transactions. This factor, we would argue, subverts much of the efficiencies gained by implementing blockchain services.

Hardware implementations like ECS face the contrasting issues. These deployments are initially capital expenditure-heavy and require an extensive IT background to get buckets ready to use for business applications. Interactivity with software deployments is also lacking. As discussed in a previous section, the way in which ECS writes data to store object data redundantly requires consuming resource overhead on the cluster. As such, it is difficult to interoperate and mix physical and software deployments (although both are options) as the hardware is not homogenous.

To tear down these barriers and make ECS more efficient, we suggest replacing many of these containerized and cumbersome data redundancy processes with blockchain services. This article proposes that the addition of a container running a blockchain service (described in the next section) will be able to reduce the overall overhead generally associated with storing objects on the ECS solution. This is because this isolated process is not only inherently replicating the data and metadata of an object but is doing so in a much more secure format. Integrating blockchain services across the heterogenous nodes in an ECS cluster would provide the best of both worlds. We would be able to take advantage of the effectiveness in transacting object data and metadata across the cluster without worry of excessive overhead or incentives to encourage nodes to stay on the network.

This article proposes augmenting ECS with a blockchain service that will enable:

- Creation of metadata consistency and storage efficiency across nodes
- Replication in case of node component failure
- Use of cryptography to read/write data in an auditable format
- Addition of more nodes or commodity nodes running the same hardware that streamlines the lifecycle process
    - Querying capabilities are only given to nodes on the network which are permissioned to do so

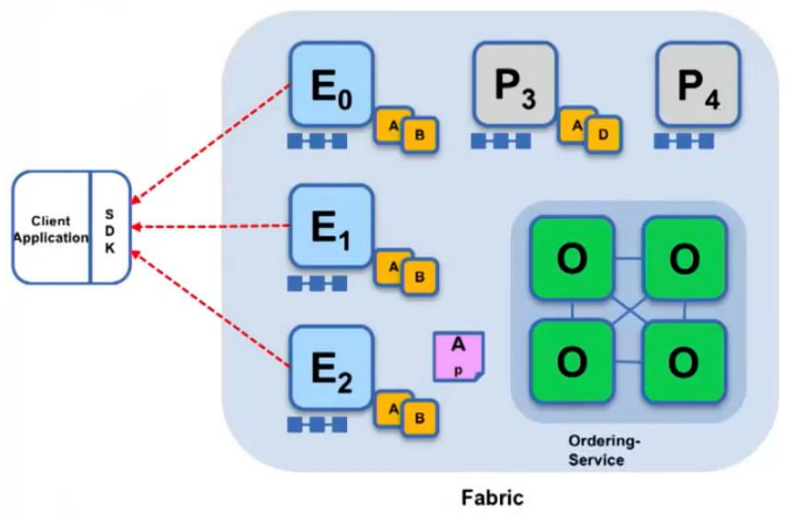
Next, this article delves into a blockchain technology best suited to run the proposed sub-process to boost the proficiency of an ECS solution.

## Hyperledger-Permissioned Blockchain

Not all blockchain technologies are created equal. One could argue that Hyperledger blockchain earns the right to be crowned the “enterprise” level blockchain technology. A joint project by the Linux Foundation and IBM, Hyperledger is an ecosystem of blockchain aimed to reduce the time and cost constraints brought on by legacy centralized databases.

All blockchains aim to provide irrefutable proof that a set of transactions occurred between participants but do so in two different formats. Hyperledger is known as a “permissioned blockchain”, where Ethereum, Bitcoin, and Storj are considered “permissionless” blockchains. For those who have been interested in cryptocurrencies, they would quickly point out that Ethereum and Bitcoin are exactly that. To run smart contract code on blockchains like Ethereum or Bitcoin, you must pay to operate that code in that blockchain’s cryptocurrency. This means that anyone with access to that cryptocurrency can transact on that blockchain. This also makes processing power quite costly to run as literally every node in that network must not only “agree” to that transaction, but the processing power required to validate (“mine”) a transaction grows exponentially as to its solvability. This is the same format used by Storj and their “farmers” to validate/provide storage capacity and bandwidth.

