

How to organize an

Azure Data Lake



*Prepared for*

Smithfield

11/12/2018

Version 0.2.1

*Prepared by*

Christian Hamson

Cloud Solution Architect Christian.hamson@microsoft.com



MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, our provision of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The descriptions of other companies’ products in this document, if any, are provided only as a convenience to you. Any such references should not be considered an endorsement or support by Microsoft. Microsoft cannot guarantee their accuracy, and the products may change over time. Also, the descriptions are intended as brief highlights to aid understanding, rather than as thorough coverage. For authoritative descriptions of these products, please consult their respective manufacturers.

© 2018 Microsoft Corporation. All rights reserved. Any use or distribution of these materials without express authorization of Microsoft Corp. is strictly prohibited.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

ii

How to organize an Azure Data Lake

Revision and Signoff Sheet

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
| 2018-01-03 | Jan Cordtz | 0.1 | Draft version |

2018-11-12 Christian Hamson 0.2.1 Included information from Chad Gronbach, Chief Technology Architect, Added ADLS gen 2 information, Added scrubbed information from previous Data Lake deployment.

Reviewers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name |  | Version Approved | Position |  | Date |

Table of Contents

[1 Summary vi](#_Toc529712041)

[2 The philosophy behind the Azure Data Lake organization 7](#_Toc529712042)

[3 Decisions to be made 10](#_Toc529712043)

[4 Data Lakes for Dev/Test and Prod 12](#_Toc529712044)

[5 The internal organization of the Azure Data lake 13](#_Toc529712045)

[5.1 Logical Zones 13](#_Toc529712046)

[5.1.1 The Raw Zone 13](#_Toc529712047)

[5.1.2 The Landing Zone 14](#_Toc529712048)

[5.1.3 The Curated (Gold, Publish) Zone 14](#_Toc529712049)

[5.1.4 The Staging Zone 15](#_Toc529712050)

[5.1.5 The dev and test zones 15](#_Toc529712051)

[5.1.6 The Analytics Sandbox Zone 15](#_Toc529712052)

[5.1.7 The Archive Zone 15](#_Toc529712053)

[5.2 File System Hierarchy 15](#_Toc529712054)

[5.3 The Data movement process within the Lake 16](#_Toc529712055)

[5.4 Archiving 17](#_Toc529712056)

[6 The security model of Azure Data Lake 18](#_Toc529712057)

[6.1 Access Control 18](#_Toc529712058)

[6.2 ACLs and POSIX 18](#_Toc529712059)

[6.3 Encryption 18](#_Toc529712060)

[6.21 Encryption in Transit 18](#_Toc529712061)

[6.22 Encryption at Rest 18](#_Toc529712062)

[6.3 Vnet Service Endpoint (Confirm that this is V2 ready) 18](#_Toc529712063)

[7 The Data Ingestion process 19](#_Toc529712064)

[8 DevOps setup 21](#_Toc529712065)

[9 Data Catalog 21](#_Toc529712066)

[10 Logging 21](#_Toc529712067)

[10.1 Enable diagnostic logging for your Data Lake 22](#_Toc529712068)

[10.2 View diagnostic logs for your Data Lake 23](#_Toc529712069)

[ Using the Data Lake Store Settings view 23](#_Toc529712070)

[ From the Azure Storage account that contains log data 24](#_Toc529712071)

[10.3 Understand the structure of the log data 25](#_Toc529712072)

[10.3.1 Request logs 25](#_Toc529712073)

[10.3.2 Audit Logs 27](#_Toc529712074)

[11 Example of an Azure Data Lake setup 28](#_Toc529712075)

[11.1 Create the Data Lake 28](#_Toc529712076)

[11.1.1 Create Directories 28](#_Toc529712077)

[11.1.2 Assign access rights 28](#_Toc529712078)

[11.2 Create the Data Factory service 28](#_Toc529712079)

[11.3 Use PowerBI to see data 28](#_Toc529712080)

[12 References 29](#_Toc529712081)

# 1 Summary

This document discusses a possible way to organize an Azure Data Lake

The content is based on the different Azure Data Lake implementations the authors have either witnessed or worked on.

The document focus is on Azure Data Lake. Some discussion of data movement tools and orchestration tools, specifically Databricks and Data Factory will be included. Other technologies will only be discussed briefly. These include SQL DB, SQL DW, Analysis Services, PowerBI, etc.

This document will need to be updated as the features of Azure Data Lake Store Generation 2 are made generally available.

# The philosophy behind the Azure Data Lake organization

The main drivers behind this approach is to have an Azure Data Lake setup that support the following:

* Is as cost-effective as sensible/possible.
* Do not compromise security.
* Fits well into a DevOps scenario
* Have a well-defined path for the information needed to be able to support an effective auditing and logging process.

The overall setup looks like this:

