Lecture 6

Cloud Computing I

(Concepts, Technology & Architecture)

Dr. Lydia Wahid

Agenda

- What is cloud computing?
- Cloud Delivery Models
- Cloud Deployment Models
- Cloud-Enabling Technology
- Cloud Architecture

What is cloud computing?

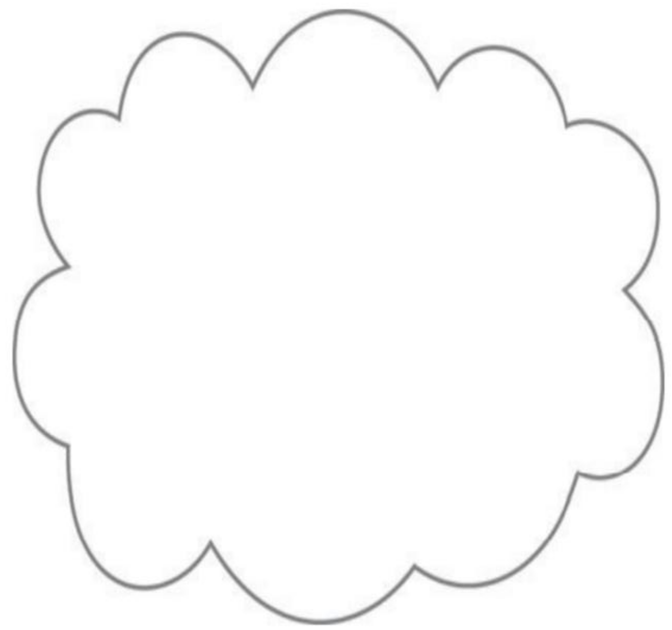
Definitions

Cloud Computing:

- "…a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service to external customers using Internet technologies." *Gartner*
- "Cloud computing is a specialized form of distributed computing that introduces utilization models for remotely provisioning scalable and measured resources." *National Institute of Standards and Technology (NIST)*

Definitions

The symbol used to denote the boundary of a cloud environment:



Basic Concepts and Terminology

Cloud:

- A cloud refers to a distinct IT environment that is designed for the purpose of remotely provisioning scalable and measured IT resources.

IT Resource:

- An IT resource is a physical or virtual IT-related artifact that can be either software-based, such as a virtual server or a custom software program, or hardware-based, such as a physical server or a network device.

Cloud Consumers and Cloud Providers:

- The party that provides cloud-based IT resources is the cloud provider. The party that uses cloud-based IT resources is the cloud consumer.

Basic Concepts and Terminology

- **On-Premise:**
  - An IT resource that is hosted in a conventional IT enterprise within an organizational boundary (that does not specifically represent a cloud) is considered to be located on the premises of the IT enterprise, or on-premise.
  - This term is used to qualify an IT resource as an alternative to "cloud-based." An IT resource that is on-premise cannot be cloud-based, and vice-versa.
  - An on-premise IT resource can access and interact with a cloud-based IT resource.
  - An on-premise IT resource can be moved to a cloud, thereby changing it to a cloud-based IT resource.

Cloud Vs. Internet

- As a specific environment used to remotely provision IT resources, a cloud has a finite boundary.
- There are many individual clouds that are accessible via the Internet.
- Whereas the Internet provides open access to many Web-based IT resources, a cloud is typically privately owned and offers access to IT resources that is metered.

Cloud Vs. Internet

- Much of the Internet is dedicated to the access of content-based IT resources published via the World Wide Web.
- IT resources provided by cloud environments, on the other hand, are dedicated to supplying back-end processing capabilities and user-based access to these capabilities.
- It is not necessary for clouds to be Web-based. A cloud can be based on the use of any protocols that allow for the remote access to its IT resources.

Benefits to cloud consumers

- On-demand access to pay-as-you-go computing resources on a short-term basis.
- The ability to release these computing resources when they are no longer needed.
- Having unlimited computing resources that are available on demand.
- The ability to add or remove IT resources at a fine-grained level.
- Abstraction of the infrastructure so applications are not locked into devices or locations and can be easily moved if needed.

Risks and Challenges

- Increased Security Vulnerabilities: responsibility over data security becomes shared with the cloud provider.
- Longer geographic distances between the cloud consumer and cloud provider can require additional network hops that introduce fluctuating latency and potential bandwidth constraints.
- Limited Portability Between Cloud Providers.

Cloud Delivery Models

Cloud Delivery Models

- A *cloud delivery model* represents a specific, pre-packaged combination of IT resources offered by a cloud provider.
- Three common cloud delivery models have become widely established and formalized:
  - Infrastructure-as-a-Service (IaaS)
  - Platform-as-a-Service (PaaS)
  - Software-as-a-Service (SaaS)

Cloud Delivery Models

- **Infrastructure-as-a-Service (IaaS):**
  - The IaaS delivery model represents a self-contained IT environment comprised of infrastructure-centric IT resources that can be accessed and managed via cloud service-based interfaces and tools.
  - This environment can include hardware, network, connectivity, operating systems, and so on.
  - The general purpose of an IaaS environment is to provide cloud consumers with a high level of control and responsibility over its configuration and utilization.
  - This model is therefore used by cloud consumers that require a high level of control over the cloud-based environment they intend to create.

Cloud Delivery Models

- **Infrastructure-as-a-Service (IaaS):**
  - A central and primary IT resource within a typical IaaS environment is the **virtual server**.
  - Virtual servers are leased by specifying server hardware requirements, such as processor capacity, memory, and local storage space.

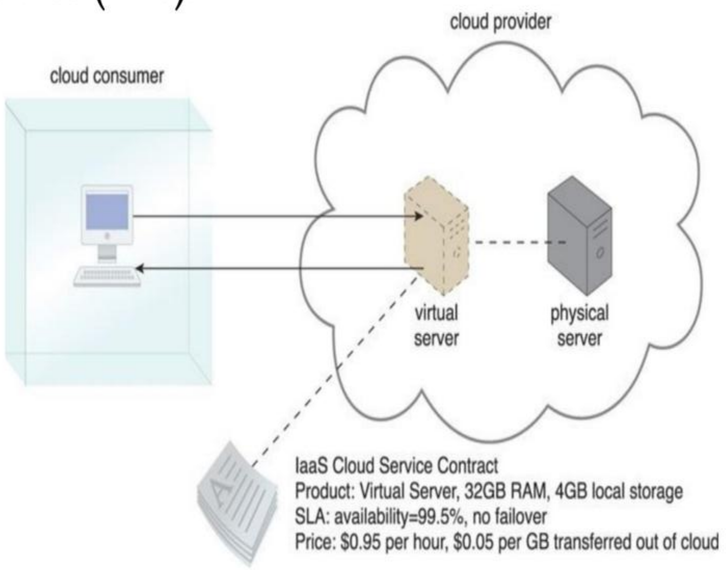


Figure 4.11. A cloud consumer is using a virtual server within an IaaS environment. Cloud consumers are provided with a range of contractual guarantees by the cloud provider, pertaining to characteristics such as capacity, performance, and availability.

Cloud Delivery Models

- **Platform-as-a-Service (PaaS):**
  - The PaaS delivery model represents a pre-defined "ready-to-use" environment typically comprised of already deployed and configured IT resources.
  - By working within a ready-made platform, the cloud consumer is spared the administrative burden of setting up and maintaining the bare infrastructure IT resources provided via the IaaS model.

