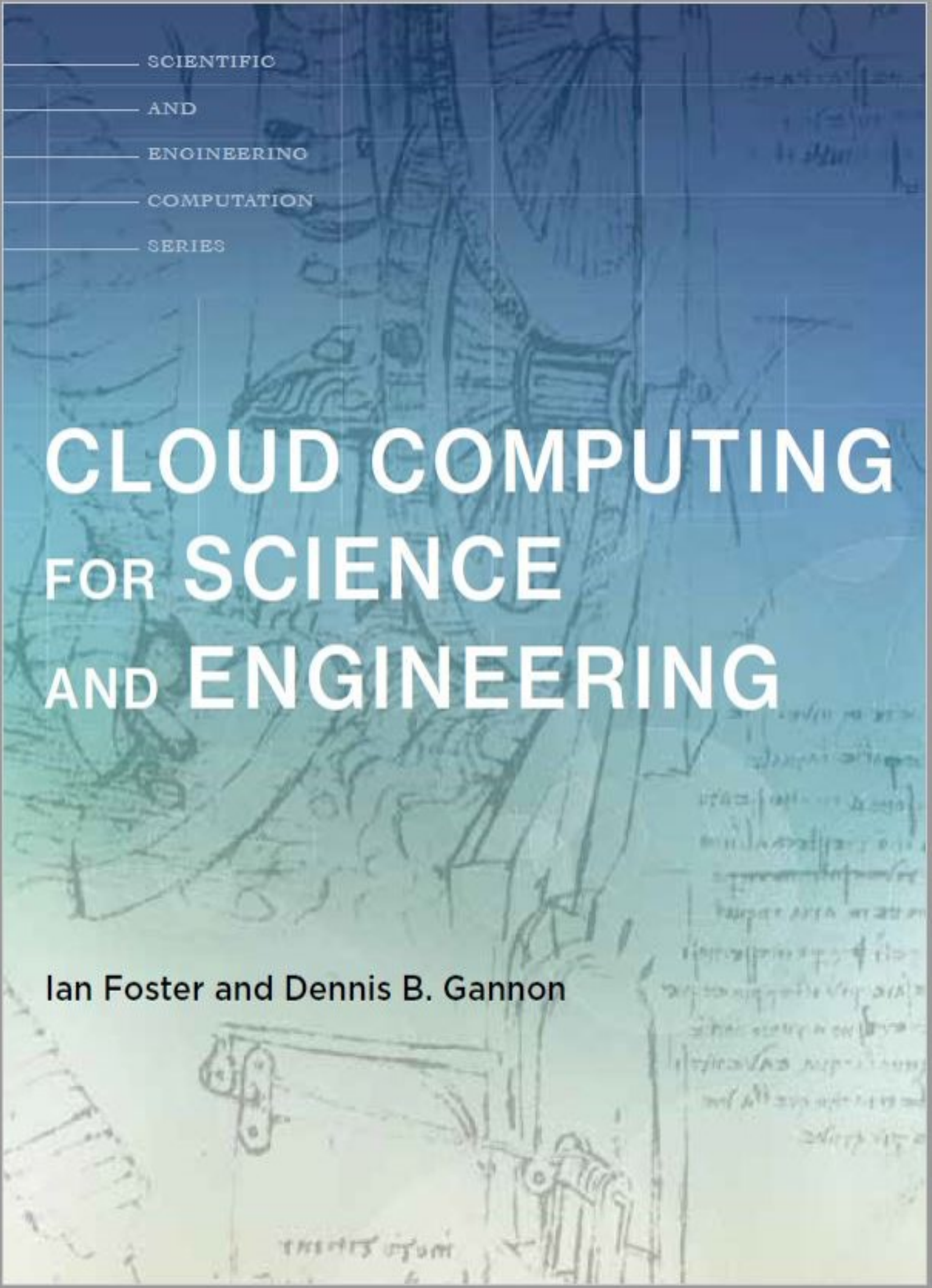


SCIENTIFIC AND ENGINEERING COMPUTATION SERIES

CLOUD COMPUTING FOR SCIENCE AND ENGINEERING

Ian Foster and Dennis B. Gannon

Storage as a Service

"As a general rule the most successful man in life is the man who has the best information."
—Benjamin Disraeli

Storage Requirements

- Many cloud services that we use routinely—services such as Box, Dropbox, OneDrive, Google Docs, YouTube, Facebook, and Netflix—are, above all, data services.
- Each runs in the cloud, hosts digital content in the cloud, and provides specialized methods for accessing, storing, and sharing that content.
- Each is, built on one or more—often multiple—storage services that have been variously optimized for speed, scale, reliability, and/or consistency.
- It is important that **how the basic infrastructure** components on which these applications are built **can be applied** to science and engineering problems.
- To address this question, we need to evaluate their relative merits.

Three Motivating Examples

Use Case 1 (UC1)

- A climate science laboratory has assembled a **set of simulation output files**.
- Each in Network Common Data Form **(NetCDF) format**: some **20 TB** in total.
- These data are to be made accessible via **interactive tools** running in a web portal.
- The data sizes are such that **data need to be partitioned** to enable distributed analyses over multiple machines running in parallel.

Three Motivating Examples

Use Case 2 (UC2)

- A seismic observatory is acquiring records describing **experimental observations**
- A **structured** data set with each observation specifying the time of the observation, experimental parameters, and the measurement itself, in **CSV format**.
- There may be a total of **1,000,000 records** totalling some **100 TB** when they finish collecting.
- They need to store these data to **enable easy access** by a large team, and to permit tracking of the data inventory and its accesses.

Three Motivating Examples

Use Case 3 (UC3)

- A team of scientists operates a **collection of** several thousand instruments, each of which generates a data record every few seconds.
- The **individual records are not large**, but managing the aggregate stream of all outputs in order to perform **analyses** across the entire collection every few hours introduces data management challenges.
- This problem is similar to that of **analysing large web traffic** or social media streams.
- **Unstructured data** due to heterogeneous capturing

Use Cases

UC1 : NetCDF Simulation data

Partitioning for parallel access

Uc2 : Large Observatory data

Easy Access, Relation between results, Structured

Uc3 : Web Traffic

Small data but Large traffic and non correlated, Unstructured

Storage Models

- An exciting feature of cloud storage systems is that they support a wide range of different storage models
  - File systems,
  - Object stores,
  - Relational databases,
  - Table stores,
  - NoSQL databases,
  - Graph databases,
  - Data warehouses,
  - and Archival storage
- The right storage model for a data collection can depend on not only the nature and size of the data, but also the analyses to be performed, sharing plans, and update frequencies, among other factors.

File System

- File system, organized around a tree of directories or folders

- The standard API for the Unix-derived version of the file system is called the Portable Operating System Interface (POSIX)

- Using command line tools, graphical user interfaces, or APIs, we create, read, write, and delete files located within directories

File System

Advantages

- It allows for the direct use of many existing programs without modification:
  - we can navigate a file system with familiar file system browsers, run programs written in our favourite analysis tool (Python, R, S tata, SPSS, Mathematica, etc.) on files, and share files via email.
- The file system model also provides a straightforward mechanism for representing hierarchical relationships among data—directories and files—and supports concurrent access by multiple readers.

➢In the early 1990s, the POSIX model was extended to distributed network file systems and there were some attempts at wide area versions.