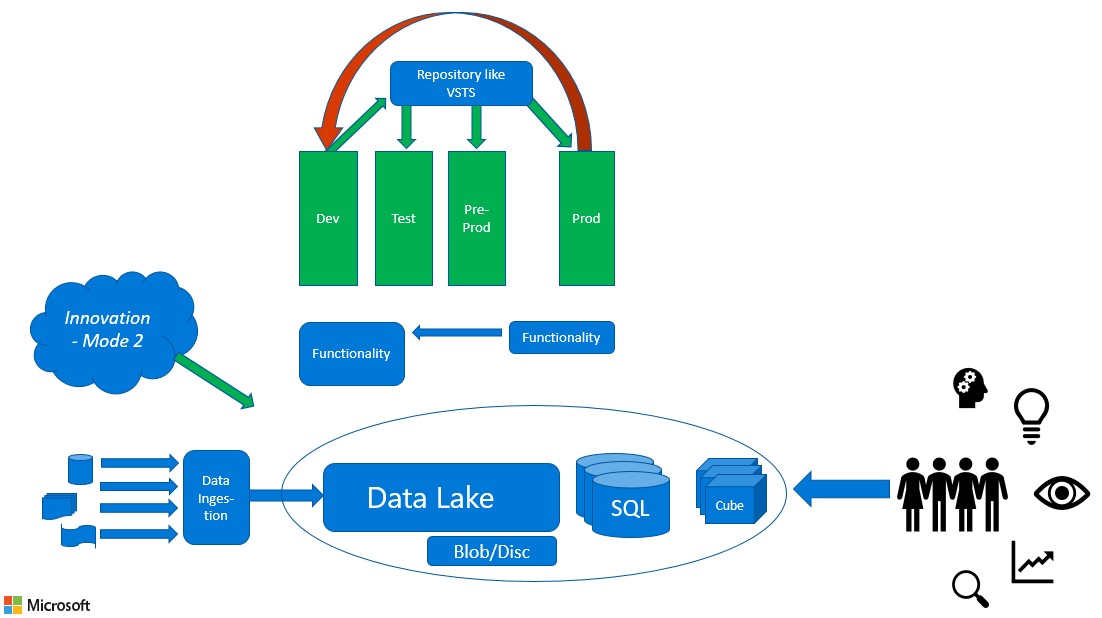


Figure 1

In the circle we have the different data layers we could use. The data layer is ingested by some “copy data” mechanism – shown here as a “Data Ingestion”. On the right side the users can use any appropriate tool to use data found in the data layer. Also, any new/strange/external/foreign data can be ingested in to the data layer – shown as the *innovation cloud*.

Any development, be it reports as well as applications will live under a DevOps regime, even the innovative ones, to assure that any object that needs to be used by a wider crowd of people is handled as a traditional production object, so that support and maintenance is not *sacrificed* in this process. In chapter 7 a further discussion about the usage and requirements for a DevOps scenario is addressed.

The philosophy is overall to have the appropriate functionality in place in the different aspects of the setup and not “just” try to make everything fit in to a relational database.

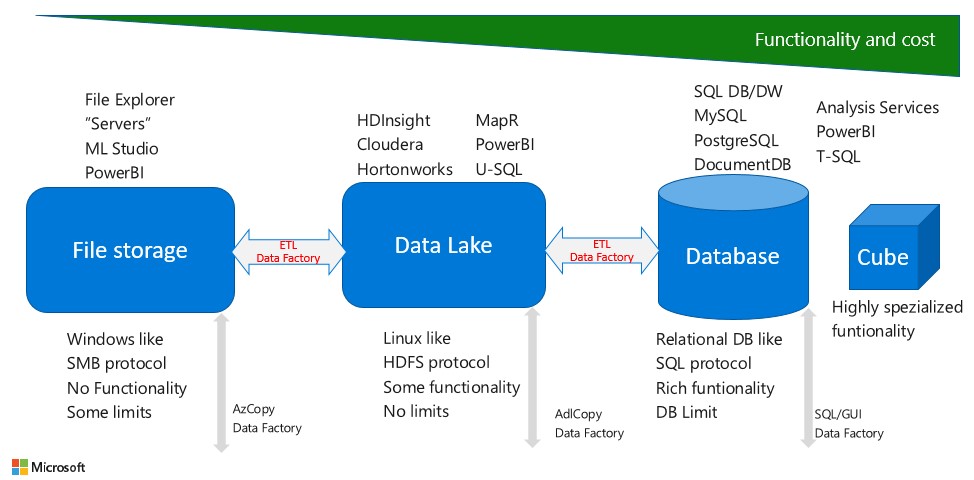


Figure 2

On the left of Figure 2 we have a “*normal*” file storage, that is a file storage as we know it from the traditional Wintel/Lintel servers and laptops. This storage can be used in direct connection with a Windows or Linux based system as it is easy to attach/mount. So, relating that to an Azure Data Lake setup where we often need a place where files can be put for easy access, this part of the setup will serve this purpose as an (often) intermediate storage layer.

In the middle we have the Azure Data Lake storage, which in this setup is our main storage area. The Azure Data Lake is both the source for any relational databases and cubes to be implemented as well as a storage layer that will be used directly by the end-users. The Azure Data Lake naturally serves as “file system” for the *Hadoops* like Databricks, HDInsight, Cloudera, Hortonworks etc.

On the right side we have the traditional relational databases, which is used for mainly two things in this setup. One is to get access to a full SQL implementation like T-SQL, secondly due to the inherited *Quality Assurance* provided by the nature of tables – you can only store things that the database is designed for – the relational database provides us a storage layer for data that needs to be under some kind of *strict control*.

Also, we will here find the *Cubes*, which very often is an extension of a relational data warehouse providing some very interesting yet specialized analytical capabilities.

# Decisions to be made

Every Data Lake has multiple decisions that need to be made to create standards. Without standards a Data Lake will become a Data Swamp. Below is a list of decisions that may need to be made to successfully implement a Data Lake. This document provides context for those decisions. Many of these decisions should follow from existing policies and standards within an organization. Below is a non-comprehensive list of some of the key decisions that should be made explicitly.