Cloud Delivery Models

- **Platform-as-a-Service (PaaS):**

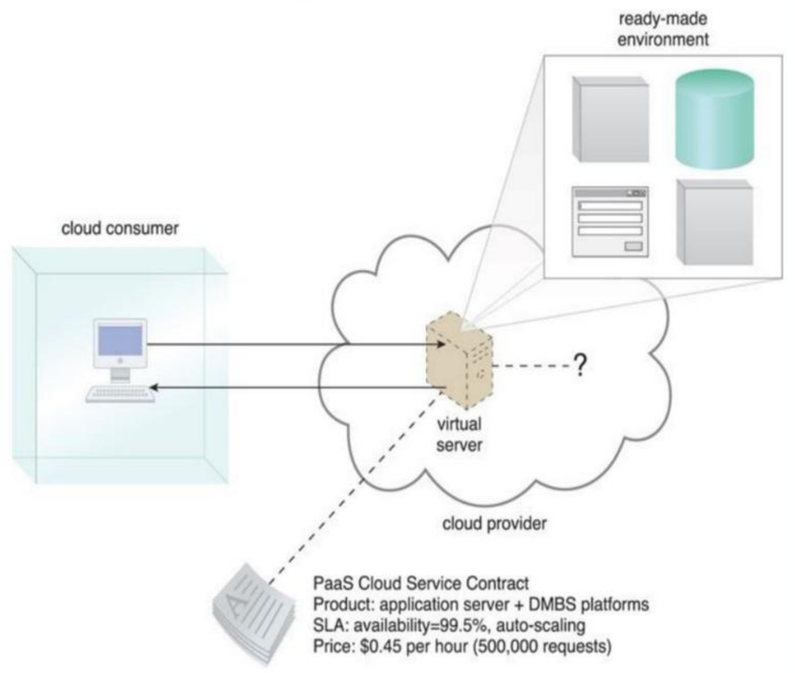


Figure 4.12. A cloud consumer is accessing a ready-made PaaS environment. The question mark indicates that the cloud consumer is intentionally shielded from the implementation details of the platform.

Cloud Delivery Models

- **Software-as-a-Service (SaaS):**
  - A software program positioned as a shared cloud service and made available as a "product" or generic utility represents the typical profile of a SaaS offering.
  - A cloud consumer is generally granted very limited administrative control over a SaaS implementation. It is most often provisioned by the cloud provider.

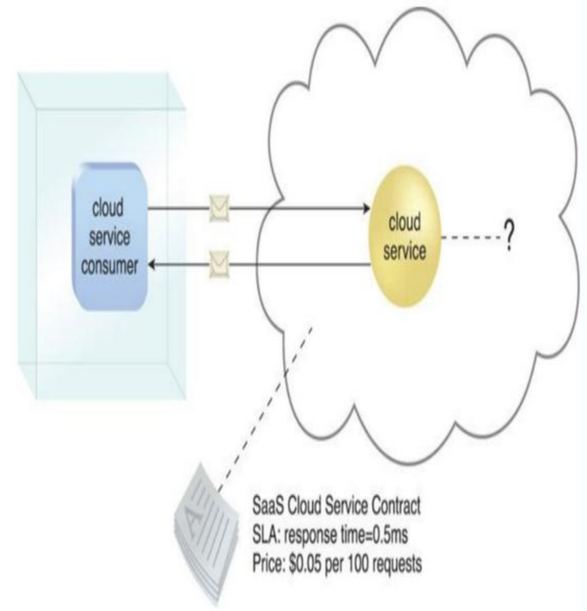


Figure 4.13. The cloud service consumer is given access the cloud service contract, but not to any underlying IT resources or implementation details.

Cloud Delivery Models

Cloud Delivery Model	Typical Level of Control Granted to Cloud Consumer	Typical Functionality Made Available to Cloud Consumer
SaaS	usage and usage-related configuration	access to front-end user-interface
PaaS	limited administrative	moderate level of administrative control over IT resources relevant to cloud consumer's usage of platform
IaaS	full administrative	full access to virtualized infra-structure-related IT resources and, possibly, to underlying physical IT resources

Cloud Delivery Models

Cloud Delivery Model	Common Cloud Consumer Activities	Common Cloud Provider Activities
SaaS	uses and configures cloud service	implements, manages, and maintains cloud service monitors usage by cloud consumers
PaaS	develops, tests, deploys, and manages cloud services and cloud-based solutions	pre-configures platform and provi-sions underlying infrastructure, middleware, and other needed IT resources, as necessary monitors usage by cloud consumers
IaaS	sets up and configures bare infrastructure, and installs, manages, and monitors any needed software	provisions and manages the physical processing, storage, networking, and hosting required monitors usage by cloud consumers

Cloud Deployment Models

Cloud Deployment Models

- A cloud deployment model represents a specific type of cloud environment, primarily distinguished by ownership, size, and access. There are four common cloud deployment models:
  - Public cloud
  - Private cloud
  - Community cloud
  - Hybrid cloud

Cloud Deployment Models

- **Public cloud:**
  - A public cloud is a publicly accessible cloud environment owned by a third-party cloud provider.
  - The IT resources on public clouds are usually provisioned via the previously described cloud delivery models and are generally offered to cloud consumers at a cost or are commercialized via other avenues (such as advertisement).
  - The cloud provider is responsible for the creation and on-going maintenance of the public cloud and its IT resources.

Cloud Deployment Models

- **Private cloud:**
  - A private cloud is owned by a single organization. Private clouds enable an organization to use cloud computing technology as a means of centralizing access to IT resources by different parts, locations, or departments of the organization.
  - With a private cloud, the same organization is technically both the cloud consumer and cloud provider.
  - Even though the private cloud may physically reside on the organization's premises, IT resources it hosts are still considered "cloud-based" as long as they are made remotely accessible to cloud consumers.

Cloud Deployment Models

- **Community cloud:**
  - A community cloud access is limited to a specific community of cloud consumers.
  - The community cloud may be jointly owned by the community members or by a third-party cloud provider that provisions a public cloud with limited access.
  - The member cloud consumers of the community typically share the responsibility for defining and evolving the community cloud.
  - Membership in the community does not necessarily guarantee access to or control of all the cloud's IT resources.
  - Parties outside the community are generally not granted access unless allowed by the community.

Cloud Deployment Models

- **Hybrid cloud:**
  - A hybrid cloud is a cloud environment comprised of two or more different cloud deployment models.
  - For example, a cloud consumer may choose to deploy cloud services processing sensitive data to a private cloud and other, less sensitive cloud services to a public cloud.

Cloud Deployment Models

- **Other Cloud Deployment Models:**
  - **Virtual Private Cloud** – Also known as a "dedicated cloud" or "hosted cloud," this model results in a self-contained cloud environment hosted and managed by a public cloud provider, and made available to a cloud consumer.
  - **Inter-Cloud** – This model is based on an architecture comprised of two or more inter-connected clouds.

Cloud-Enabling Technology

Cloud-Enabling Technology

- Modern-day clouds are supported by a set of primary technology components that collectively enable key features and characteristics associated with contemporary cloud computing such as:
  - **Internet Architecture**
  - **Data Center Technology**
  - **Virtualization Technology**
  - **Web Technology**
  - **Multitenant Technology**

Cloud-Enabling Technology

- **Internet Architecture:**
  - All clouds must be connected to a network.
  - The Internet, allow for the remote provisioning of IT resource.
- **Data Center Technology:**
  - At the foundation of any cloud is the data center hardware on which workloads run—servers, storage, and networking.
  - Grouping IT resources in close proximity with one another, rather than having them geographically dispersed, allows for power sharing, higher efficiency in shared IT resource usage, and improved accessibility.

Cloud-Enabling Technology

Virtualization Technology:

- **Virtualization is the process of converting a physical IT resource into a virtual IT resource.**
- The first step in creating a new virtual server through **virtualization software** is the allocation of physical IT resources, followed by the **installation of an operating system**.
- Virtualization software (called **hypervisor**) runs on a physical server called a host or physical host. A hypervisor is generally limited to one physical server and can therefore only create virtual images of that server.
- Virtualized IT resource management is often supported by **virtualization infrastructure management (VIM)** tools that collectively manage virtual IT resources.