Hyperledger on the other hand, has no native cryptocurrency and does not require any other outside resource to validate transactions. All participants must have a predefined notion of what assets are being shared operating on a need-to-know basis. For those businesses (i.e. finance) required to keep compliance for things like Know-Your-Customer (KYC), this is important. Any endorsement (addition to the blockchain) follows an agreed upon set of rules, also known as “chaincode”. Only those who are a part of the network validate/process the transaction sent to the fabric. Each node (alternatively called “peers”) runs a series of processes to reach that consensus required to validate new transactions. The below diagram and description of Hyperledger processing is adapted from the IBM and Cognitive Class.ai *Blockchain Essentials* course(14):



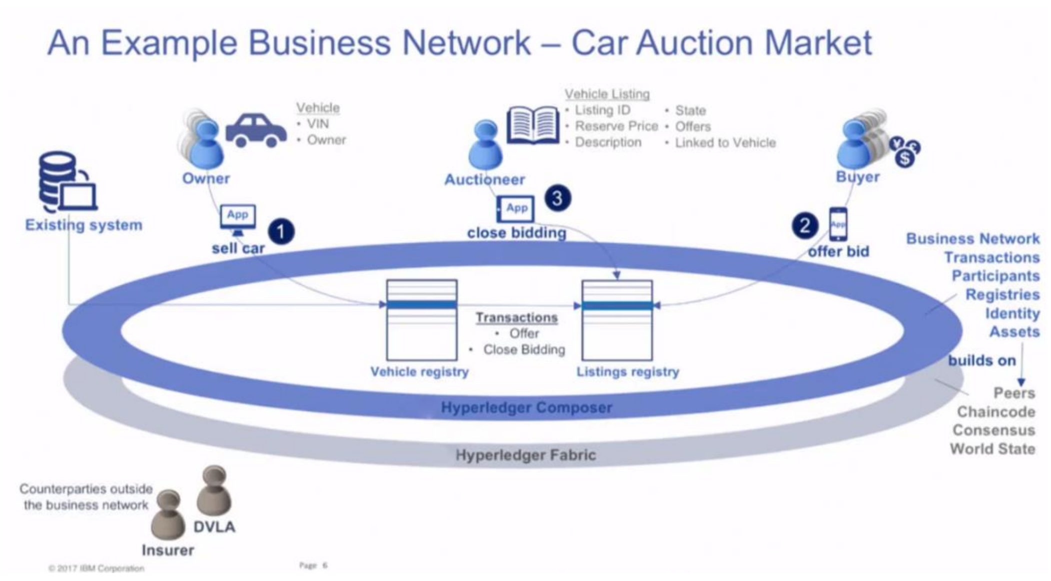
The process starts by a client application proposing a transaction. Each node is given an Endorsement Policy which defines who can do what and who is able to endorse a specific transaction(14). Next, the application sends an execution request to an Endorsing Peers (identified in the blue E squares above). These isolated endorsing processes do a pre-validation of the transaction but make no updates to the shared ledger. Each execution captures the set of Reads and Writes (properly named RW Sets) to be re-checked later in the process for consensus(14).

A proposal response is sent back to the application with a specialized signature confirming the request and the application responds to another isolated process named the Ordering Service (identified as the green O squares above). The Ordering Service will take the signed request from the application and queues the responses to be ordered on the ledger. The Ordering Service collects transactions into proposed blocks for distribution to committing peers. Peers deliver the transaction to other peers and the transaction are ordered via three different algorithm types: SOLO (for single node/development environment), Apache Kafka, and SBFT (Byzantine Fault Tolerance)(14). Regardless of ordering algorithm used, once ordered, every committing peer validates against the endorsement policy and validated transactions are saved to what is known as the “world state” and the ledger. The world state is the final shared database among all nodes in the fabric-tracking; the final point of truth. The ledger tracks all transactions, whether valid or not. Therefore, invalid transactions are retained on the ledger but not the world state. Finally, the application can register the success or failure of the commit it originally sent by each peer in the fabric. This is what certifies data protection, validity, and redundancy in a Hyperledger deployment. This full process usually runs on a virtual machine (VM) and applications read/write/query to an endpoint running on that VM. Hyperledger Fabric, as many processes running on a VM usually are, can be containerized. This means Hyperledger Fabric can be easily integrated into existing processes that are running on containers.