➢By the late 1990s, the Linux Cluster community created Lustre, a true parallel file system that supports the POSIX standard

File System

Disadvantages

- Particularly as data volumes grow
- From a data modelling perspective, it provides no support for enforcing conventions concerning the representation of data elements and their relationships.
  - Thus, while one may, for example, choose to store environmental data in NetCDF files, genomes in FASTA files, and experimental observations in comma-separated-value (CSV) files, the file system does nothing to prevent the use of inconsistent representations within those files.
- The rigid hierarchical organization enforced by a file system often does not match the relationships that one wants to capture in science.
- Lacking any information on the data model, file systems cannot help users navigate complex data collections .
- The file system model also has problems from a scalability perspective: the need to maintain consistency as multiple processes read and write a file system can lead to bottlenecks in file system implementations

Object Stores

- The object storage model, like the file system model, stores unstructured binary objects.
- In the database world, objects are often referred to as blobs , for binary large objects, and we use that name here when it is consistent with terminology adopted by cloud vendors.
- An object/blob store simplifies the file system model in important ways:
  - in particular, it eliminates hierarchy and
  - forbids updates to objects once created.
- Different object storage services differ in their details, but in general they support a two-level folder-file hierarchy that allows for the creation of object containers , each of which can hold zero or more objects
- Each object is identified by a unique identifier and can have various metadata associated with it.
- Objects cannot be modified once uploaded: they can only be deleted—or, in object stores that support versioning, replaced

Object Storage

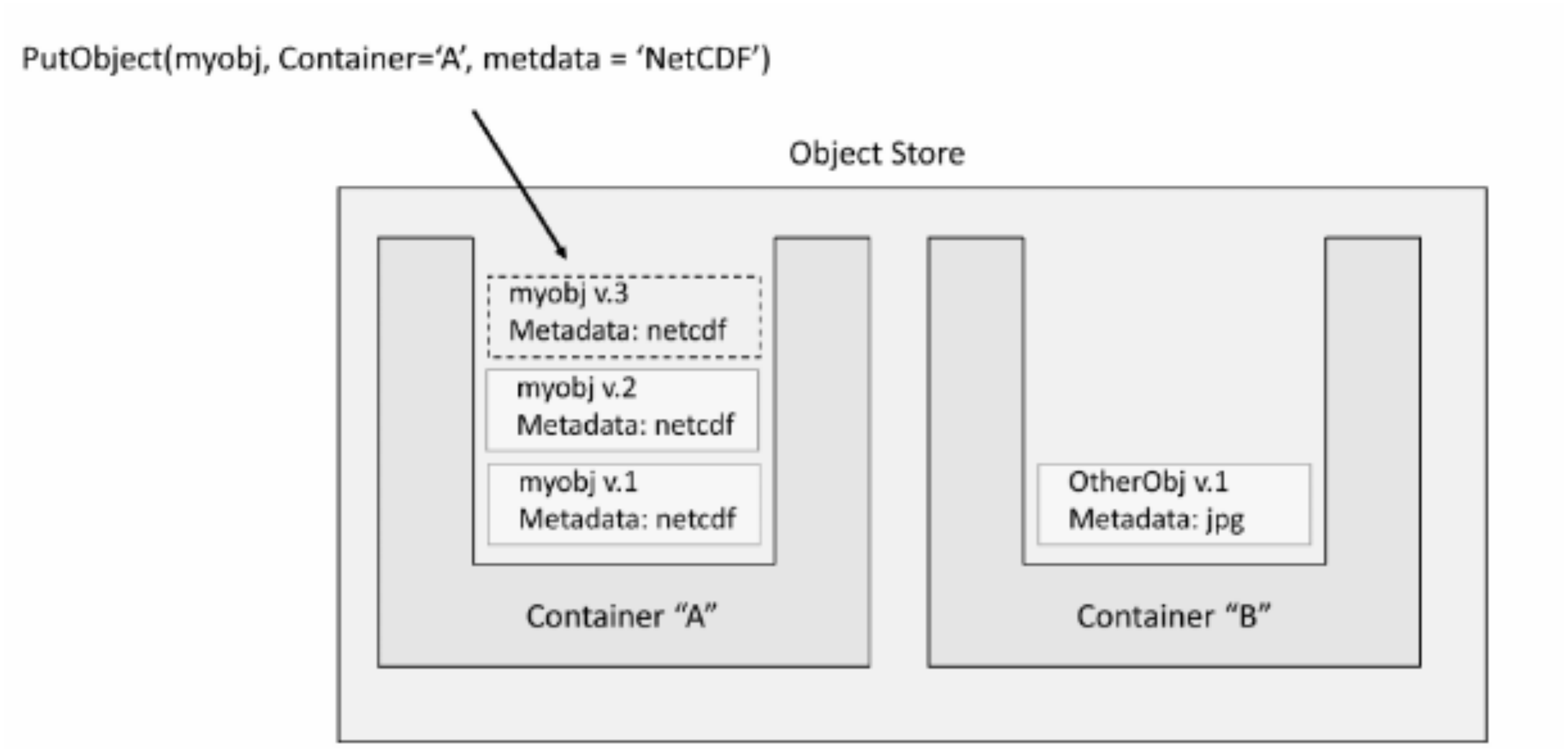


Figure 2.1: Object storage model with versioning. Each NetCDF file is stored in a separate container, and all versions of the same NetCDF file are stored in the same container.

Object Stores

Advantages

- The object store model has important advantages in terms of simplicity, performance, and reliability.
- The fact that objects cannot be modified once created makes it easy to build highly scalable and reliable implementations.
  - For example, each object can be replicated across multiple physical storage devices to increase resilience and (when there are many concurrent readers) performance, without any specialized synchronization logic in the implementation to deal with concurrent updates.
- Objects can be moved manually or automatically among storage classes with different performance and cost parameters.

Object Stores

Disadvantages

- It provides little support for organizing data and no support for search:
  - A user must know an object's identifier in order to access it.
- Thus, an object store would likely be inadequate as a basis for organizing the 1,000,000 environmental records of UC2 as
  - We would need to create a separate index to map from file characteristics to object identifiers.
- Nor does an object store provide any mechanism for working with structured data.
  - Thus, for example, while we could load each UC2 dataset into a separate object, a user would likely have to download the entire object to a computer to compute on its contents.
- Finally, an object store cannot easily be mounted as a file system or accessed with existing tools in the ways that a file system can.

Object Stores

- We can use this storage model to store the NetCDF data UC1 in use case UC1.
- As shown in Figure, we create a single container and store each NetCDF file in that container as an object, with the NetCDF filename as the object identifier.
- Any authorized individual who possesses that object identifier can then access the data via simple HTTP requests or API calls

Relational Database

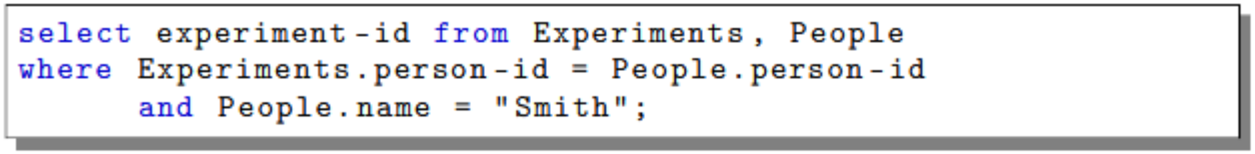
- A database is a structured collection of data about entities and their relationships.
- It models real-world objects—both entities (e.g., microscopes, experiments, and samples, as in UC2) and relationships (e.g., "sample1" was tested on "July 24")—and captures structure in ways that allow these entities and relationships to be queried for analysis.
- A database management system (DBMS) is a software suite designed to safely store and efficiently manage databases and to assist with the maintenance and discovery of the relationships that databases represent.
- In general, a DBMS encompasses three components:
  - Its data model (which defines how data are represented),
  - Its query language (which defines how the user interacts with the data), and support for
  - Transactions and crash recovery (to ensure reliable execution despite system failures)
- Useful for UC2

Relational Database

- For a variety of reasons, science and engineering data often belong in a database rather than in files or objects.
- The use of a DBMS simplifies data management and manipulation and provides for efficient querying and analysis, durable and reliable storage, scaling to large data sizes, validation of data formats, and management of concurrent accesses e.g. UC2.
- While DBMSs in general, and cloud-based DBMSs in particular, support a wide variety of data formats, two major classes can be distinguished:
  - Relational and
  - NoSQL

Relational Database (RDBMS)

- Relational DBMSs allow for the efficient storage, organization, and analysis of large quantities of tabular data: data organized as tables, in which rows represent entities (e.g., experiments) and columns represent attributes of those entities (e.g., experimenter, sample, result).
- The associated Structured Query Language (SQL) can then be used to specify a wide range of operations on such tables
  - For example, the following SQL joins two tables, Experiments and People , to find all experiments performed by Smith:



```
select experiment-id from Experiments, People
where Experiments.person-id = People.person-id
      and People.name = "Smith";
```

  - SQL statements can be executed with high efficiency thanks to sophisticated indexing and query planning techniques.
  - Thus this join can be executed quickly even if there are millions of records in the tables being joined.
- Many open source, commercial, and cloud-hosted relational DBMSs exist. Among the open source DBMSs
  - MySQL and
  - PostgreSQL (often simply Postgres) are particularly widely used.
- Both MySQL and Postgres are available in Storage as a Service in cloud-hosted forms.