Cloud-Enabling Technology

Virtualization Technology:

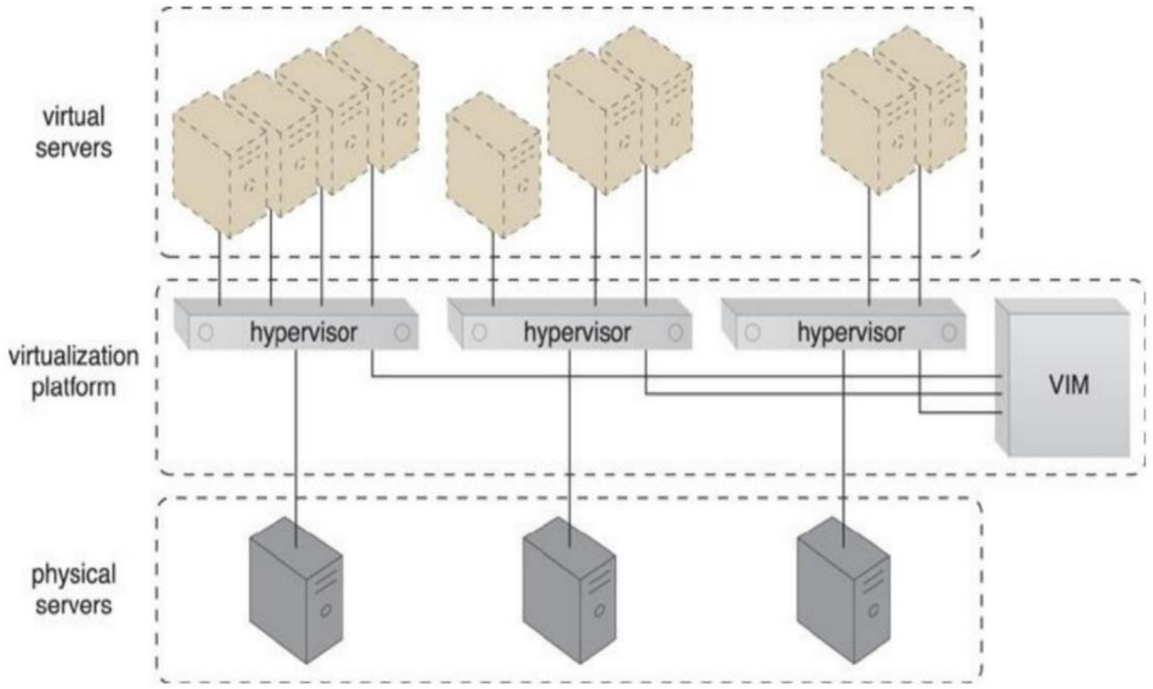


Figure 8.27. Virtual servers are created via individual hypervisor on individual physical servers. All three hypervisors are jointly controlled by the same VIM.

Cloud-Enabling Technology

Web Technology:

- Web technology is generally used as both the **implementation medium** and the **management interface** for cloud services.
- The World Wide Web is a system of interlinked IT resources that are accessed through the Internet. The two basic components of the Web are **the Web browser client** and the **Web server**.
- Three fundamental elements comprise the technology architecture of the Web:
  - **URL**: identifiers that point to Web-based resources
  - **HTTP**: communications protocol used to exchange content and data throughout the World Wide Web
  - **HTML, XML**: Markup languages provide a means of expressing Web data and metadata.

Cloud-Enabling Technology

Web Technology:

- For example, a Web browser can request to execute an action like read, write, update, or delete on a Web resource on the Internet, and proceed to identify and locate the Web resource through its URL.
- The request is sent using HTTP to the resource host, which is also identified by a URL.
- The Web server locates the Web resource and performs the requested operation, which is followed by a response being sent back to the client.
- The response may be comprised of content that includes HTML and XML statements.

Cloud-Enabling Technology

Multitenant Technology:

- The multitenant application design was created to enable multiple users (tenants) to access the same application logic simultaneously.
- Each tenant has its own view of the application that it uses.

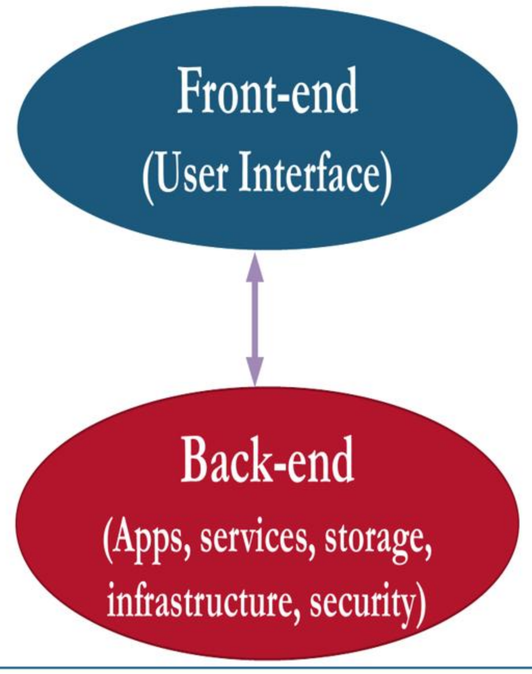
Cloud Architecture

Cloud Architecture

- Cloud Infrastructure Components:
  - There are two sides of the cloud environment. The **front end** is what's visible to the end user; in other words, it's the **user interface**. The **back-end** infrastructure is what runs the cloud.
  - This back end is made up of data center hardware, virtualization, applications, and services.
  - The front end communicates with the back end through **middleware**.
  - **Middleware** is software that enables one or more kinds of communication or connectivity between applications or application components in a distributed network.

Cloud Architecture

- Cloud Infrastructure Components:



Cloud Architecture

- Principles of cloud architecture:
  - Before you can design your cloud, you must first assess your existing environment and business needs. Here are just some of the questions your team will need to explore:
    - What are your existing workloads and applications? Where do they currently run, and who uses them?
    - How is your overall cloud utilization? Is it lower than it should be because it was designed to accommodate peak loads? Do you need to scale up to support new workloads?
    - Are you running into any bottlenecks in compute performance, memory, or networking?

Cloud Architecture

- Fundamental Cloud Architecture examples:
  - Workload Distribution Architecture
  - Dynamic Scalability Architecture
  - Elastic Resource Capacity Architecture
  - Service Load Balancing Architecture
  - …..and many more

Cloud Architecture: Fundamental Cloud Architecture

- Workload Distribution Architecture:
  - IT resources can be horizontally scaled via the **addition of one or more identical IT resources**, and a **load balancer** that provides runtime logic capable of evenly distributing the workload among the available IT resources.
  - The resulting workload distribution architecture reduces both IT resource over-utilization and under-utilization.

Cloud Architecture: Fundamental Cloud Architecture

- Workload Distribution Architecture:

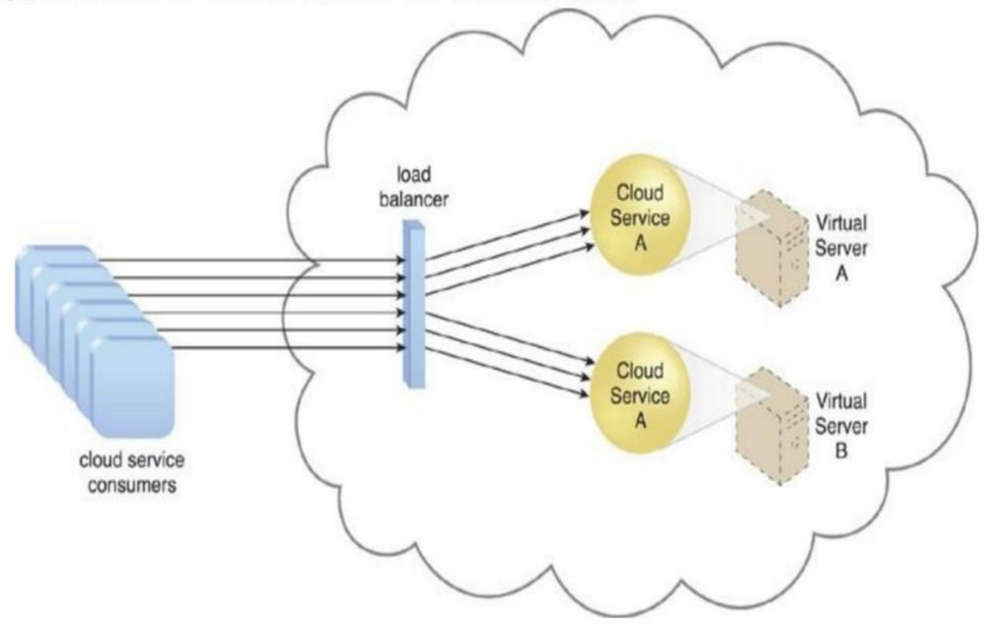


Figure 11.1. A redundant copy of Cloud Service A is implemented on Virtual Server B. The load balancer intercepts cloud service consumer requests and directs them to both Virtual Servers A and B to ensure even workload distribution.