1. Underlying service – Microsoft recently introduced a new version of Data Lake Store, Azure Data Lake Store gen 2. This service is tentatively scheduled to enter General Availability in early 2019. Choices now are 1) Data Lake gen 2 (public preview), 2) Blob storage, and 3) Data Lake gen 1. We strongly recommend not using Data Lake gen 1 for new Data Lakes. The choice between Blob and gen 2 is often made based on volume of data. Gen 2 is based on blob storage and it allows you to interface with your data using both file system and object storage paradigms.
2. Data Lake Organization Decisions
   1. How many Data Lakes
      1. Separate Lakes for Dev, QA, and Prod or one Lake
      2. Separate Data Lakes at extremely high organizational level. For example, one each for Marketing, Research and Development, and Operations. This decision is often deferred.
   2. What are the high-level logical zones of the Data Lake
   3. What should the File System Hierarchy be?
   4. What is the file naming convention?
3. Security Decisions
   1. Access Control – Should follow from security policies and standards within the organization. Includes best practices around groups, users, service principals, tooling for access control (Azure Active Directory), and organizational procedures and governance.
   2. Zone, Folder, and file level access and authorization.
   3. Encryption, customer managed keys or Azure managed keys
4. Operational Decisions
   1. Lake Deployment method. Should be driven by, and conform to, organization dev/ops methodology and tooling. A typical Enterprise standard is use power shell or Azure CLI, with Azure resource manager templates from some IDE such as visual studio, not the Azure portal.
   2. Diagnostic and access logging. Often Enterprises have an existing tooling solution that Data Lake logging is expected to integrate with.
   3. Associated tooling – Associated tooling is often driving by the Enterprise service catalog with a strong bias towards previous approved services and tools.
      1. Data movement and processing orchestration (Azure Data Factory)
      2. Data movement and processing tools and compute contexts (Databricks, SSIS, Integration Run Time)
      3. Data access tools, including ad hoc data access tools that use the Data lake directly
      4. Non-data lake storage directly related to Data Lake (for example for logs)
      5. Archiving, tooling and processes
   4. Support model
5. Governance
   1. Data Governance
      1. Metadata management and tooling
   2. Change control process

# Data Lakes for Dev/Test and Prod

Often there are some discussions regarding how many data lakes you should implement. We often meet the following scenarios. This document focuses on scenario B

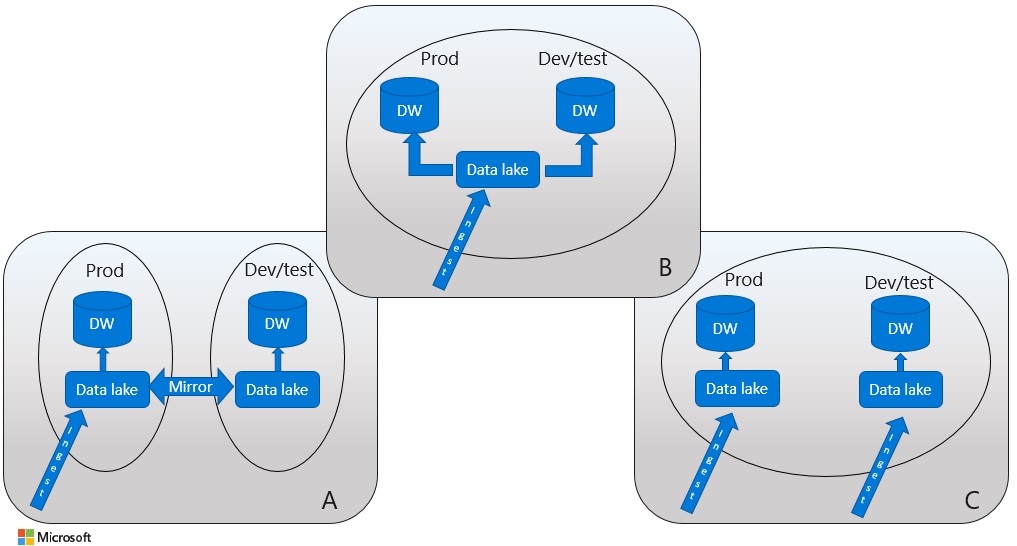


Figure 3

**Scenario A**: 2 Data Lakes, one for production and one for test/dev environments. The Data Lake in the production is where the data ingestion is done. This Data Lake is then mirrored to the dev/test Data Lake – mostly as a batch process – this accomplish the task of having identical data content for production and dev/test. The databases – symbolized by the DW identification – are then ingest with data from the Data Lake. The Dev/test often serves as failover environment for the production environment in this case.

**Scenario B**: 1 Data Lake – serving both the production and dev/test environment. Data ingestion is done to one place in this Data Lake. Data that is to be used for production purposes is then transported to a different area of this Data Lake. And data to be used for dev/test is also transported to a corresponding area - very often undergoing a process ensuring anonymization of the content - or some of the content.

**Scenario C**: The same as Scenario A, with the one difference that each of the Data Lakes has their own data ingestion process, thereby creating different systems for production and dev/test environment.

The different scenarios have their benefits, so it is important to have an open discussion about what fits the case in question.

# The internal organization of the Azure Data lake

As the focus is to use one Data Lake for Dev/Test and Prod the internal organization of the Data Lake must take care of the requirements of the different environments

Also, security must be addressed inside the Data Lake itself. This is discussed in chapter 5.

Figure 4 shows the logical zones discussed in this chapter.



Figure 4

## Logical Zones

### 5.1.1 The Raw Zone

Data ingestion is ingested to the Raw Zone as shown on Figure 4. Data ingestion is done using a Data Gateway approach. This implies that there are few channels in to the Data Lake. All data that needs to go to the lake is either brought to the Data Ingestion server/service. Or the Data Ingestion service pulls data from systems like databases and ftp servers.

Characteristics of data in the Raw Zone:

* Storage in native format for any type of data
* Exact copy from the source
* Immutable to change
* Typically append-only
* History retained indefinitely
* Extremely limited access to the Raw Data Zone – no operationalized usage
* Everything downstream from here can be regenerated from raw data

### 5.1.2 The Landing Zone

Some enterprises choose to use a Landing Zone upstream of the Raw Zone. A Landing Zone is useful when data quality checks or validation is required before the data is routed to the Raw Data Zone for retention (checksums, data coming from 3rd parties etc.) Many enterprises have some data that lands directly into the RAW zone and other data that lands in the landing zone.