Relational Database (RDBMS)

Relational databases have two important properties:

- First, they support a relational algebra that provides a clear, mathematical meaning to the SQL language, facilitating efficient and correct implementations.
- Second, they support ACID semantics, a term that captures four important database properties:
  - Atomicity (the entire transaction succeeds or fails),
  - Consistency (the data collection is never left in an invalid or conflicting state),
  - Isolation (concurrent transactions cannot interfere with each other), and
  - Durability (once a transaction completes, system failures cannot invalidate the result)

Relational Database (NoSQL)

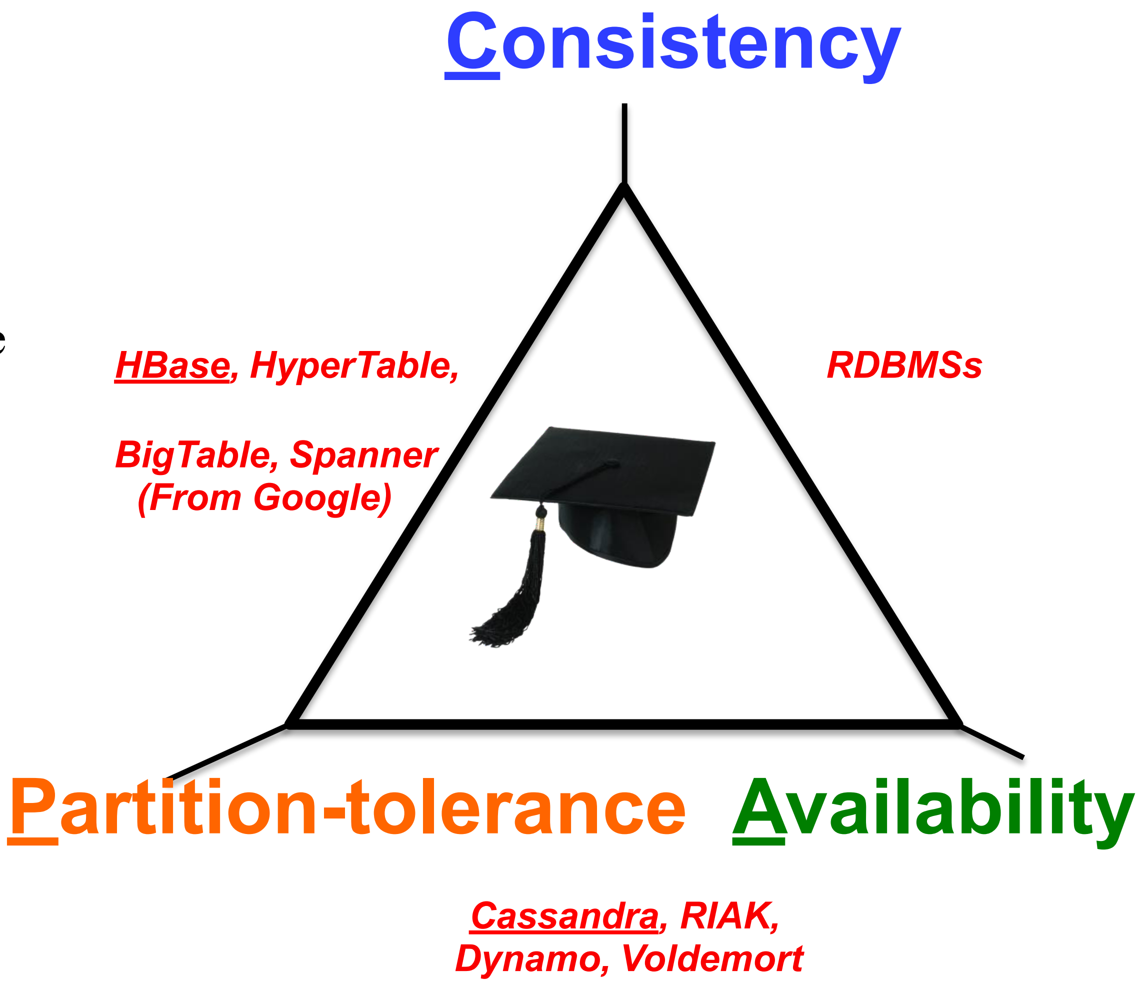
- A relational DBMS is almost certainly the right technology to use for highly structured datasets of moderate size.
- But if your data are less regular (if, for example, you are dealing with large amounts of text or if different items have different properties) or extremely large, you may want to consider a NoSQL DBMS .
- The design of these systems has typically been motivated by a desire to
  - scale the quantities of data and number of users that can be supported, and
  - to deal with unstructured data that are not easily represented in tabular form.
- Another definition of NoSQL is "not only SQL," meaning that most of SQL is supported but other properties are available.
  - For example, a NoSQL database may allow for the rapid ingest of large quantities of unstructured data, such as the instrument events of UC3
- Arbitrary data can be stored without modifications to a database schema, and new columns introduced over time as data and/or understanding evolves

Relational Database (NoSQL)

- NoSQL databases in the cloud are often distributed over multiple servers and also replicated over different data centres.
- Hence, they often fail to satisfy all of the ACID properties. Consistency is often replaced by eventual consistency, meaning that database state may be momentarily inconsistent across replicas.
- This relaxation of ACID properties is acceptable if your concern is to respond rapidly to queries about the current state of a store's inventory.
  - It may be unacceptable if the data in question are, for example, medical records.
- CAP Theorem: Not possible to have all three properties viz.
  - Consistency indicates that all computers see the same data at the same time.
  - Availability indicates that every request receives a response about whether it succeeded or failed.
  - Partition tolerance indicates that the sys tem continues to operate even if a network failure prevents computers from communicating

CAP Theorem

- Starting point for NoSQL Revolution
- A distributed storage system can achieve at most two of C, A, and P.
- When partition- tolerance is important, you have to choose between consistency and availability