Cloud Architecture: Fundamental Cloud Architecture

Dynamic Scalability Architecture:

- Dynamic allocation enables **variable utilization** as dictated by usage demand fluctuations, since unnecessary IT resources are efficiently reclaimed without requiring manual interaction.
- The **automated scaling listener** is configured with **workload thresholds** that dictate when new IT resources need to be added to the workload processing. This mechanism can be provided with logic that determines how many additional IT resources can be dynamically provided, based on the terms of a given cloud consumer's provisioning contract.

Cloud Architecture: Fundamental Cloud Architecture

Dynamic Scalability Architecture:

- The following types of dynamic scaling are commonly used:
  - **Dynamic Horizontal Scaling:** IT resource instances are ***scaled out and in*** to handle fluctuating workloads. The automatic scaling listener monitors requests and signals resource replication to initiate IT resource duplication.
  - **Dynamic Vertical Scaling:** IT resource instances are ***scaled up and down*** when there is a need to adjust the processing capacity of a single IT resource.
  - **Dynamic Relocation:** The IT resource is relocated to a host with more capacity.

Dynamic Scalability Architecture

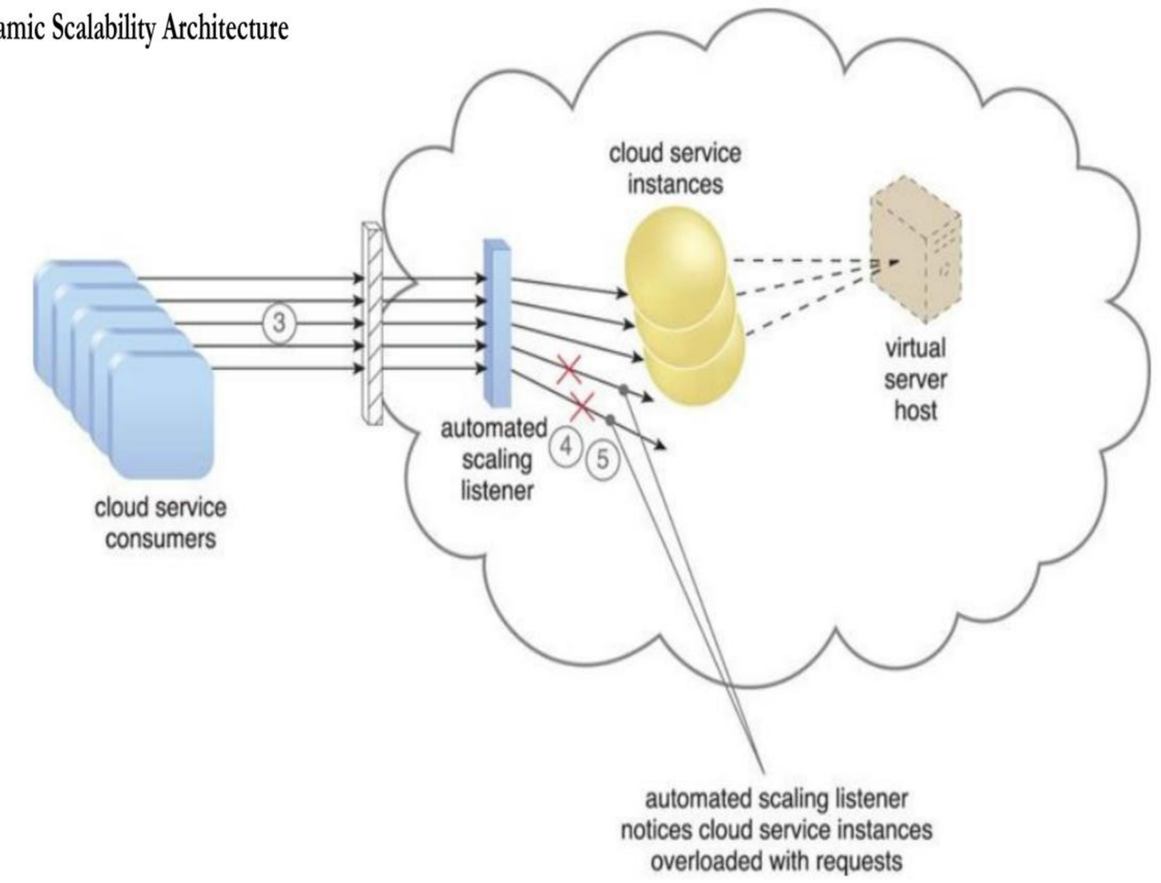


Figure 11.6. The number of requests coming from cloud service consumers increases (3). The workload exceeds the performance thresholds. The automated scaling listener determines the next course of action based on a predefined scaling policy (4). If the cloud service implementation is deemed eligible for additional scaling, the automated scaling listener initiates the scaling process (5).

Dynamic Scalability Architecture

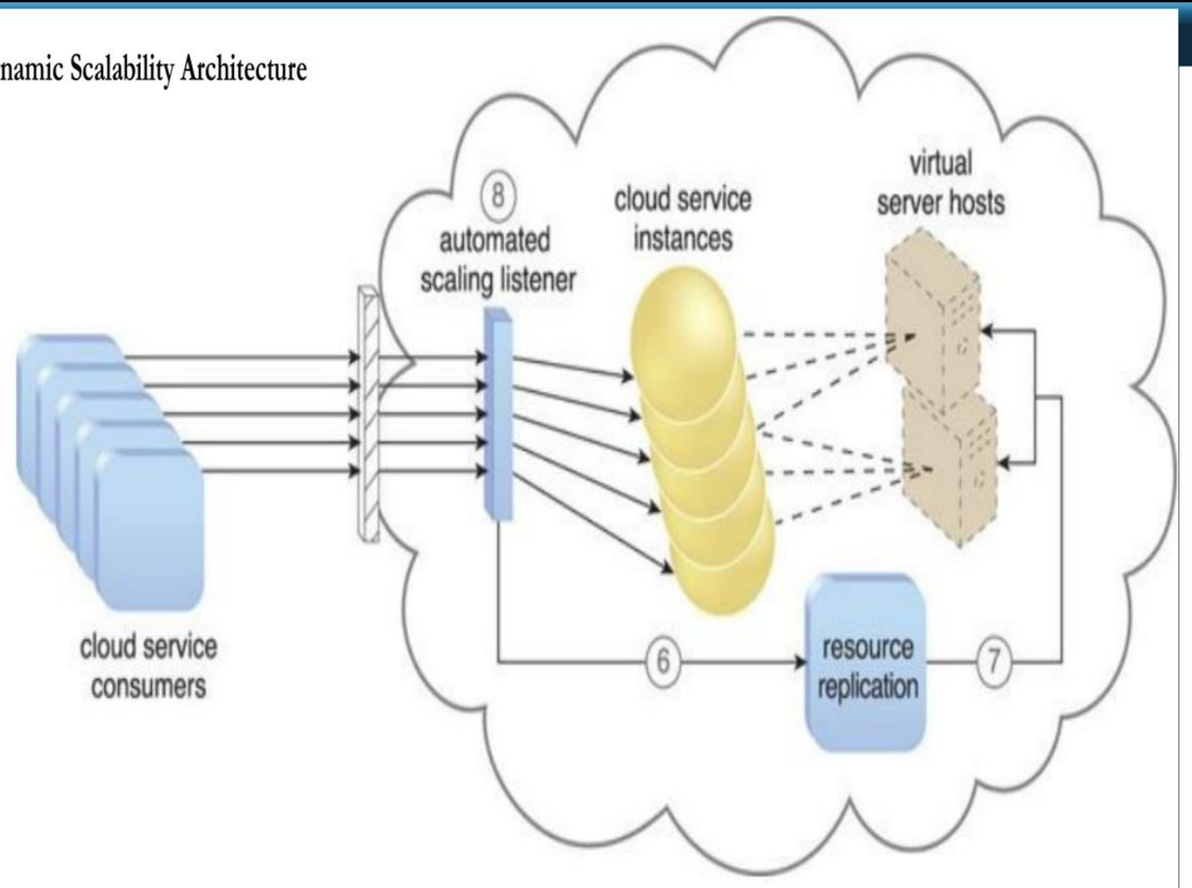


Figure 11.7. The automated scaling listener sends a signal to the resource replication mechanism (6), which creates more instances of the cloud service (7). Now that the increased workload has been accommodated, the automated scaling listener resumes monitoring and detracting and adding IT resources, as required (8).

Reference

- Cloud Computing: Concepts, Technology & Architecture by Thomas Erl et al. Prentice Hall, 2013.
  - The source of all images and Tables are this reference.