As ECS and many other S3-related object storage solutions are simply a batch of containers running such processes, it would be theoretically, something like the proceeding segment.

## ECS Meet Hyperledger

This section will visualize the theory presented through screenshots of Hyperledger Composer. While it is (for whatever reason) depreciated as of August 2019(15), Hyperledger Composer can still be used through the web portal (called Hyperledger Composer Playground(16) or provisioned on GCP in a pre-loaded instance is the best tool to practice, code, and deploy a Hyperledger blockchain network. Through a Hyperledger Business Network (output as a Business Network Archive, BNA, or .bna file), a developer codes four components of a Hyperledger Network to create a full ecosystem of processes. The below illustrates all the pieces and procedures that make up a Hyperledger Business Network presented from the *CognitiveClass.ai* course on the essentials of Hyperledger blockchain(14):

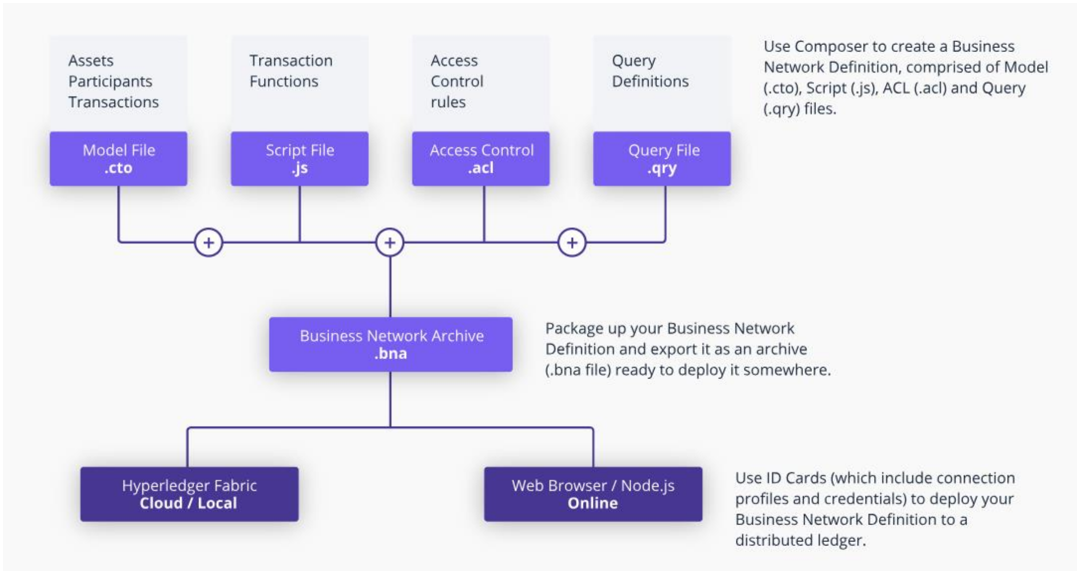


This section will demonstrate each component of a Hyperledger Composer .bna file which will generate a Business Network simulating how Hyperledger can be used in conjunction with object storage technologies. We will create an example file as well as draw from examples within the Hyperledger community to explain the idea.

Best practice dictates that blockchain is **not** used as the database (or bucket in this case) to store files. As written about extensively by James Worthington in a March 2017 WordPress blog, while it is technically possible to transform an image into a base64 image string (a format that can be stored on Hyperledger's NoSQL {CouchDB} database(17)), it would be impractical to store such unique and complex set of strings that would begin to aggressively to consume capacity, especially in enterprise use cases. Instead, what can be done is to create a unique value (i.e. hash of the file being stored) and simply storing that string and any associated metadata of that object. The benefit is to take advantage of the innate data replication functionality to certify the object (either stored somewhere off-chain or in the case of ECS, processes/containers running in the Storage Service layer) and more notably – it's metadata – is stored in a cryptographically secured fashion. This provides verifiable proof that the source is the exact file that was securely hashed at the time the application sent the request for it to be stored.