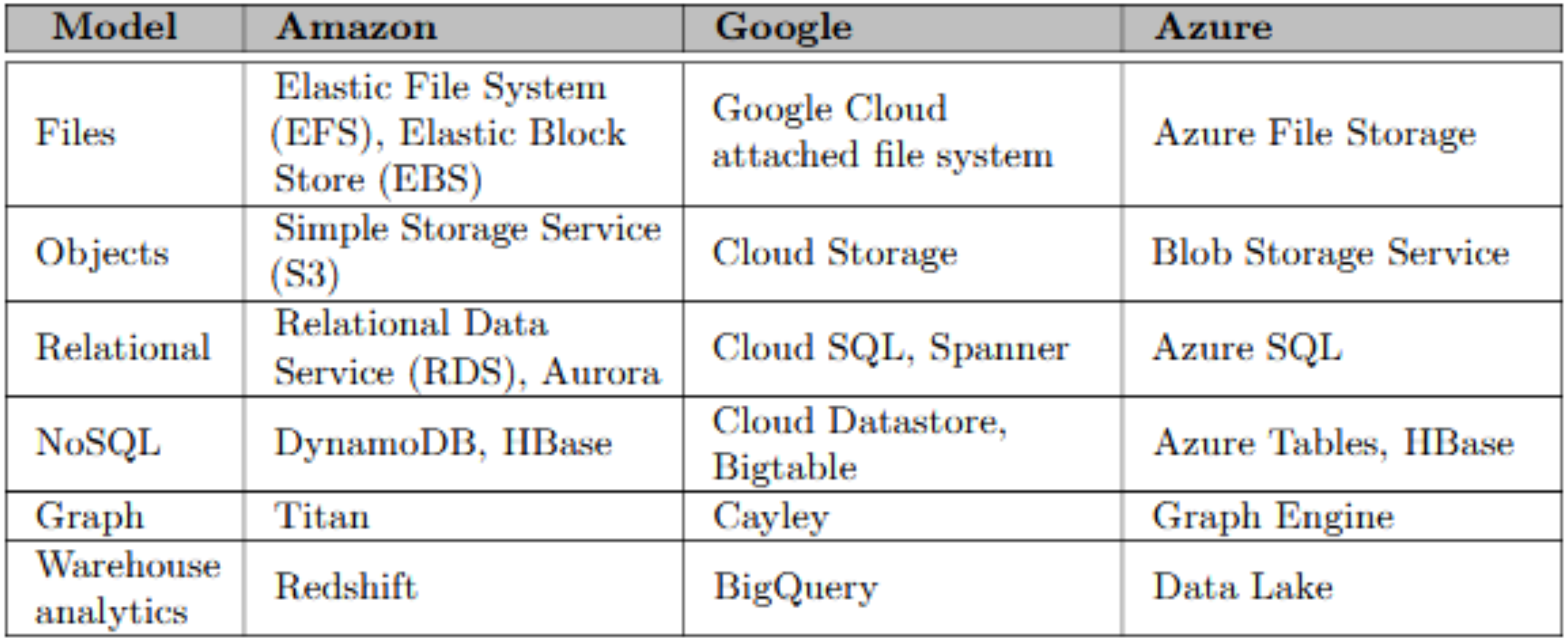
Graph Databases

- A graph is a data structure in which edges connect nodes
- Graphs are useful when we need to search data based on relationships among data items. For example,
  - In UC2, measurements from different experiments might be related by their use of the same measurement modality;
  - In a database of scientific publications, publications can be represented as nodes and citations, shared authors, or even shared concepts as edges.
- Often graph databases are built on top of existing NoSQL databases.

Data Warehouses

- The term data warehouse is commonly used to refer to data management systems optimized to support analytic queries that involve reading large datasets.
- Data warehouses have different design goals and properties than do DBMSs. For example,
  - A medical centre's clinical DBMS is typically designed to enable many concurrent requests to read and update information on individual patients (e.g., "what is Ms. Smith' s current weight?" or "Mr. Jones was prescribed Aspirin").
- Data from this DBMS are uploaded periodically (e.g., once a day) into the medical centre's data warehouse to support aggregate queries such as
  - "What factors are correlated with length of stay of patients in hospital?"
- Several cloud vendors offer data warehouse solutions that can scale to extremely large data volumes

The Cloud Storage Landscape



Model	Amazon	Google	Azure
Files	Elastic File System (EFS), Elastic Block Store (EBS)	Google Cloud attached file system	Azure File Storage
Objects	Simple Storage Service (S3)	Cloud Storage	Blob Storage Service
Relational	Relational Data Service (RDS), Aurora	Cloud SQL, Spanner	Azure SQL
NoSQL	DynamoDB, HBase	Cloud Datastore, Bigtable	Azure Tables, HBase
Graph	Titan	Cayley	Graph Engine
Warehouse analytics	Redshift	BigQuery	Data Lake

Summary

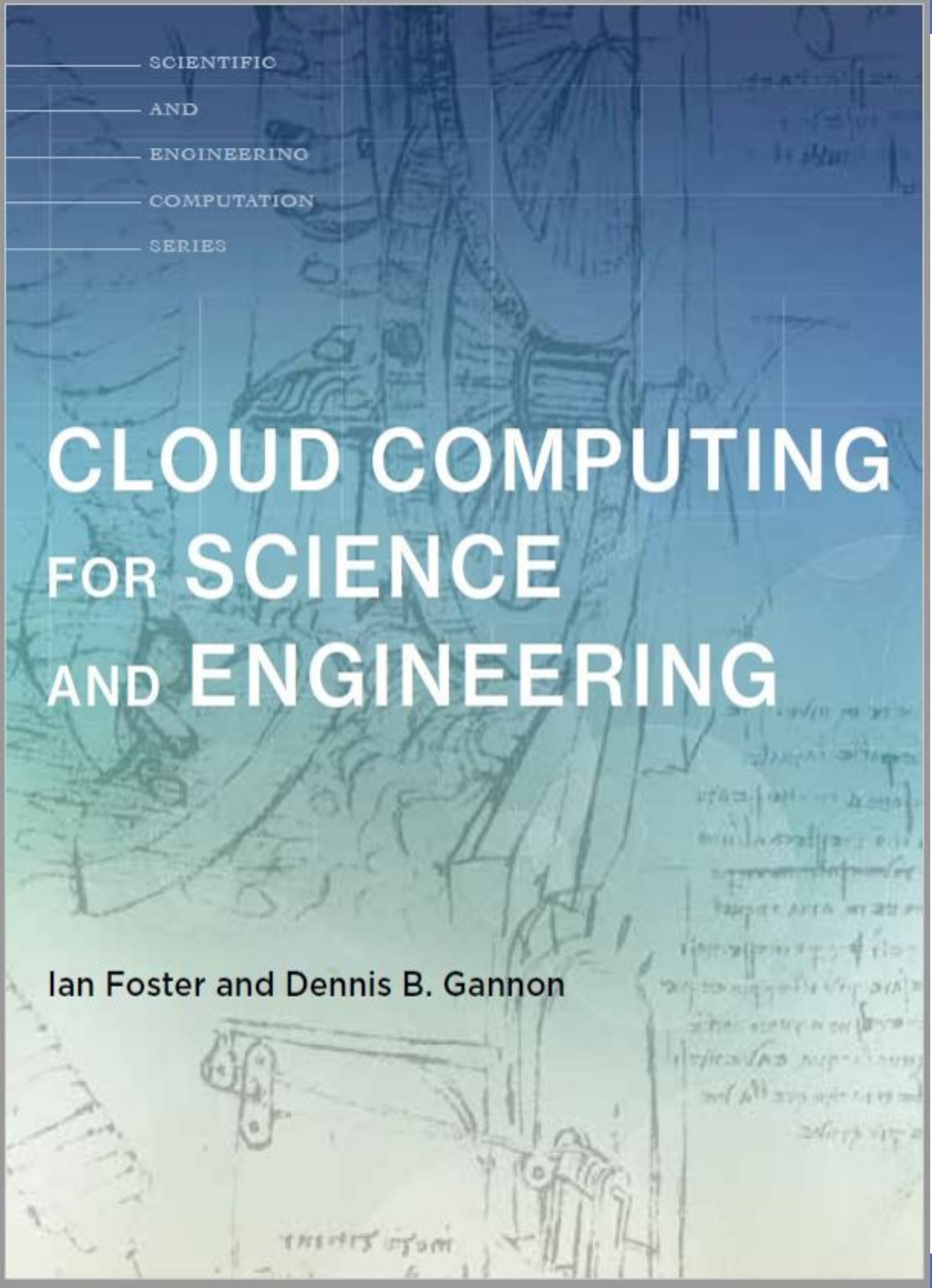
- The first use case UC1, involving climate simulation output files in NetCDF format, is clearly a case for
  - Amazon S3,
  - Google Cloud Storage, or
  - Azure Blob Storage.
- Each blob can be up to 1 TB in size (5 TB for S3).