Thank You

Lecture 7

Cloud Computing II

(Cloud Computing Platforms: Microsoft Azure and AWS)

Dr. Lydia Wahid

Agenda

- Microsoft Azure
- Amazon Web Services (AWS)
- Class Assignment
- Appendix A: Definitions
- Appendix B: AutoML classification

Microsoft Azure

What is Microsoft Azure?

- Azure is a cloud computing platform that allows you to access and manage cloud services and resources provided by Microsoft.
- Azure provides the following categories of services: **AI + machine learning, Analytics, Compute, Databases, Developer tools, DevOps, Hybrid + multicloud, Identity, Integration, Internet of Things, Management and governance, Media, Migration, Mixed reality, Mobile, Networking, Security, Storage, Virtual desktop infrastructure, Web.**
- Each category includes a list of services that Azure provides.

Microsoft Azure Services

- **Artificial Intelligence + Machine learning:**
  - Develop applications with the latest AI and ML capabilities.
  - The following are some of the services provided in this category:
    - Customize your own state-of-the-art computer vision models
    - Create bots and connect them across channels
    - Accelerate information extraction from documents
    - Translation, speech to text, text to speech, Language understanding

Microsoft Azure Services

- **Analytics:**
  - Gather, store, process, analyze, and visualize data of any variety, volume, or velocity
  - The following are some of the services provided in this category:
    - Design AI with Apache Spark-based analytics
    - Provision cloud Hadoop, Spark, R Server, HBase, and Storm clusters (check Appendix A)
    - Predictive analytics, machine learning, and statistical modeling for big data using Microsoft R server
    - Real-time analytics on fast-moving streaming data

Microsoft Azure Services

- **Compute:**
  - Access cloud compute capacity and scale on demand—and only pay for the resources you use
  - The following are some of the services provided in this category:
    - Create, manage, operate, and optimize HPC (check Appendix A) and big compute clusters of any scale
    - Create and provision virtual machines in Windows and Linux
    - Deploy and operate always-on, scalable, distributed apps
    - Run your VMware workloads on Azure

Microsoft Azure Services

- **Containers:**
  - Develop and manage your containerized applications
  - Containers are executable units of software in which application code is packaged, along with its libraries and dependencies, so that it can be run anywhere, whether it be on the cloud or on-premise infrastructure
- **DevOps:**
  - It is a combination of the terms development and operations. DevOps is a methodology meant to improve work throughout the software development lifecycle. You can visualize a DevOps process as an infinite loop, comprising these steps: plan, code, build, test, release, deploy, operate, monitor,….

Microsoft Azure Services

- **Hybrid + multicloud:**
  - Build and run hybrid apps across cloud boundaries
- **Identity:**
  - Manage user identities and access to protect against advanced threats across devices, data, apps, and infrastructure
- **Integration:**
  - Integrate on-premises and cloud-based applications, data, and processes across your enterprise

Microsoft Azure Services

- **Migration:**
  - Migrate to the cloud with guidance, tools, and resources
- **Mixed reality:**
  - Blend your physical and digital worlds
  - Automatically align and anchor 3D content to objects in the physical world

Microsoft Azure Machine Learning

- Azure Machine Learning includes several **resources** and **assets** to enable you to perform your machine learning tasks. These resources and assets are needed to run any job.
- **Resources**: setup or infrastructural resources needed to run a machine learning workflow. Resources include:
  - Workspace
  - Datastore
  - Compute
- **Assets**: created using Azure Machine Learning commands or as part of a training/scoring (check Appendix A) run. They include:
  - Model
  - Data
  - Environment
  - Component

Microsoft Azure Machine Learning: Resources

- **Workspace**
  - The workspace is the top-level resource for Azure Machine Learning, providing a **centralized place** to work with all the artifacts you create when you use Azure Machine Learning.
  - The workspace keeps a history of all jobs, including logs, metrics, output, and a snapshot of your scripts.
- **Datastore**
  - Azure Machine Learning datastores securely keep the connection information to your data storage on Azure, so you don't have to code it in your scripts.

Microsoft Azure Machine Learning: Resources

- **Compute:** Azure Machine Learning supports the following types of compute:
  - **Compute cluster** - a cluster of CPU or GPU compute nodes in the cloud.
  - **Compute instance** - a fully configured and managed development environment in the cloud
  - **Inference cluster** - used to deploy trained machine learning models to Azure Kubernetes Service (AKS) (check Appendix A)
  - **Attached compute** - You can attach your own compute resources to your workspace and use them for training and inference.

Microsoft Azure Machine Learning: Assets

- **Model**
  - Azure machine learning models consist of the binary file(s) that represent a machine learning model and any corresponding metadata.
- **Data**
  - Azure Machine Learning allows you to work with different types of data:
    - URIs (a location in local/cloud storage)
    - Tables (a tabular data abstraction)
    - Primitives (string, Boolean, number,..)

Microsoft Azure Machine Learning: Assets

- **Environment**
  - An encapsulation that specifies the **software packages**, **environment variables**, and **software settings** around your training and scoring scripts.
  - Azure Machine Learning supports two types of environments: curated and custom:
    - **Curated environments** Default. Intended to be used as is, they contain collections of Python packages and settings.
    - In **custom environments**, you're responsible for setting up your environment and installing packages or any other dependencies that your training or scoring script needs.

Microsoft Azure Machine Learning: Assets

- **Component**
  - An Azure Machine Learning component is a self-contained **piece of code** that does one step in a machine learning pipeline.
  - Components can do tasks such as **data processing**, **model training**, **model scoring**, and so on.
  - A component is analogous to a function - it has a name, parameters, expects input, and returns output.

Microsoft Azure AutoML

- Automated machine learning, also referred to as automated ML or AutoML, is the process of automating the iterative tasks of machine learning model development.
- Automated ML in Azure Machine Learning is based on a breakthrough from Microsoft Research division.

Microsoft Azure AutoML

- **How does AutoML work?**
  - During training, Azure Machine Learning creates a number of pipelines in parallel that try different algorithms and parameters for you.
  - The service iterates through ML algorithms paired with feature selections, where each iteration produces a model with a training score.
  - The better the score for the metric you want to optimize for, the better the model is considered to "fit" your data. It will stop once it hits the exit criteria defined in the experiment.

Microsoft Azure AutoML

- Using **Azure Machine Learning**, you can design and run your automated ML training experiments with these steps:
  1. **Identify the ML problem** to be solved.
  2. **Choose whether you want a code-first experience or a no-code studio web experience**:
     - Code-first experience: you can use the Azure Machine Learning SDKv2 or the Azure Machine Learning CLIv2.
     - Limited/no-code experience: you can use the web interface in Azure Machine Learning studio at https://ml.azure.com (this link will take you to your account if signed in)

Microsoft Azure AutoML

- Using **Azure Machine Learning**, you can design and run your automated ML training experiments with these steps:
  3. **Specify the source of the labeled training data**: You can bring your data to Azure Machine Learning either from your local machine or an existing cloud-based storage.
  4. **Configure the automated machine learning parameters** that determine how many iterations over different models, hyperparameter settings, advanced preprocessing/featurization, and what metrics to look at when determining the best model.
  5. **Submit the training job**
  6. **Review the results**

Microsoft Azure AutoML: Demo Example

- **Task:** Train a classification model with AutoML in the Azure Machine Learning studio.
- Check Appendix B

Microsoft Azure Cognitive Service

- Azure Cognitive Service for Language is a cloud-based service that provides Natural Language Processing (NLP) features for understanding and analyzing text such as:
  - Named Entity Recognition (NER), Personally Identifying Information (PII)
  - Text analytics for health
  - Key phrase extraction
  - Language detection
  - Sentiment analysis and opinion mining
  - Question answering
  - Summarization
  - Custom Named Entity Recognition (Custom NER)
  - Custom text classification
  - Conversational language understanding

Microsoft Azure Cognitive Service

- You can build applications using the web-based Language Studio, REST APIs, and client libraries.
- You can access the language studio from: language.cognitive.azure.com
- Note that you first create a **language resource** in azure portal.

Amazon Web Services (AWS)

Amazon Web Services (AWS)

- Amazon Web Services (AWS) provide on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered, pay-as-you-go basis.