From the Landing Zone data will go into the Raw Zone. Data does not need to be persisted in the Landing Zone.

As data in this part of the process will undergo transformations to suite their end-destination, the Data Lake will provide an area where these processes can store any intermediate results if needed. This area is called Work on Figure 4. No data will be “left behind” in this area, so it is important that the processes either clean up after themselves or an overall cleaning processes is established.

Also, no end-user access is provided to this area.

### 5.1.3 The Curated (Gold, Publish) Zone

Production data is stored in the area named Publish on Figure 3. This area is organized in a way that suits end-user access. Often data is stored on a “per department” level. It is important to notice that if a common element like a list of products is needed for “all departments”, there will be a copy of this under “each department”. Of course, you could establish a kind of “common department” area, but this is seldom done in praxis.

All data going to the Publish area is stored as read-only elements. If needed a read/write area “per department” or “per user” can be established. This is often the case for the users known as super-users because they often create things like reports for other users. For some users that need to add their own data to the data stored in the Publish area this read/write access can also be appropriate. But overall the need for read/write areas is often been kept at a minimum level.

### 5.1.4 The Staging Zone

Some enterprises choose to separate the curated data into that used for “self-service” BI, analytics, reporting, etc. from data used in other production applications. If so, the zone for data that is ready for these typically non-BI applications is the staging zone.

### 5.1.5 The dev and test zones

Data that is being used for dev/test environments is stored in dev and test zones. If there is a copy of the data lake for dev/test purposes these zones are not needed.

### 5.1.6 The Analytics Sandbox Zone

A workspace for data science and exploratory activities with minimal governance and standards (purposely undisciplined. Valuable efforts are “productionized” and “operationalized” to the Curated Data Zone. Not used for self-service, operationalized, purposes. User level write access to subset of this zone.

### 5.1.7 The Archive Zone

✓ An active archive ✓ Contains aged data offloaded from a data warehouse or other application ✓ Available for querying when needed (typically only occasionally)

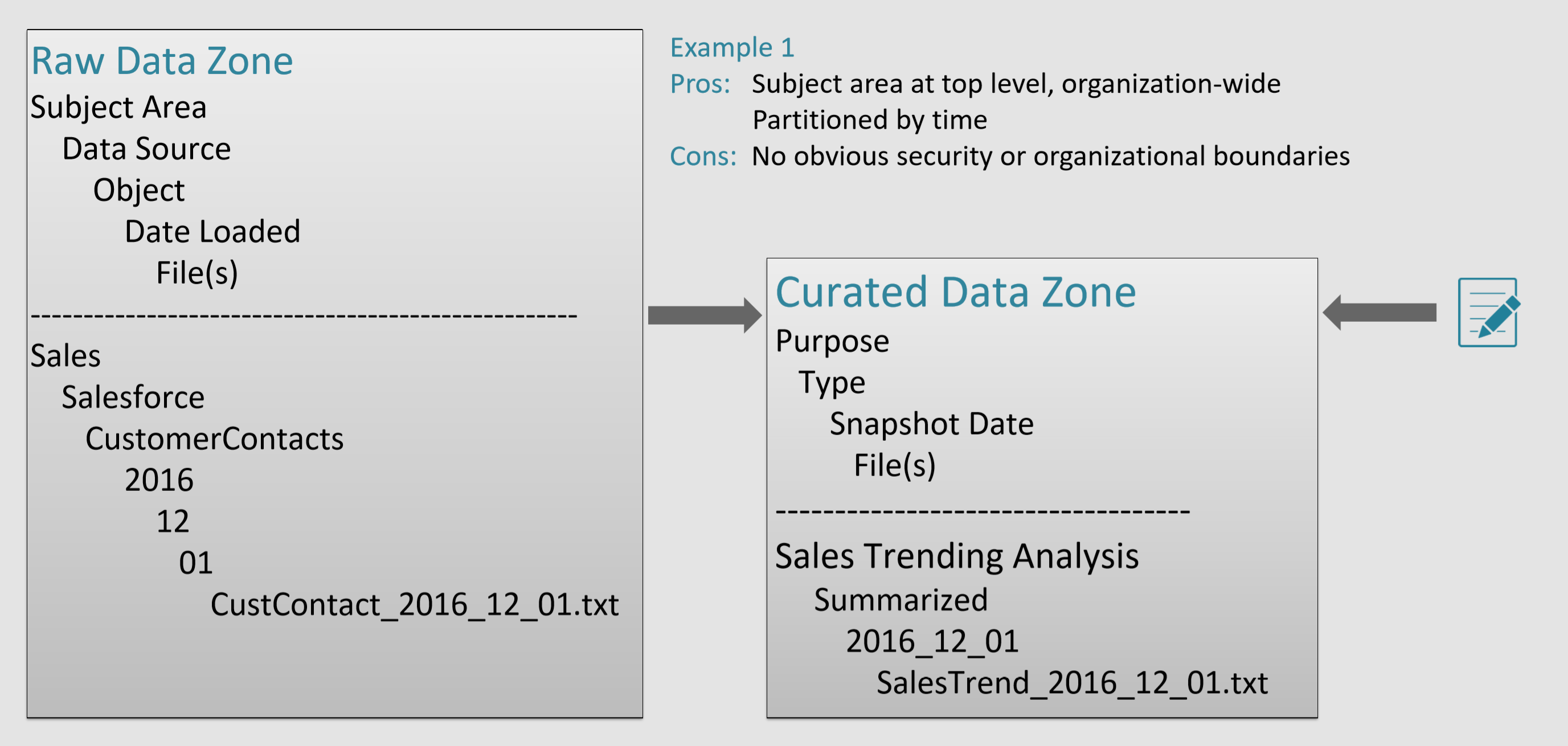
## File System Hierarchy

The file system hierarchy organizes the data and is central to Azure Data Lake design. The hierarchy should be straight-forward to use programmatically and be human readable. Considerations and best practices in designing a hierarchy include:

* Ease of data discovery & retrieval – will one type of structure make more sense?
* Focus on security implications early – what data redundancy is allowed in exchange for security
* Include data lineage & relevant metadata with the data file itself whenever possible (ex: columns indicating source system where the data originated, source date, processed date, etc)
* Include the time element in both the folder structure & the file name
* Be liberal yet disciplined with folder structure (lots of nests are ok)
* Clearly separate out the zones so governance & policies can be applied separately

Common parameters used to organize data include:

* Time Partitioning (Year, Month, Day, Hour, Minute)
* Subject Area
* Security Boundary (Department, Business Unit, etc.)
* Downstream Application / Purpose
* Retention Policy (Temporary, Permanent, Project Lifetime, Legal hold period…)
* Business Impact / Criticality (High, Med, Low, …)
* Owner / Steward / SME
* Probability of Data Access
* Confidentiality (Public, Internal Only, Supplier Confidential, PII, etc)

Of the many possibilities for a file hierarchy one is shown: 

Note that the organization is different across two zones. In the Raw zone Data Source is a partition, a backwards looking partition. In the Curated Zone a forward-looking partition, Purpose, is at a high level. Full file paths look like:

/<Zone>/<Subject Area>/<Data Source>/<Object>/<Date Loaded>/<Files(s)>

/raw/sales/salesforce/customerContacts/2018/01/01/custContact\_2018\_01\_01.txt

/<Zone>/<Purpose>/<Type>/<Date>/<File(s)>

/curated/salesTrendingAnalysis/summarized/2018/01/salesTrend\_2018\_01\_01.txt

## The Data movement process within the Lake

The processes taking data from the Landing Zone to the Publish and Analytics areas are mostly ordinary Extract-Transform-Load processes. Tools for this could be SQL Server Integration Services (SSIS), Databricks, Hive, etc. Even though such products often work with many different sources and targets, the only allowed source and target in this scenario is the Data Lake itself.

If data must travel further from the Publish or Analytics Area the ETL process often come in play again, but in this case of course there is only one restriction on source, that is the Data Lake, whereas the targets can be any storage layer suitable for the end-purpose – very often being a relational database working as a Data Warehouse/Data Mart. The movement process can be orchestrated by Azure Data Factory.

## Archiving

Data in the Landing Zone can (should) be archived when Data Lake functionality is no longer required – please refer to Figure 2. Archiving can be done to Azure Blob storage The Azure Blob storage offers four levels of storage: Hot, Warm, Cold, and Archive. Or within the Lake itself with four levels of storage and geographic redundancy. Orchestration would be performed by Azure Data Factory

The usual considerations for Archiving data apply. Legal Holds, how often the data is used, data retention standards, etc.

# The security model of Azure Data Lake

## Access Control

To be able to handle the processes and structures described in chapter 4, a combination of identities, standard users, groups, superusers, and service principals are allowed access. On Figure 5 you will find a typical mapping of these identities to the logical zones.

## ACLs and POSIX

**Data Lake supports a superset of POSIX permissions:** The security model for Data Lake Gen2 supports ACL and POSIX permissions along with some extra granularity specific to Data Lake Storage Gen2. Settings may be configured through admin tools or through frameworks like Hive and Spark.

## Encryption

## Encryption in Transit

Data in Transit in Azure Data Lake Store is always encrypted by HTTPS. There is no option to disable this encryption. HTTPS is the only protocol support by the ADLS REST interfaces.

## Encryption at Rest

Data at Rest in Azure Data Lake Store is on-by-default transparent encryption. There is no option to disable this encryption. Upon account creation, a choice is made between service managed keys or customer managed keys. With either choice, the Master Encryption Key is secured in Azure Key Vault. All data written to the Azure storage platform is encrypted through 256-bit [AES encryption](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard), one of the strongest block ciphers available. All redundant copies of data are encrypted and all redundancy options are supported. Storage Service Encryption is FIPS 140-2 compliant.

## Vnet Service Endpoint (Confirm that this is V2 ready)

Storage accounts, including the Data Lake can be configured to allow access only from specific Azure Virtual Networks.

By enabling a [Service Endpoint](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-service-endpoints-overview) for Azure Storage within the Virtual Network, traffic is ensured an optimal route to the Azure Storage service. The identities of the virtual network and the subnet are also transmitted with each request. Administrators can subsequently configure network rules for the Storage account that allow requests to be received from specific subnets in the Virtual Network. Clients granted access via these network rules must continue to meet the authorization requirements of the Storage account to access the data.