## Defining the ECS Distributed Ledger

Constructing the Business Network in Hyperledger Composer is done through coding of four components(14):


Assets
Participants
Transactions
Transaction
Functions
Access
Control
rules
Query
Definitions
Use Composer to create a Business Network Definition, comprised of Model (.cto), Script (.js), ACL (.acl) and Query (.qry) files.
Model File .cto
Script File .js
Access Control .acl
Query File .qry
Business Network Archive .bna
Package up your Business Network Definition and export it as an archive (.bna file) ready to deploy it somewhere.
Hyperledger Fabric Cloud / Local
Web Browser / Node.js Online
Use ID Cards (which include connection profiles and credentials) to deploy your Business Network Definition to a distributed ledger.

- **Models** are written using a domain-specific language and capture the schema/shape of the assets, participants, and transactions that are a part of the Business Network.

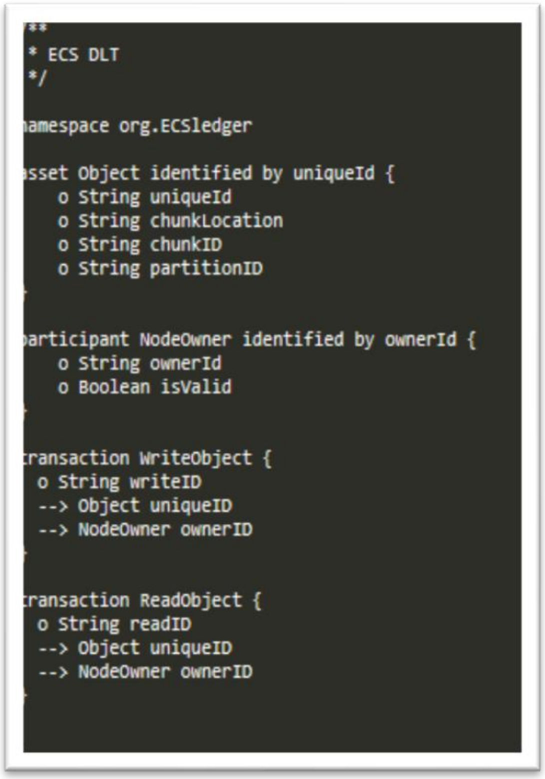
- **Script File** is the business logic (i.e. chaincode or “smart contract”) written in JavaScript and captured in script files. When a transaction is submitted, it is going to find the script functions that are interested in that transaction and run them. That can have side effects on assets in asset registries.

- **Access Control Lists (ACLs)** are declarative statements used to ensure the proper participants maintain the appropriate access.

- **Query Definition** set up actionable queries that can be made and by whom.

The Model, Script, and Access Control files are required to create a Business Network Archive file while the Query file is optional. Together, when packaged into the Business Network Archive, after following a few commands to set up a Linux VM (regularly Ubuntu) with the proper libraries and packages, a developer can use this file to deploy the Business Network Definition as the logic for the distributed ledger. This deployment is done on premises or in a public cloud deployment.

The Model of the Business Network Definition is the first portion to code and is packaged into what is known as a Hyperledger CTO file (.cto). In it, a developer codes how *participants* interact with *assets* through *transactions* and *events*. These are classes saved to a single system namespace which contains the definitions of asset, participant, transaction, and event. In the system namespace definitions, asset and participant have no required values, while events and transactions are marked by a unique identifier such as an eventId or transactionId and a timestamp(18).

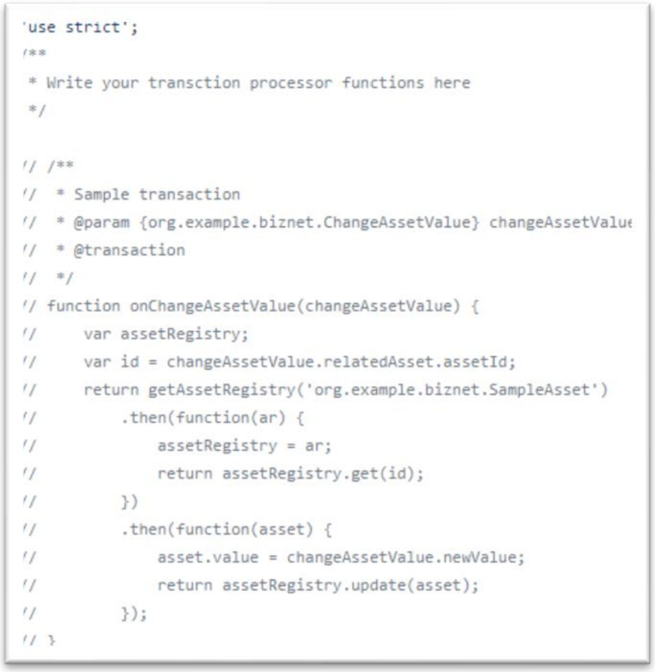


```
**
* ECS DLT
*/

amespace org.ECSledger

sset Object identified by uniqueId {
   o String uniqueId
   o String chunkLocation
   o String chunkID
   o String partitionID
}

articipant NodeOwner identified by ownerId {
   o String ownerId
   o Boolean isValid
}

ransaction WriteObject {
 o String writeID
 --> Object uniqueID
 --> NodeOwner ownerID
}

ransaction ReadObject {
 o String readID
 --> Object uniqueID
 --> NodeOwner ownerID
}
```