- AWS categories of services include **computing**, **storage**, **networking**, **database**, **analytics**, **application services**, **deployment**, **management**, **machine learning**, **mobile**, **developer tools**, and tools for the **Internet of Things**.

Amazon Web Services (AWS)

- **Amazon EC2:** Amazon Elastic Compute Cloud
  - Allows users to create and rent virtual computers to run their own applications.
  - Allows users to configure their own virtual machine (which is called an "instance") such as processor, storage, networking, operating system.
- **Amazon S3:** Amazon Simple Storage Service
  - Provides object storage
  - Manages data as objects (an object includes: data itself, metadata, and a globally unique identifier)

Amazon Web Services (AWS)

- **Amazon VPC:** Amazon Virtual Private Cloud
  - Provides provisioning to a logically isolated section of AWS Cloud.
  - Customers access the EC2 over an IPsec (Check Appendix A) based virtual private network.
- **Amazon EMR:** Amazon Elastic MapReduce
  - It is cluster platform for running big data frameworks, such as Apache Hadoop and Apache Spark, on AWS to process and analyze vast amounts of data.

Amazon Web Services (AWS): EMR

- The central component of Amazon EMR is the **cluster**.
- A cluster is a collection of **Amazon EC2 instances**. Each instance in the cluster is called a *node*.
- Each node has a role within the cluster, referred to as the *node type*.
  - **Primary node:** A node that manages the cluster by running software components to coordinate the distribution of data and tasks among other nodes for processing.
  - **Core node:** A node with software components that run tasks and store data in the HDFS on your cluster.
  - **Task node:** A node with software components that only runs tasks and does not store data in HDFS. Task nodes are optional.

Amazon Web Services (AWS): EMR

- When you launch your cluster, you choose the frameworks and applications to install for your data processing needs such as: Hive, Hadoop, Spark, and so on.
- Amazon EMR service architecture consists of several layers, each of which provides certain capabilities and functionality to the cluster:
  - Storage
  - Cluster resource management
  - Data processing frameworks
  - Applications and programs

Amazon Web Services (AWS): EMR

- Amazon EMR service architecture layers:
  - **Storage:** There are several different types of storage:
    - **HDFS:** it is reclaimed when you terminate a cluster
    - **EMR File System (EMRFS):** directly access data stored in Amazon S3 as if it were a file system like HDFS
    - **Local file system:** refers to a locally connected disk

Note: When you create a Hadoop cluster, each node is created from an Amazon EC2 instance. Data on instance store volumes persists only during the lifecycle of its Amazon EC2 instance.)

Amazon Web Services (AWS): EMR

- Amazon EMR service architecture layers:
  - **Cluster resource management:**
    - responsible for managing cluster resources and scheduling the jobs for processing data.
    - By default, Amazon EMR uses YARN.
  - **Data processing frameworks:**
    - It is the engine used to process and analyze data.
    - The main processing frameworks available for Amazon EMR are Hadoop MapReduce and Spark.

Amazon Web Services (AWS): EMR

- Amazon EMR service architecture layers:
  - **Applications and programs:**
    - Amazon EMR supports many applications, such as Hive, Pig, and the Spark Streaming library.
    - You can use libraries and languages to interact with the applications that you run in Amazon EMR. For example, you can use Java, Hive, or Pig with MapReduce or Spark Streaming, Spark SQL, MLlib, and GraphX with Spark.

References

- azure.microsoft.com/en-us/products/
- learn.microsoft.com/en-us/azure/machine-learning/concept-azure-machine-learning-v2?tabs=sdk
- learn.microsoft.com/en-us/azure/machine-learning/concept-automated-ml
- learn.microsoft.com/en-us/azure/cognitive-services/language-service/
- aws.amazon.com/products
- docs.aws.amazon.com/emr/latest/ManagementGuide/emr-what-is-emr.html

Class Assignment

Class Assignment

- You are required to use Microsoft Azure and apply it on a problem of your choice.
- The problem can fall in the following categories: Classification, Regression, Time series forecasting, Natural language processing and Computer vision.
- Bonus grades will be given for the following:
  - Choosing either Time series forecasting or NLP or Computer vision problem.
  - OR using Azure Machine Learning SDKv2
  - OR using Azure Machine Learning CLIv2
- Delivery date: During the lecture of week12
- Assignment grade: 5 marks (without the bonus)

Appendix A: Definitions

Appendix A: Definitions

- **Storm:** Apache Storm is a distributed stream processing computation framework
- **Microsoft R Server:** is now Microsoft Machine Learning Server which is a flexible enterprise platform for analyzing data at scale, building intelligent apps, and discovering valuable insights across a business now with full support for Python and R.
- **HPC:** High-performance computing (HPC), also called "big compute", uses a large number of CPU or GPU-based computers to solve complex mathematical tasks.

Appendix A: Definitions

- **Model scoring:** In machine learning, model scoring is the process of assessing the accuracy of a model.
- **Azure Kubernetes Service:** Azure Kubernetes Service (AKS) is a cloud service that lets you run Kubernetes clusters without managing the underlying infrastructure. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications across multiple nodes.
- **Deploy:** Software deployment is to make a software system available to its intended users.

Appendix A: Definitions

- **IPsec:** Internet Protocol Security (IPsec) is a secure network protocol suite that authenticates and encrypts packets of data to provide secure encrypted communication between two computers over an Internet Protocol network. It is used in virtual private networks (VPNs).

Appendix B: AutoML classification

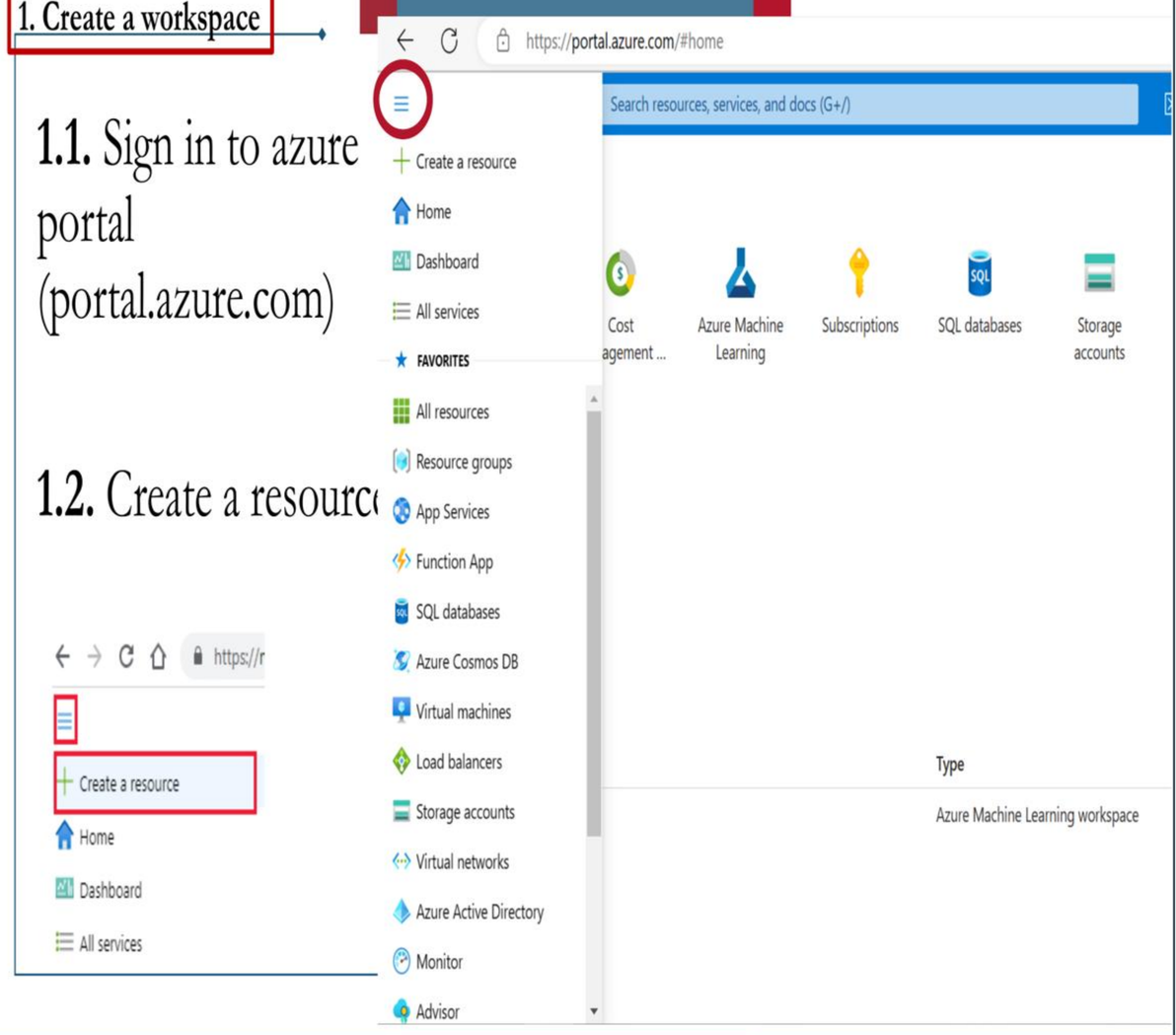
Microsoft Azure AutoML: Demo Example

1. Create a workspace
2. Go to Azure Machine Learning studio
3. Create and load dataset
4. Configure job
5. Configure task
6. Explore the models