# The Data Ingestion process

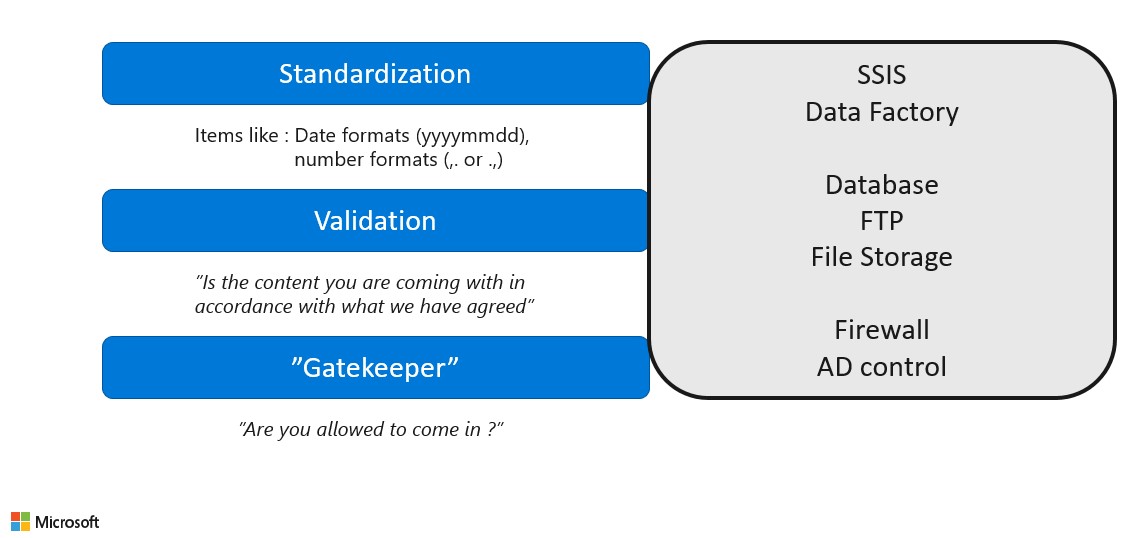


Figure 7

The overall principal for data in the Landing Zone is - as discussed - to store raw data. But to be able to control the ingestion of such raw data in to the Landing Zone the idea is to have a separate service – preferable on a dedicated server (on-premise or cloud based) – that handles 3 different aspects of this process.

The first part of the process (Figure 7) is a Gatekeeper function. Here it is controlled whether the process wanting to store data can enter the Landing Zone. This is often a question about user-access rights controlled by the Azure AD. An example could be that the Data Ingestion services provides an ftp-service and the process wanting to ingest data simply must be able to login to this ftp-service.

When access is granted the second part of the process is to validate that the data is actually in accordance with what is expected by the process trying to ingest data. It could be things like file-format, specific headers etc.

The last part of the data ingestion service – which is optional – is to do some standardization. Even though the overall principal is that the Landing Zone contains raw data it might be sensible/convenient at this stage to do some standardization on certain data formats, so that these objects are easier to use cross different data items (files).

It could be things like conform date formats and time formats (maybe a split of those) and number formats with a conform decimal separator and no thousands separator.

On Figure 7 a few examples of services are shown in the “grey box”, which could be used to handle the different functionalities of the Data Ingestion service.

In the gatekeeper function you will probably use a firewall and the Azure AD service to control the “login” process.

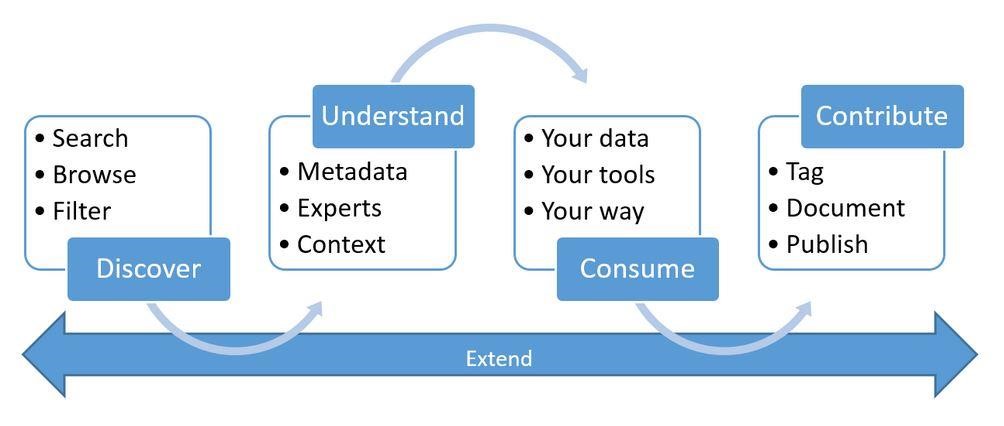
For the validation process you could use a database’s schema/table functionality to see if the data can actually be inserted. Also, you might have an ftp service and a file storage for objects that need other kind of validation.

And for the standardization process you could use Azure Data Factory to do this in the actual cope process – or – if you need a more advanced process SQL Server Integration Service (SSIS) can be used as a “real” ETL tool to accomplish this.

# DevOps setup

# Data Catalog

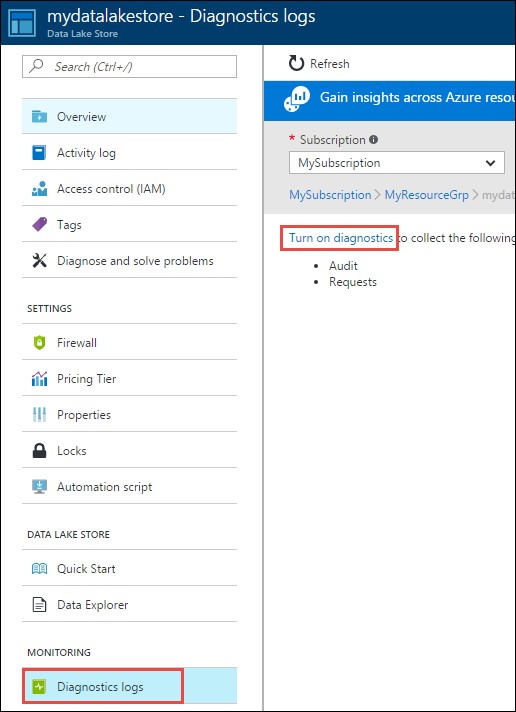
To be able to know what kind of data is present in the different areas in the Data Lake, who ones this data etc. it is imperative to document this. The goals of a Data Catalog tool and process are shown in Figure