In the following code snippet, we define an identifier for the single namespace. Next, the asset named “Object” is committed to the blockchain with a unique identifier. This uniqueID is denoted with many “String” elements. These elements can be many different types of data formats, i.e. Doubles, Boolean, etc. In this case, the uniqueID could be a generated hash or even the URL/link to the file stored off-chain. These elements are what make up identifiers used in the tables presented in the *How Objects are Organized in the Data Center* section of the article. These elements are best stored as strings and can easily be used to identify where the data of an object is residing on a cluster.

Next, we identify who will be the participants. It is important to identify that each node in the deployment have some unique identifier and should have some trigger to say whether it is a valid node that is part of the system. Finally, there are two “transactions” which allow for a client to read/write object data to the Fabric.

Next is the Script file, compiled as the logic.js file. This file can be quite extensive to write and is responsible for the majority of the "logical rules and tests" to be applied to the Fabric deployment; everything from validating the assets to defining rules to be executed. This requires extensive coding and interactivity with the complete ECS Software Stack.

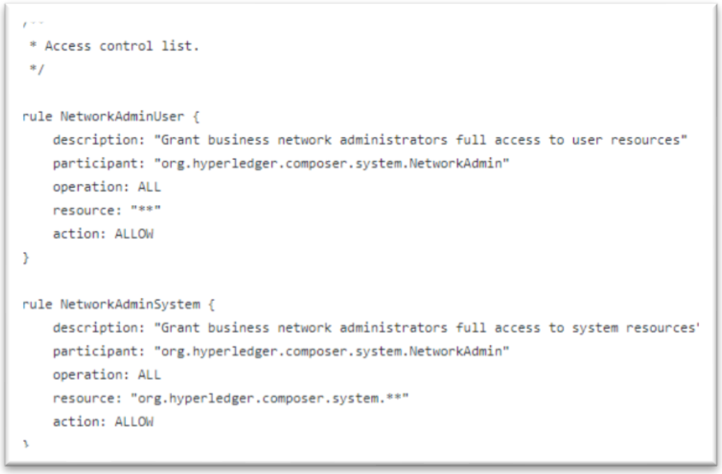


```
'use strict';
/**
 * Write your transction processor functions here
 */

// /**
//  * Sample transaction
//  * @param {org.example.biznet.ChangeAssetValue} changeAssetValue
//  * @transaction
//  */
// function onChangeAssetValue(changeAssetValue) {
//     var assetRegistry;
//     var id = changeAssetValue.relatedAsset.assetId;
//     return getAssetRegistry('org.example.biznet.SampleAsset')
//         .then(function(ar) {
//             assetRegistry = ar;
//             return assetRegistry.get(id);
//         })
//         .then(function(asset) {
//             asset.value = changeAssetValue.newValue;
//             return assetRegistry.update(asset);
//         });
// }
```