Summary

- The second use case UC2, involving 1,000,000 records describing experimental observations, could also be handled with simple blob storage, but the cloud presents us with better solutions.
- The simplest is to use a standard relational SQL database, but the merits of this approach depend on how strict we are with the schema that describes the data. Do all records have the same structure, or do some have fields that others do not?
- In the latter case, a NoSQL database may be a superior solution.
- Other factors are scale and the possible need for parallel access.
- Cloud NoSQL stores like Azure Tables, Amazon DynamoDB, and Google Bigtable, are massively scalable and replicated .
- Unlike conventional SQL database solutions, they a re designed for highly parallel massive data analysis

Summary

- The third use case UC3, involving a massive set of instrument event records, is also appropriate for cloud NoSQL databases.
- However, data warehouses such as Amazon Redshift and Azure Da ta Lake are designed to be complete platforms for performing data analytics on massive data collections.
- If our instrument records are streaming in real time, we can use event streaming tools based on publish/subscribe semantics


SCIENTIFIC
AND
ENGINEERING
COMPUTATION
SERIES
CLOUD COMPUTING FOR SCIENCE AND ENGINEERING
Ian Foster and Dennis B. Gannon

Using Cloud Storage Services

"Collecting data is only the first step toward wisdom, but sharing data is the first step toward community."
—Henry
Louis Gates Jr

Cloud Storage Services

- While the services of different cloud providers are often similar in outline, they invariably differ in the details
- Next few slides is an effort to illustrate the use of the services used in three major public clouds
  - Amazon,
  - Azure,
  - Google

Two Access Methods: Portals and APIs

- Each cloud provider's web portal typically allows you to accomplish anything that you want to do with a few mouse clicks.

- While such portals are good for performing simple actions, they are not ideal for the repetitive tasks, such as managing hundreds of data objects, that scientists need to do on a daily basis.

- For such tasks, we need an interface to the cloud that we can program.

- Cloud providers make this possible by providing REST APIs that programmers can use to access their services programmatically.

- For programming convenience, you will usually access these APIs via software development kits (SDKs), which give programmers language-specific functions for interacting with cloud services

Two Access Methods: Portals and APIs

- Each cloud has special features that make it unique, and thus the different cloud provider's REST APIs and SDKs are not identical.
- Two efforts are under way to create a standard Python SDK:
  - CloudBridge and
  - Apache Libcloud (*libcloud.apache.org*)
- While both aim to support the standard tasks for all clouds, those tasks are only the lowest common denominator of cloud capabilities;
- Therefore, any unique capabilities of each cloud are available only through the REST API and SDK for that platform.
- Libcloud is still in making and not complete

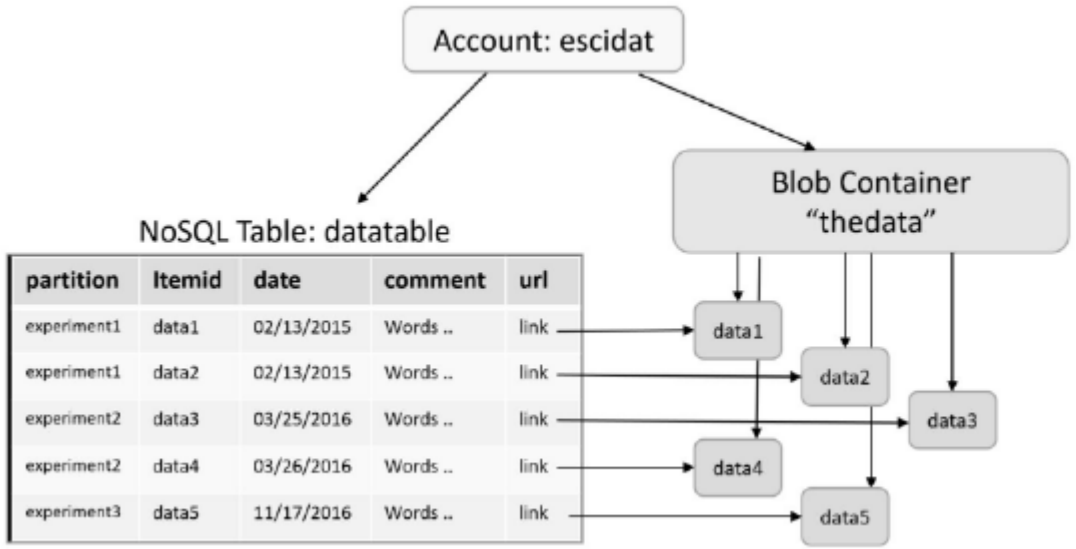
* Representational State Transfer (REST) is an application programming interface (API) that permits requests to be transmitted via the secure Hypertext Transfer Protocol (HTTPS) that is used by web browsers

Simple Example

- We have a collection of data samples stored on our personal computer and for each sample we have four items of metadata:

item number, creation date, experiment id, and a text string comment

- To enable access to these samples by collaborators, we want to upload them to cloud storage and to create a searchable NoSQL table, also hosted in the cloud, containing the metadata and cloud storage URL for each object



partition	Itemid	date	comment	url
experiment1	data1	02/13/2015	Words ..	link
experiment1	data2	02/13/2015	Words ..	link
experiment2	data3	03/25/2016	Words ..	link
experiment2	data4	03/26/2016	Words ..	link
experiment3	data5	11/17/2016	Words ..	link

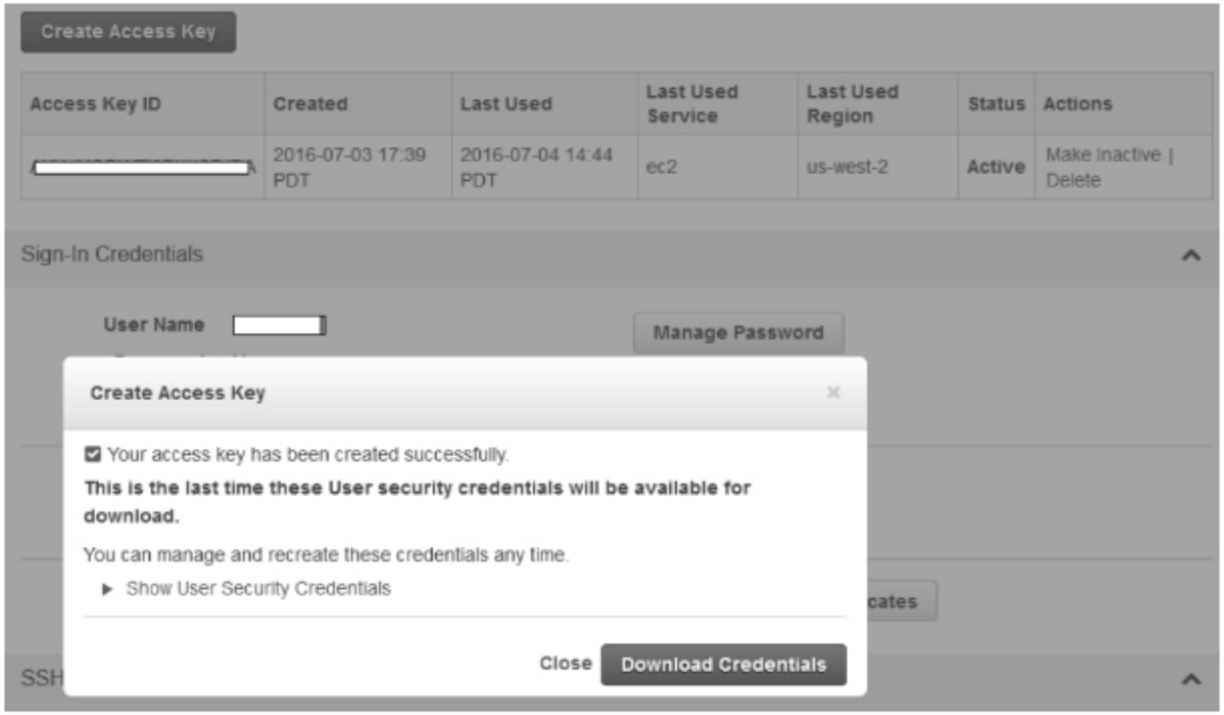
Simple Example

- We assume that each data sample is in a binary file on our personal computer and that the associated metadata are contained in a comma separated value (CSV) file, with one line per item, also on our personal computer.
- Each line in this CSV file has the following format:

item id, experiment id, date, filename, comment string

Using Amazon Cloud Storage Services

- Solution to the example problem uses S3 to store the blobs and DynamoDB to store the table.
- We first need our Amazon key pair, i.e., access key plus secret key, which we can obtain from the Amazon IAM Management Console
- Having created a new user, we select the create access key button to create our security credentials, which we can then download



Access Key ID	Created	Last Used	Last Used Service	Last Used Region	Status	Actions
[redacted]	2016-07-03 17:39 PDT	2016-07-04 14:44 PDT	ec2	us-west-2	Active	Make Inactive | Delete

Sign-In Credentials

User Name [redacted]

Create Access Key

Your access key has been created successfully.