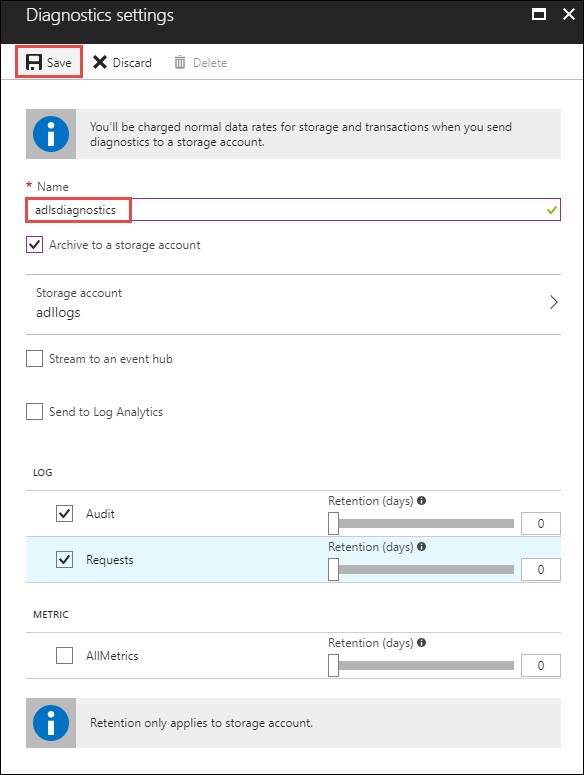
# Logging

You can enable diagnostic logging for the Data Lake to collect data access audit trails that provides information such as list of users accessing the data, how frequently the data is accessed, how much data is stored in the account, etc.

## Enable diagnostic logging for your Data Lake

Open your Data Lake, and from your Data Lake blade, click **Settings**, and then click **Diagnostic logs**.

In the **Diagnostics logs** blade, click **Turn on diagnostics**.

In the **Diagnostic** blade, make the following changes to configure diagnostic logging.

For **Name**, enter a value for the diagnostic log configuration.

You can choose to store/process the data in different ways.

Select the option to **Archive to a storage account** to store logs to an Azure Storage account. You use this option if you want to archive the data that will be batch-processed later. If you select this option, you must provide an Azure Storage account to save the logs to.

Select the option to **Stream to an event hub** to stream log data to an Azure Event Hub. Most likely you will use this option if you have a downstream processing pipeline to analyze incoming logs at real time.

Select the option to **Send to Log Analytics** to use the Azure Log Analytics service to analyze the generated log data. If you select this option, you must provide the details for the Operations Management Suite workspace that you would use the perform log analysis.

Specify whether you want to get audit logs or request logs or both.

Specify the number of days for which the data must be retained. Retention is only applicable if you are using Azure storage account to archive log data.

Click **Save**.

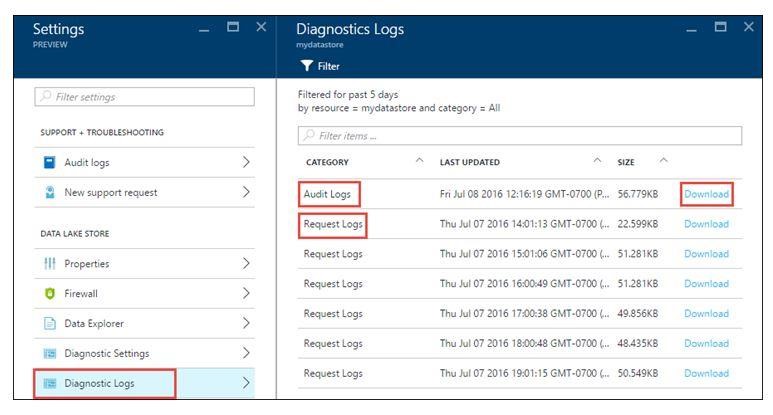
## View diagnostic logs for your Data Lake

There are two ways to view the log data for your Data Lake Store account.

* From the Data Lake Store account settings view
* From the Azure Storage account where the data is stored

### Using the Data Lake Store Settings view

From your Data Lake Store account **Settings** blade, click **Diagnostic Logs**.



In the **Diagnostic Logs** blade, you should see the logs categorized by **Audit Logs** and **Request Logs**.

Request logs capture every API request made on the Data Lake Store account.

Audit Logs are like request Logs but provide a much more detailed breakdown of the operations being performed on the Data Lake Store account. For example, a single upload API call in request logs might result in multiple "Append" operations in the audit logs.

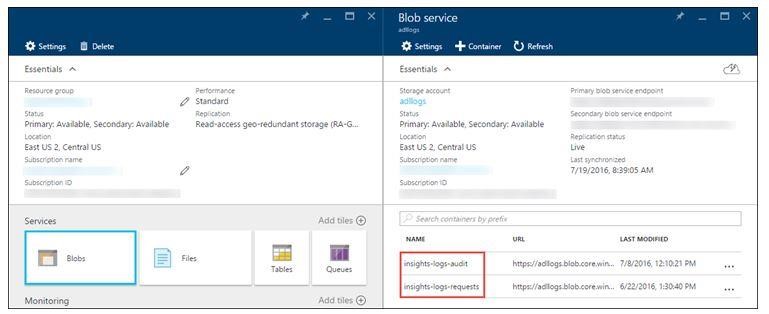
To download the logs, click the **Download** link against each log entry.