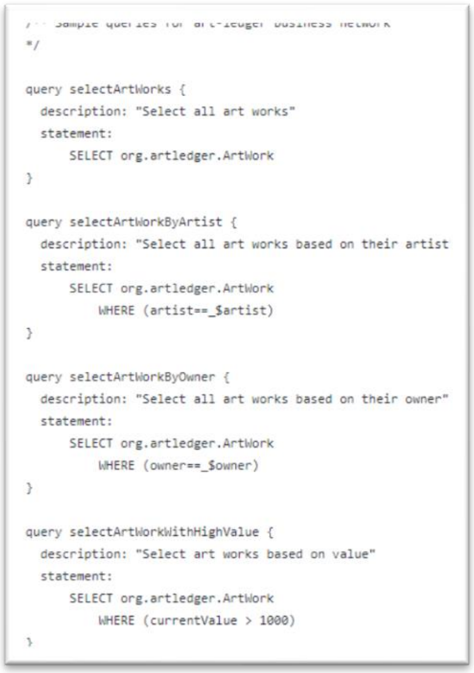
The following code snippet is an example of a very basic logic.js file adapted from a file storage GitHub repo on the topic of blockchain and file storage(19).

Seen here is a simple transaction function. In this case, when an asset is transacted, all attributes (i.e. assetID), what namespace it is in, and what new, validated information need to be committed to the append-only ledger.

Finally, the Access Control List and the Query files act as the gatekeeper to the Hyperledger code and the query rules that a developer can set to recall permissioned data deployed onto the chain. The below code are great examples from an "Art Ledger" sample code(20). What is nice about this example is that it is keeping the same code design principles that we would use to integrate with ECS.



```
/**
 * Access control list.
 */

rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system resources'
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```



```
/** Sample queries for art-ledger business network
*/

query selectArtWorks {
  description: "Select all art works"
  statement:
      SELECT org.artledger.ArtWork
}

query selectArtWorkByArtist {
  description: "Select all art works based on their artist
  statement:
      SELECT org.artledger.ArtWork
          WHERE (artist==_$artist)
}

query selectArtWorkByOwner {
  description: "Select all art works based on their owner"
  statement:
      SELECT org.artledger.ArtWork
          WHERE (owner==_$owner)
}