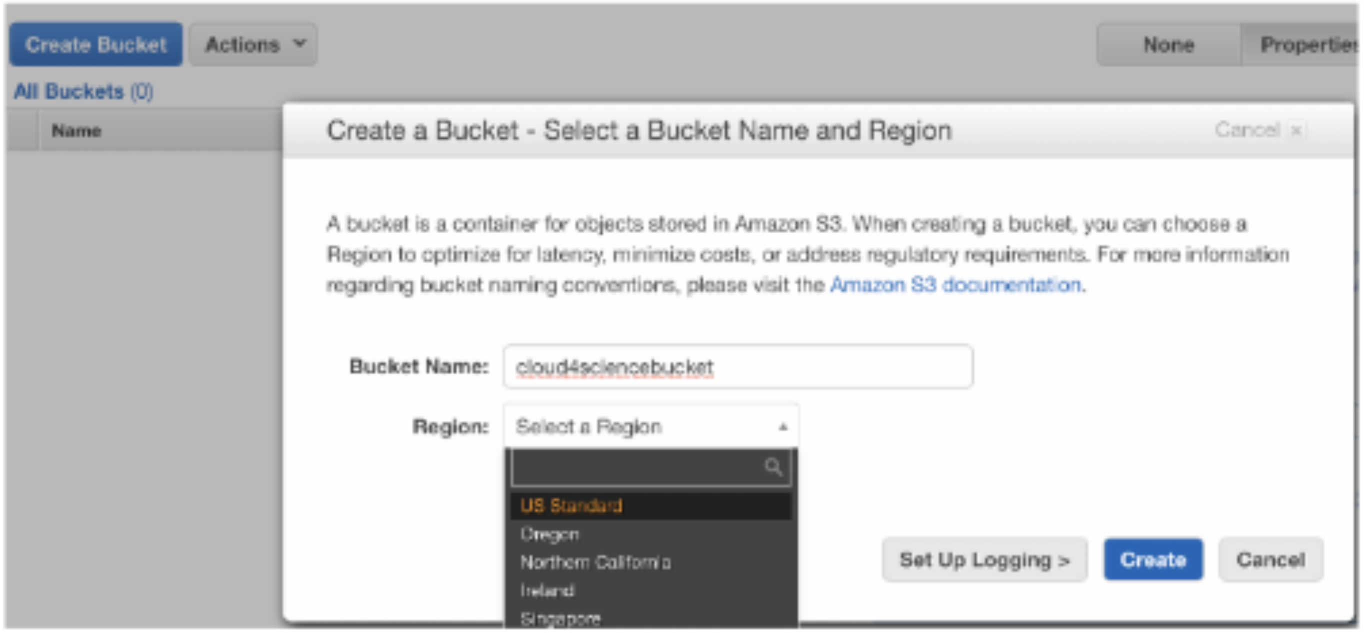
This is the last time these User security credentials will be available for download.

You can manage and recreate these credentials any time.

▸ Show User Security Credentials

Using Amazon Cloud Storage Services

- We can proceed to create the required S3 bucket, upload our blobs to that bucket, and so forth, all from the Amazon web portal.
- Although we can use Amazon portal to create a bucket.



Create a Bucket - Select a Bucket Name and Region

A bucket is a container for objects stored in Amazon S3. When creating a bucket, you can choose a Region to optimize for latency, minimize costs, or address regulatory requirements. For more information regarding bucket naming conventions, please visit the Amazon S3 documentation.

Bucket Name: cloud4sciencebucket

Region: Select a Region

https://console.aws.amazon.com/s3

- However, there are a lot of blobs, so we instead use the Amazon Python Boto3 SDK for these tasks
- Amazon operates i n 13 regions; region us-west-2 is located in Oregon

Boto3 and AWS S3

- Boto3 is the Amazon Web Services (AWS) SDK for Python.
- To use the S3 system, we need to create an S3 resource object.

```
import boto3
s3 = boto3.resource('s3',
    aws_access_key_id='YOUR ACCESS KEY',
    aws_secret_access_key='your secret key' )
```

- Having created the S3 resource object, we can now create the S3 bucket, datacont, in which we will store our data objects

```
import boto3
s3 = boto3.resource('s3')
s3.create_bucket(Bucket='datacont', CreateBucketConfiguration={
    'LocationConstraint': 'us-west-2'})
```

- Now that we have created the new bucket, we can load our data objects into it

```
# Upload a file, 'test.jpg' into the newly created bucket
s3.Object('datacont', 'test.jpg').put(
    Body=open('/home/mydata/test.jpg', 'rb'))
```

Boto3 and DynamoDB (NoSQL)

- We can even create the DynamoDB table in which we will store metadata and references to S3 objects.
- We create this table by defining a special key that is composed of a *PartitionKey* and a *RowKey*.
- NoSQL systems such as DynamoDB are distributed over multiple storage devices, which enable constructing extremely large tables that can then be accessed in parallel by many servers, each accessing one storage device.
- Hence the table's aggregate bandwidth is multiplied by the number of storage devices.
- DynamoDB distributes data via row: for any row, every element in that row is mapped to the same device.
- Thus, to determine the device on which a data value is located, you need only look up the *PartitionKey*, which is hashed to an index that determines the physical storage device in which the row resides.
- The *RowKey* specifies that items are stored in order, sorted by the *RowKey* value.
- While it is not necessary to have both keys, we also illustrate the use of RowKey here.