1. Create a workspace

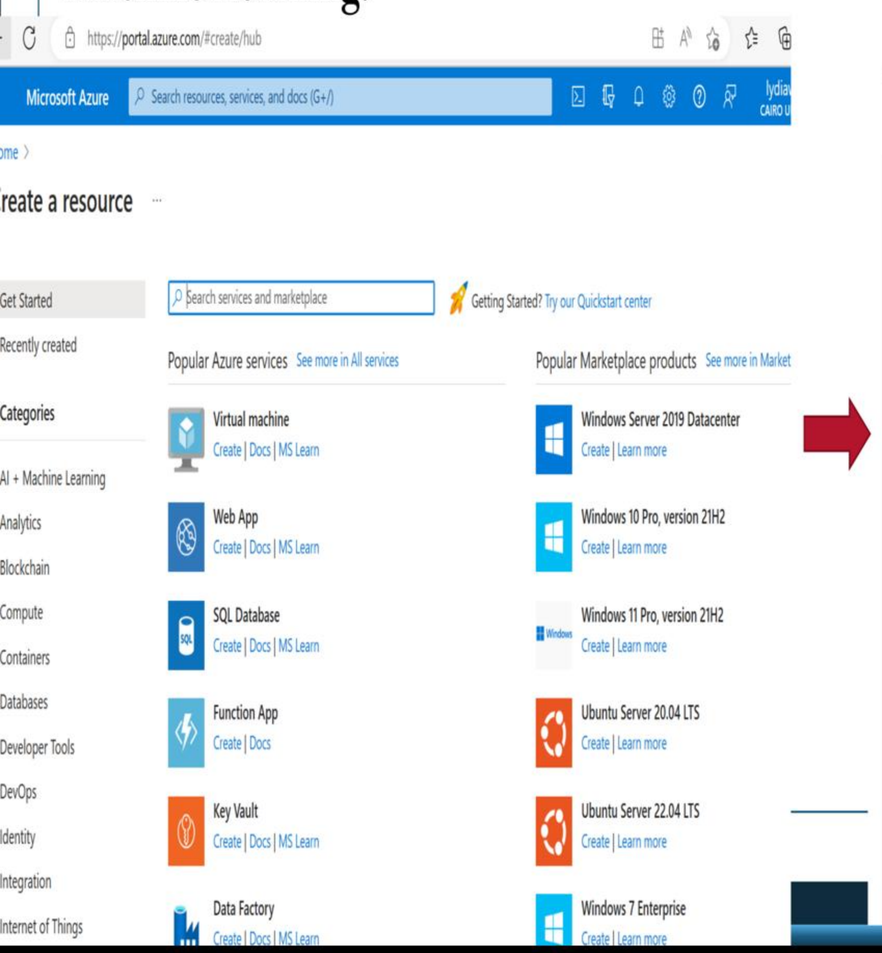
1.1. Sign in to azure portal (portal.azure.com)

1.2. Create a resourc



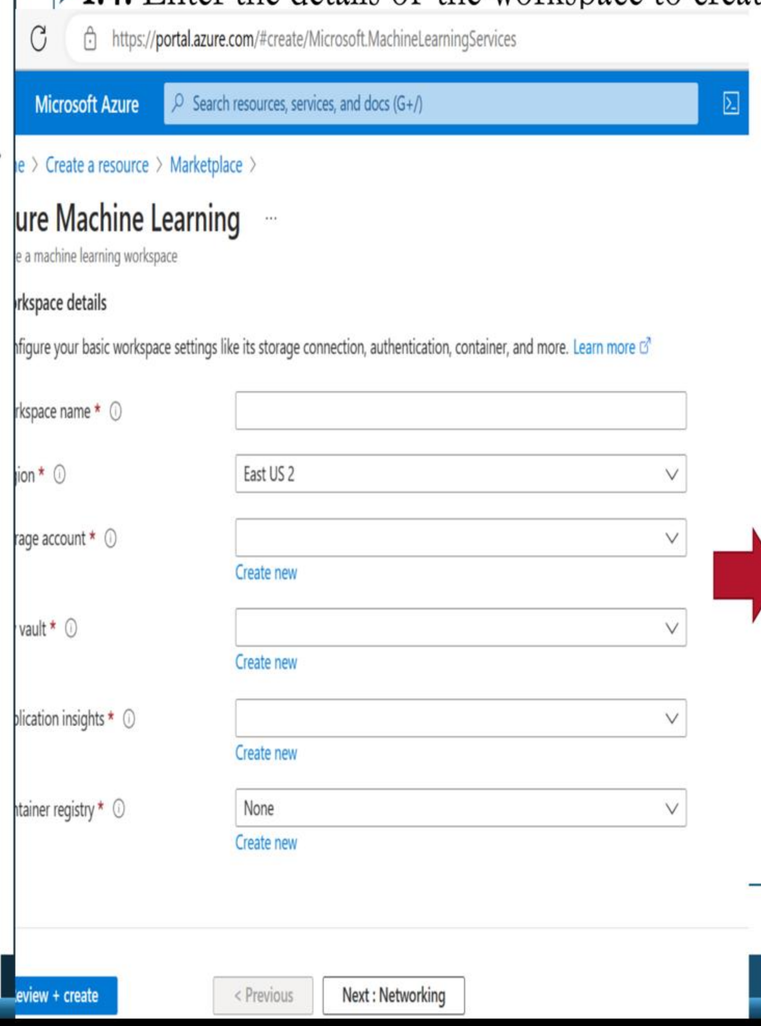
1. Create a workspace

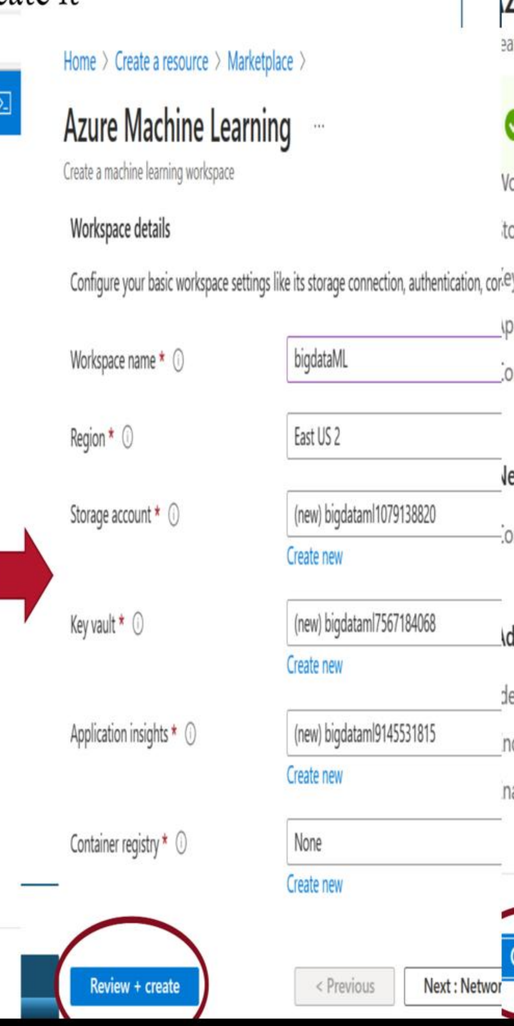
1.3. Use the search bar to find and create **Azure Machine Learning.**