### From the Azure Storage account that contains log data

Open the Azure Storage account blade associated with Data Lake Store for logging, and then click Blobs. The **Blob service** blade lists two containers.

The container **insights-logs-audit** contains the audit logs.

The container **insights-logs-requests** contains the request logs.



Within these containers, the logs are stored under the following structure.



## Understand the structure of the log data

The audit and request logs are in a JSON format. In this section, we look at the structure of JSON for request and audit logs.

### 10.3.1 Request logs

Here's a sample entry in the JSON-formatted request log. Each blob has one root object called *records* that contains an array of log objects.

|  |
| --- |
| {  "records":  [ . . . .  ,  {  "time": "2016-07-07T21:02:53.456Z", "resourceId":  "/SUBSCRIPTIONS/<subscription\_id>/RESOURCEGROUPS/<resource\_group\_name>/PROVIDERS/MICROSOFT.DATALAKEST ORE/ACCOUNTS/<data\_lake\_store\_account\_name>",  "category": "Requests",  "operationName": "GETCustomerIngressEgress",  "resultType": "200",  "callerIpAddress": "::ffff:1.1.1.1",  "correlationId": "4a11c709-05f5-417c-a98d-6e81b3e29c58",  "identity": "1808bd5f-62af-45f4-89d8-03c5e81bac30", "properties":  {"HttpMethod":"GET","Path":"/webhdfs/v1/Samples/Outputs/Drivers.csv","RequestContentLength":0,"Client  RequestId":"3b7adbd907T21:02:52.472Z","EndTime"-3519-4:"2016f28-a61c-07-bd89506163b8","StartTime":"2016-07T21:02:53.456Z"} -07-  }  ,. . . .  ]  } |

Request log schema

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| Time | String | The timestamp (in UTC) of the log |
| resourceId | String | The ID of the resource that operation took place on |
| category | String | The log category. For example, **Requests**. |
| operationName | String | Name of the operation that is logged. For example, getfilestatus. |
| resultType | String | The status of the operation, For example, 200. |
| callerIpAddress | String | The IP address of the client making the request |
| correlationId | String | The ID of the log that can used to group together a set of related log entries |
| identity | Object | The identity that generated the log |
| properties | JSON | See below for details |

Request log schema properties

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| HttpMethod | String | The HTTP Method used for the operation. For example, GET. |
| Path | String | The path the operation was performed on |
| RequestContentLength | int | The content length of the HTTP request |
| ClientRequestId | String | The ID that uniquely identifies this request |
| StartTime | String | The time at which the server received the request |
| EndTime | String | The time at which the server sent a response |

### 10.3.2 Audit Logs

Here's a sample entry in the JSON-formatted audit log. Each blob has one root object called *records* that contains an array of log objects

|  |
| --- |
| {  "records":  [ . . . .  ,  {  "time": "2016-07-08T19:08:59.359Z",  "resourceId":  "/SUBSCRIPTIONS/<subscription\_id>/RESOURCEGROUPS/<resource\_group\_name>/PROVIDERS/MICROSOFT.DATALAKES  TORE/ACCOUNTS/<data\_lake\_store\_account\_name>",  "category": "Audit",  "operationName": "SeOpenStream",  "resultType": "0",  "correlationId": "381110fc03534e1cb99ec52376ceebdf;Append\_BrEKAmg;25.66.9.145",  "identity": "A9DAFFAF-FFEE-4BB5-A4A0-1B6CBBF24355",  "properties":  {"StreamName":"adl://<data\_lake\_store\_account\_name>.azuredatalakestore.net/logs.csv"}  }  ,  . . . .  ]  } |

Audit log schema

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| Time | String | The timestamp (in UTC) of the log |
| resourceId | String | The ID of the resource that operation took place on |
| Category | String | The log category. For example, **Audit**. |
| operationName | String | Name of the operation that is logged. For example, getfilestatus. |
| resultType | String | The status of the operation, For example, 200. |
| correlationId | String | The ID of the log that can used to group together a set of related log entries |
| Identity | Object | The identity that generated the log |
| properties | JSON | See below for details |

Audit log properties schema

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| StreamName | String | The path the operation was performed on |

# Example of an Azure Data Lake setup

This example shows how an Azure Data Lake can be established using the Azure Portal.

## Create the Data Lake

### 11.1.1 Create Directories

### 11.1.2 Assign access rights

## Create the Data Factory service

## Use PowerBI to see data

We can use PowerBI to directly open and use files with the Azure Data Lake.

# References

The following are references to the Microsoft Azure services mentioned in this document.

* Azure Data Lake: <https://docs.microsoft.com/en-us/azure/data-lake-store/>
* Azure Data Lake Analytics: <https://docs.microsoft.com/en-us/azure/data-lake-analytics/>
* Azure Data Factory:<https://docs.microsoft.com/en-us/azure/data-factory/>
* Azure Data Catalog:<https://azure.microsoft.com/en-us/services/data-catalog/>
* SQL Databases – in Azure- o SQL Server DB: <https://docs.microsoft.com/en-us/azure/sql-database/>o SQL Server DW: <https://docs.microsoft.com/en-us/azure/sql-data-warehouse/>o MySQL: <https://docs.microsoft.com/en-us/azure/mysql/>o PostgresSQL: <https://docs.microsoft.com/en-us/azure/postgresql/>
* Cubes – in Azure:<https://docs.microsoft.com/en-us/azure/analysis-services/>
* End-user tools- o PowerBI: [https://powerbi.microsoft.com/en-us/documentation/powerbi-landingpage/](https://powerbi.microsoft.com/en-us/documentation/powerbi-landing-page/) o HDInsight – R/Python:<https://docs.microsoft.com/en-us/azure/hdinsight/>