query selectArtWorkWithHighValue {
  description: "Select art works based on value"
  statement:
      SELECT org.artledger.ArtWork
          WHERE (currentValue > 1000)
}
```

On the left, the Access Control List is only a few lines of code. Developers create rules for the Hyperledger Business Network. The above grants administrators of the network access to all system and user resources. On the right is a sample query file in which a developer defines what exactly can be queried and how.

In summary, an inclusion of just a few lines of containerized code can fundamentally optimize the way in which objects are stored on technologies like ECS with efficiency and security. While the capabilities to store files in a complex binary string is possible, it is not practical for enterprise-grade solutions. The benefits of a permissioned blockchain managing the integrity of the *metadata only* in a cluster is that it can work in tandem with the Storage Service layer of the ECS Software stack. Each would have their specific microservices to be responsible for but would contribute together to the higher optimization of the platform overall.

The next section explores how this idea can be optimized further through machine learning.

## Leveraging Machine Learning to make Blockchain Smarter

Latency and bandwidth are always top concerns for application developers. Blockchain technology is notorious for being ambiguous regarding how the technology will perform as more data is stored on it and it matures. To always be ahead, we propose an architectural pattern idea that integrates machine learning with blockchain technology. Architectural patterns for artificial intelligence-based applications and services keep evolving with regard to optimizations in scalability, fault tolerance and data privacy. We propose an architecture which integrates the deployment of machine learning and deep learning models with a permissioned blockchain implementation.

A permissioned blockchain serves the purpose of storing model storage metadata and parameters which are essential in providing machine learning use cases such as predictions or recommendations. With the use of a permissioned blockchain, storage metadata can be versioned like any version control system and preserved in a shared ledger while anonymizing the origins of the assets used for model training and metadata creation. This allows the consumers of the blockchain-based system to have complete privacy of the data.

The shared ledger tracks parameters associated with model storage metadata as well as the model itself. Each of the model components such as the model graph, storage metadata and individual parameters such as hyperparameters are isolated for a specific entity which would be the primary asset having the machine learning use cases. The primary asset here can refer to any data that can be associated with a tangible entity in a real-world use case such as application history and usage statistics of a user, financial metrics of a specific company, operational data of a firm and so on. The metrics and parameters modelled based on the asset mentioned will be isolated as separate entities on the shared ledger and will be completely anonymized.

Based on the metrics derived from an asset's parameters, the model itself is updated as a record in the ledger and henceforth creates an additional version. This essentially creates additional versions of the central assets and allows for tracking of a changelog of the asset.

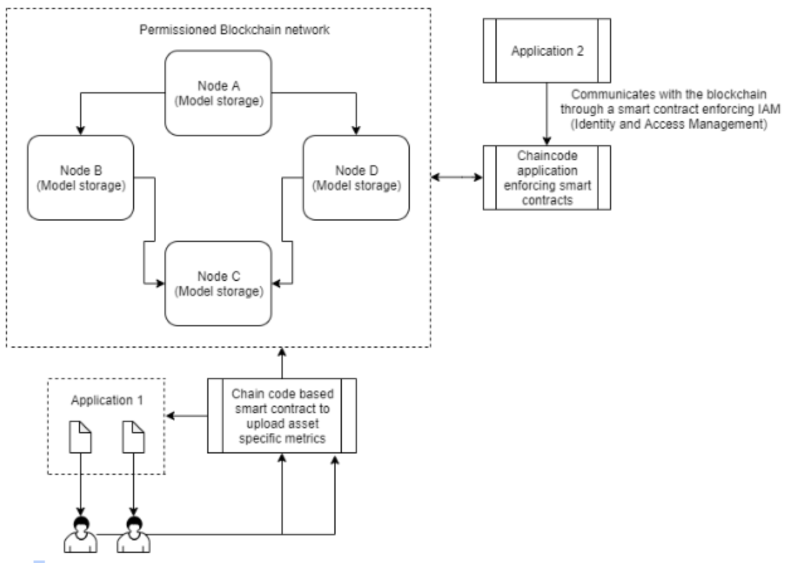
These changelogs tracked model parameters are offered to be ingested for better recommendations and metrics by third party applications through an IAM-based interface.

The IAM-based interfacing creates a permission layer for other applications to interact with the blockchain to retrieve model versions for an asset.

Identity Access Management (IAM) is responsible for granting the proper access to applications trying to use the blockchain and version it on a permissioned blockchain implementation such as Hyperledger fabric. This enables application owners to have a changelog tracked and versioned history of models which relates to changes in an asset's parameters which could be usage history, operational data or financial metrics based on the type of asset. This provides a deeper insight in assessing correlation in the different versions of the asset.

Additionally, this architecturally allows for models to be built from an aspect of the data that was being used by the asset rather than focusing on the asset itself. To illustrate this use case, consider two users who have different shopping habits and preferences which are transacted and added into the shared ledger on a permissioned blockchain. In this case, the two users are the assets under consideration and the important metrics for individually modelling the asset parameters such as shopping habits and preferences are the parameters themselves. Thus, the models versioned and tracked through a changelog will be developed for shopping habits and preferences and not for users. The models for each asset will be versioned independent of each other and correlations in the version changes will be done individually. Since this is on a permissioned blockchain such as Hyperledger, the models and version, though independent of each other at this point, cannot be tracked or associated to the original users, hence preserving anonymity and privacy.