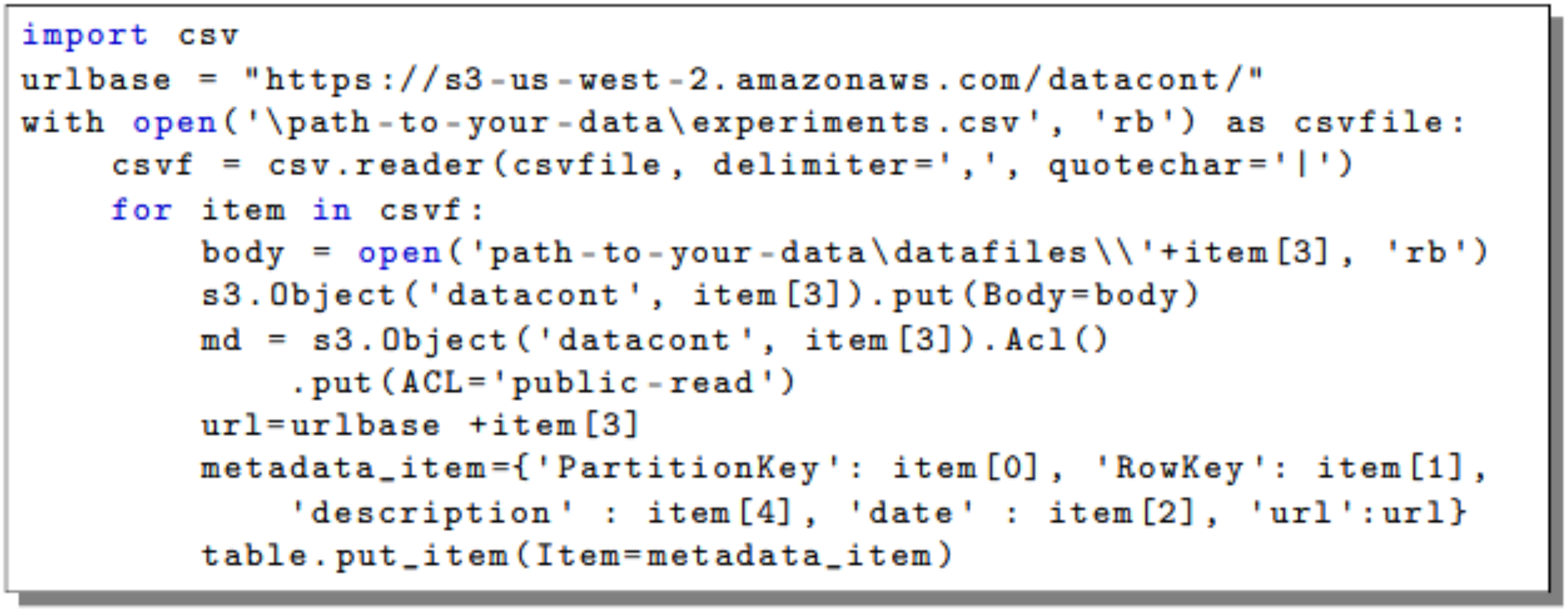
Boto3 and DynamoDB

Create the DynamoDB table

```
dyndb = boto3.resource('dynamodb', region_name='us-west-2' )

# The first time that we define a table, we use
table = dyndb.create_table(
    TableName='DataTable',
    KeySchema=[
        { 'AttributeName': 'PartitionKey', 'KeyType': 'HASH'},
        { 'AttributeName': 'RowKey', 'KeyType': 'RANGE' }
    ],
    AttributeDefinitions=[
        { 'AttributeName': 'PartitionKey', 'AttributeType': 'S' },
        { 'AttributeName': 'RowKey',       'AttributeType': 'S' }
    ]
)

# Wait for the table to be created
table.meta.client.get_waiter('table_exists')
                          .wait(TableName='DataTable')

# If the table has been previously defined, use:
# table = dyndb.Table("DataTable")
```

Boto3 and DynamoDB

- We are now ready to read the metadata from the CSV file, move the data objects into the blob store, and enter the metadata row into the table.
- This is done as follows. Recall that our CSV file format has item[3] as filename, item[0] as itemID, item[1] as experimentID, item[2] as date, and item[4] as comment



```
import csv
urlbase = "https://s3-us-west-2.amazonaws.com/datacont/"
with open('\path-to-your-data\experiments.csv', 'rb') as csvfile:
    csvf = csv.reader(csvfile, delimiter=',', quotechar='|')
    for item in csvf:
        body = open('path-to-your-data\datafiles\\'+item[3], 'rb')
        s3.Object('datacont', item[3]).put(Body=body)
        md = s3.Object('datacont', item[3]).Acl()
            .put(ACL='public-read')
        url=urlbase +item[3]
        metadata_item={'PartitionKey': item[0], 'RowKey': item[1],
            'description' : item[4], 'date' : item[2], 'url':url}
        table.put_item(Item=metadata_item)
```

Microsoft Azure Storage Services

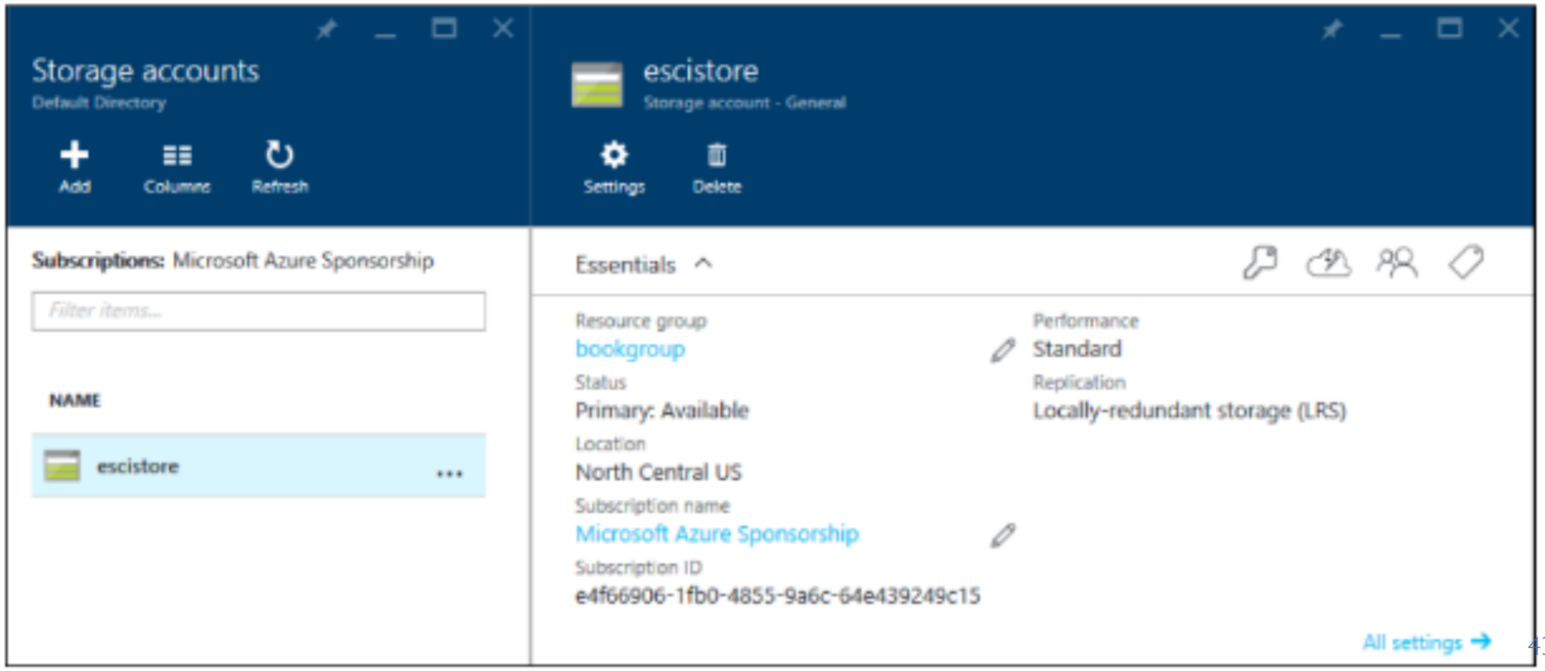
- Amazon account ID i s defined by a pair consisting of your access key and your secret key.
- Similarly, your Azure account is defined by your personal ID and a subscription ID.
- Your personal ID is probably your email address, so that is public; the subscription ID is something to keep secret
- The differences between Amazon DynamoDB and the Azure Table service are subtle.
- With the Azure Table service, each row has the fields PartitionKey , RowKey , comments , date ,and URL as before, but this time the RowKey is a unique integer for each row.
- The PartitionKey is used as a hash to locate the row into a specific storage device, and the RowKey is a unique global identifier for the row

Microsoft Azure Storage Services

- In S3, you create buckets and then create blobs within a bucket.
- S3 also provides an illusion of folders, although these are actually just blob name prefixes (e.g., folder1/).
- In contrast, Azure storage is based on Storage Accounts ,a higher level abstraction than buckets.
- You can create as many storage accounts as you want; each can contain five different types of objects: blobs, containers, file shares, tables, and queues.
- Blobs are stored in bucket-like containers that can also have a pseudo directory-like structure, similar to S3 buckets

Microsoft Azure Storage Services

- Given your user ID and subscription ID , you can use the Azure Python SDK to create storage accounts, much as we create buckets in S3.
- However, it easier to use the Azure portal.
- Login and click on storage account in the menu on the left to bring up a panel for storage accounts. To add a new account, click on the + sign at the top of the panel. You need to supply a name and some additional parameters such as location, duplication, and distribution .