Furthermore, Applications subscribing to the permissioned blockchain can get individual versioned and change tracked models for each independent model specific entity and perform larger mining and analysis based on changes in the model version history for the two entities.


Permissioned Blockchain network
Node A (Model storage)
Node B (Model storage)
Node D (Model storage)
Node C (Model storage)
Application 2
Communicates with the blockchain through a smart contract enforcing IAM (Identity and Access Management)
Chaincode application enforcing smart contracts
Application 1
Chain code based smart contract to upload asset specific metrics

## Summary

Integrating blockchain technology into object storage solutions (i.e. ECS) can address the growing data problem. For any tech savvy organization, having to manage and deploy infrastructure to wrangle and massage value out of the data is cost-prohibitive. While alternatives such as public cloud and blockchain-based storage solutions seem tempting at such a low cost, they are still very much unproven. For blockchain-based object storage solutions i.e. Storj, the use of cryptocurrency and it being a public (un-permissioned) blockchain does not fit well with more enterprise-level requirements.

Instead, this article posits that integrating a containerized instance of a permissioned blockchain like Hyperledger Fabric into the ECS Software stack will change how IT organizations of businesses manage object storage. With just a few simple lines of code, we are able to theoretically have the foundation for a cryptographically secure, orchestrated (through chaincode/“smart contracts”), and automated network to optimize metadata read/writing/searching on the single namespace. Inherent data replication and storing unique strings (of object metadata) in a small data format makes it more efficient than it’s triple-mirrored counterpart deployed in ECS and other object storage solutions today. Nodes enabled with the proposed blockchain services would expand the ECS solution beyond being a platform for S3 buckets into one that is pioneering movement toward decentralized cloud technologies. Finally, this article addressed how this idea could be optimized further through machine learning.

As our data grows, so does our desire to understand it. Bringing these two technologies together enables us to do so more efficiently – clearing the pathway forward for unimaginable human progress.

## Appendix

(1)Dell EMC Whitepaper (September 2019), "ECS Overview and Architecture", page 12-13, September 2019, https://www.dellemc.com/en-be/collaterals/unauth/white-papers/products/storage-1/h14071-ecs-architectural-guide-wp.pdf

(2)Dell EMC Whitepaper (September 2019), "ECS Overview and Architecture", page 9, September 2019, https://www.dellemc.com/en-be/collaterals/unauth/white-papers/products/storage-1/h14071-ecs-architectural-guide-wp.pdf

(3)Dell EMC Whitepaper (September 2019), "ECS Overview and Architecture", page 15-16, September 2019, https://www.dellemc.com/en-be/collaterals/unauth/white-papers/products/storage-1/h14071-ecs-architectural-guide-wp.pdf

(4)Dell EMC Whitepaper (September 2019), "ECS Overview and Architecture", page 17-18, https://www.dellemc.com/en-be/collaterals/unauth/white-papers/products/storage-1/h14071-ecs-architectural-guide-wp.pdf

(5)MarkNetwork Blockchain (2019), "Will blockchain disrupt Cloud 2.0?", https://www.mark-network.com/will-blockchain-disrupt-cloud-2-0/

(6)Nead, Nate & Garnett, Connor (2019), "Blockchain's Next Frontier: Cloud Computing?", https://investmentbank.com/blockchain-cloud-computing/

(7)Crunchbase (2017), "Storj Labs: Total Funding Amount", https://www.crunchbase.com/organization/storj

(8)Wilkinson, Shawn (2014), "Storj- A Peer-to-Peer Cloud Storage Network", page 2-3, https://storj.io/storj2014.pdf

(9)Wilkinson, Shawn (2014), "Storj- A Peer-to-Peer Cloud Storage Network", page 4-5, https://storj.io/storj2014.pdf

(10)Wilkinson, Shawn (2014), "Storj- A Peer-to-Peer Cloud Storage Network", page 6, https://storj.io/storj2014.pdf

(11)Wilkinson, Shawn (2014), "Storj- A Peer-to-Peer Cloud Storage Network", page 8-9, https://storj.io/storj2014.pdf

(12) Mearian, Lucas (2019), "Blockchain-based storage service takes on Amazon AWS, unveils pricing", https://www.computerworld.com/article/3454365/blockchain-based-storage-service-takes-on-amazon-aws-unveils-pricing.html

(13) BlockApps (2017), "How Blockchain Will disrupt Data Storage", https://blockapps.net/blockchain-disrupt-data-storage/

(14) Cognitive Class (2019), "Blockchain Essential: Course", https://courses.cognitiveclass.ai/courses/course-v1:developerWorks+BC0101EN+v1/info

(15)Hyperledger Composer (2019), "Hyperledger Composer: About", https://www.hyperledger.org/projects/composer

(16)Hyperledger Composer (2019), "Navigating Playground: The Business Networks page", https://hyperledger.github.io/composer/v0.19/playground/playground-index

(17) Worthington, James (2017), "Storing Images in Hyperledger Fabric (Blockchain)", https://belltane.wordpress.com/2017/03/27/storing-images-in-hyperledger-fabric-blockchain/

(18)Hyperledger Composer (2019), "Hyperledger Composer Modeling Language", https://hyperledger.github.io/composer/v0.19/reference/cto_language

(19)Preetam (2018), "GitHub: hyperledger_composer_file_storage", https://github.com/Preetam007/hyperledger_composer_file_storage

(20)Cripps, Peter (2019), "Art-Ledger: IBM Hyperledger Project", https://github.com/petercrippsIBM/art-ledger/blob/master/