Subscriptions: Microsoft Azure Sponsorship

escistore

Resource group
bookgroup

Status
Primary: Available

Location
North Central US

Subscription name
Microsoft Azure Sponsorship

Subscription ID
e4f66906-1fb0-4855-9a6c-64e439249c15

Performance
Standard

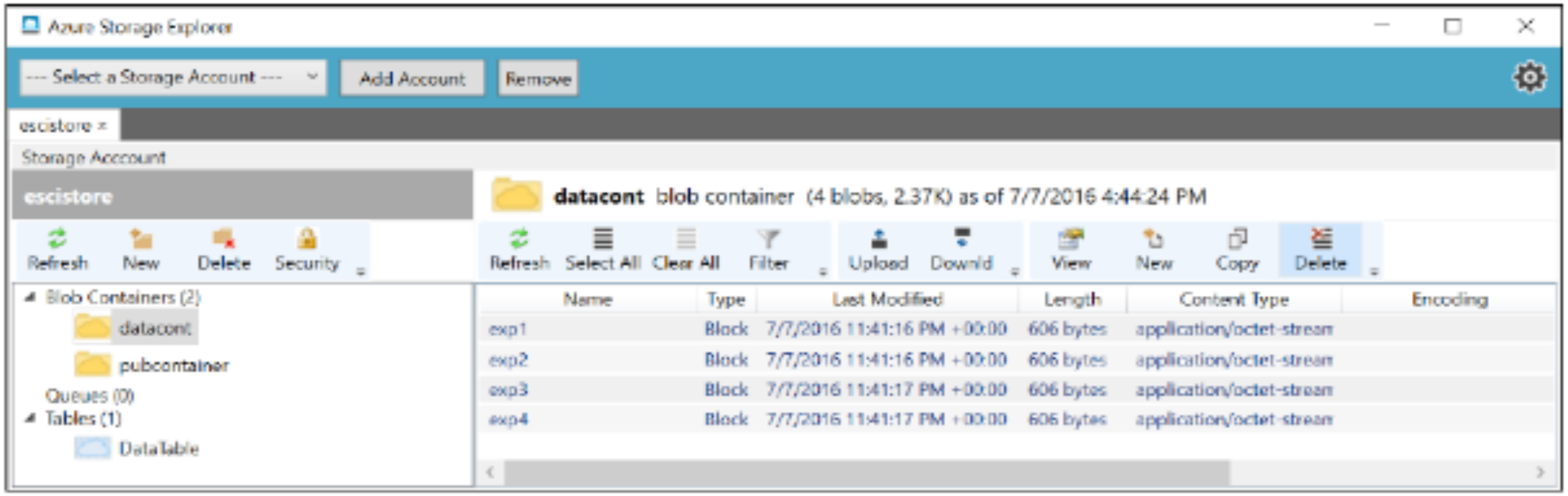
Replication
Locally-redundant storage (LRS)

Microsoft Azure Storage Services: SDK Approach

```
import azure.storage
from azure.storage.table import TableService, Entity
from azure.storage.blob import BlockBlobService
from azure.storage.blob import PublicAccess
# First, access the blob service
block_blob_service = BlockBlobService(account_name='escistore',
    account_key='your storage key')
block_blob_service.create_container('datacont',
    public_access=PublicAccess.Container)
# Next, create the table in the same storage account
table_service = TableService(account_name='escistore',
                             account_key='your account key')
if table_service.create_table('DataTable'):
    print("Table created")
else:
    print("Table already there")
```

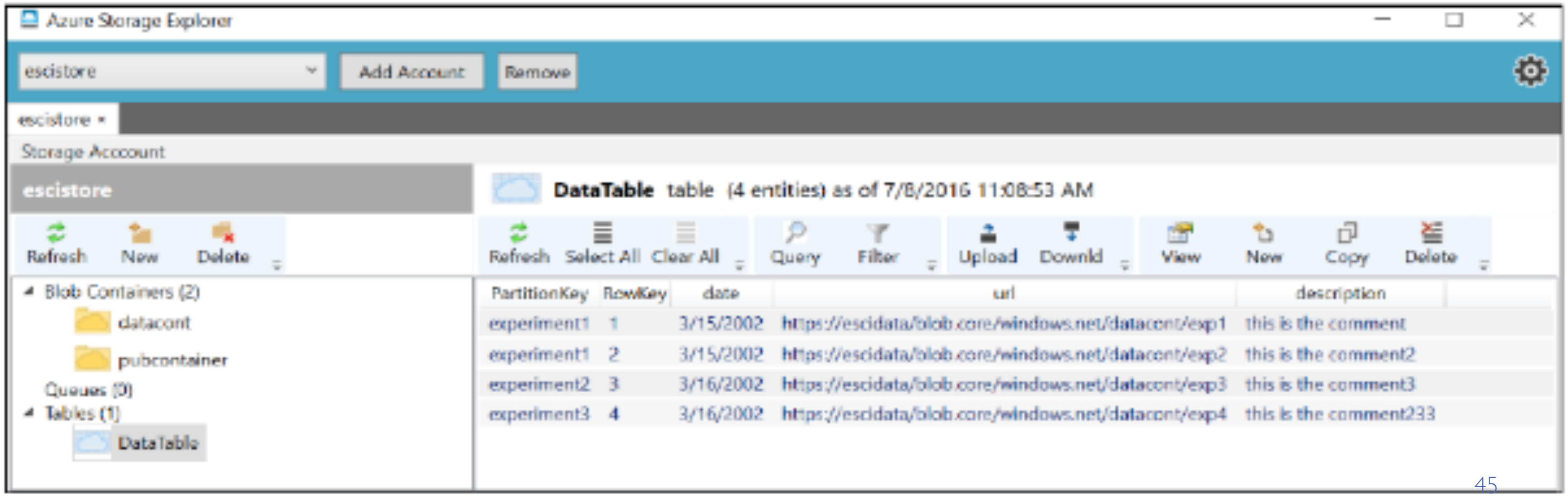
```
import csv
with open('\path-to-your-data\experiments.csv', 'rb') as csvfile:
   csvf = csv.reader(csvfile, delimiter=',', quotechar='|')
   for item in csvf:
     print(item)
     block_blob_service.create_blob_from_path(
            'datacont', item[3],
            "\path-to-your-files\datafiles\\"+item[3]
            )
     url="https://escistore.blob.core.windows.net/datacont/"+item[3]
     metadata_item = {'PartitionKey':item[0], 'RowKey':item[1],
          'description' : item[4], 'date' : item[2], 'url':url}
     table_service.insert_entity('DataTable', metadata_item)
```

Azure View



datacont blob container (4 blobs, 2.37K) as of 7/7/2016 4:44:24 PM

Name	Type	Last Modified	Length	Content Type	Encoding
exp1	Block	7/7/2016 11:41:16 PM +00:00	606 bytes	application/octet-stream	
exp2	Block	7/7/2016 11:41:16 PM +00:00	606 bytes	application/octet-stream	
exp3	Block	7/7/2016 11:41:17 PM +00:00	606 bytes	application/octet-stream	
exp4	Block	7/7/2016 11:41:17 PM +00:00	606 bytes	application/octet-stream	



DataTable table (4 entities) as of 7/8/2016 11:08:53 AM

PartitionKey	RowKey	date	url	description
experiment1	1	3/15/2002	https://escidata/blob.core/windows.net/datacont/exp1	this is the comment
experiment1	2	3/15/2002	https://escidata/blob.core/windows.net/datacont/exp2	this is the comment2
experiment2	3	3/16/2002	https://escidata/blob.core/windows.net/datacont/exp3	this is the comment3
experiment3	4	3/16/2002	https://escidata/blob.core/windows.net/datacont/exp4	this is the comment233

Resources

- Amazon Boto3 `aws.amazon.com/sdk-for-python/`
- Azure `azure.microsoft.com/en-us/develop/python/`
- Google's Cloud `cloud.google.com/sdk/`
- Openstack CloudBridge `cloudbridge.readthedocs.io/en/latest/`
- The Globus Python SDK `github.com/globus/globus-sdk-python` and the Globus CLI `github.com/globus/globus-cli`