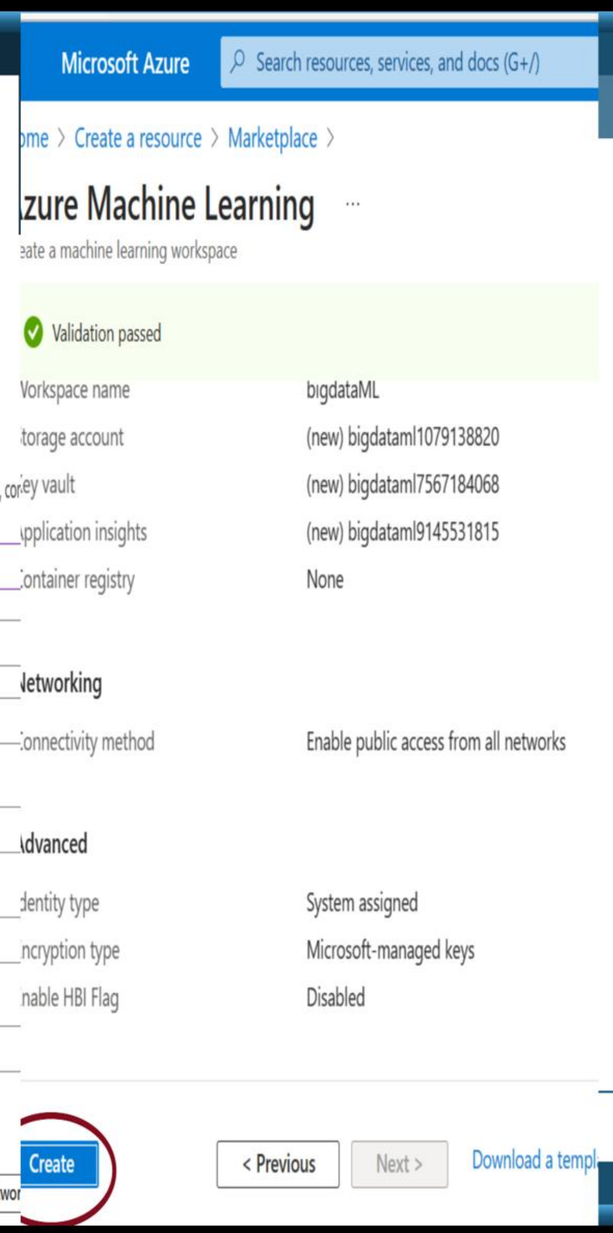


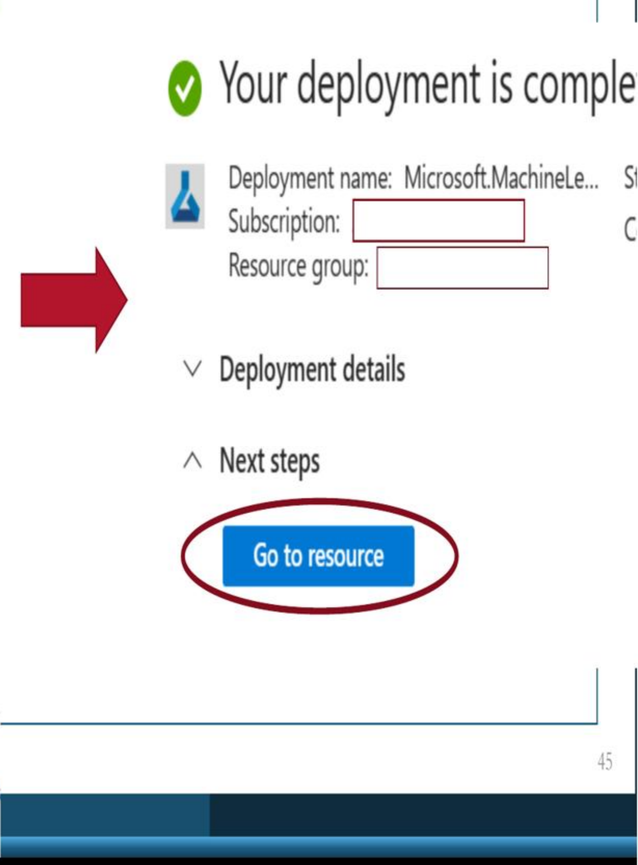
1. Create a workspace

1.4. Enter the details of the workspace to create it

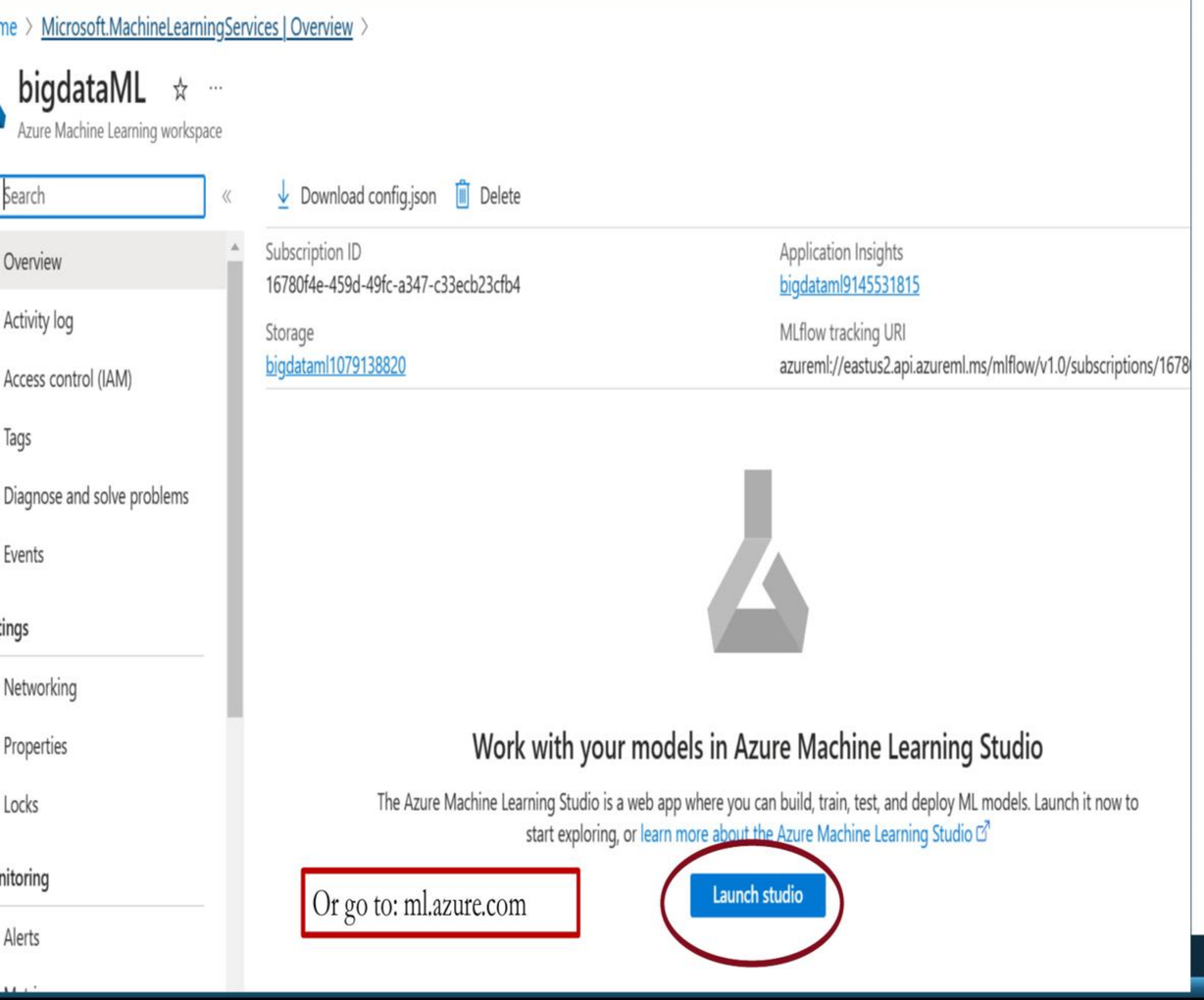








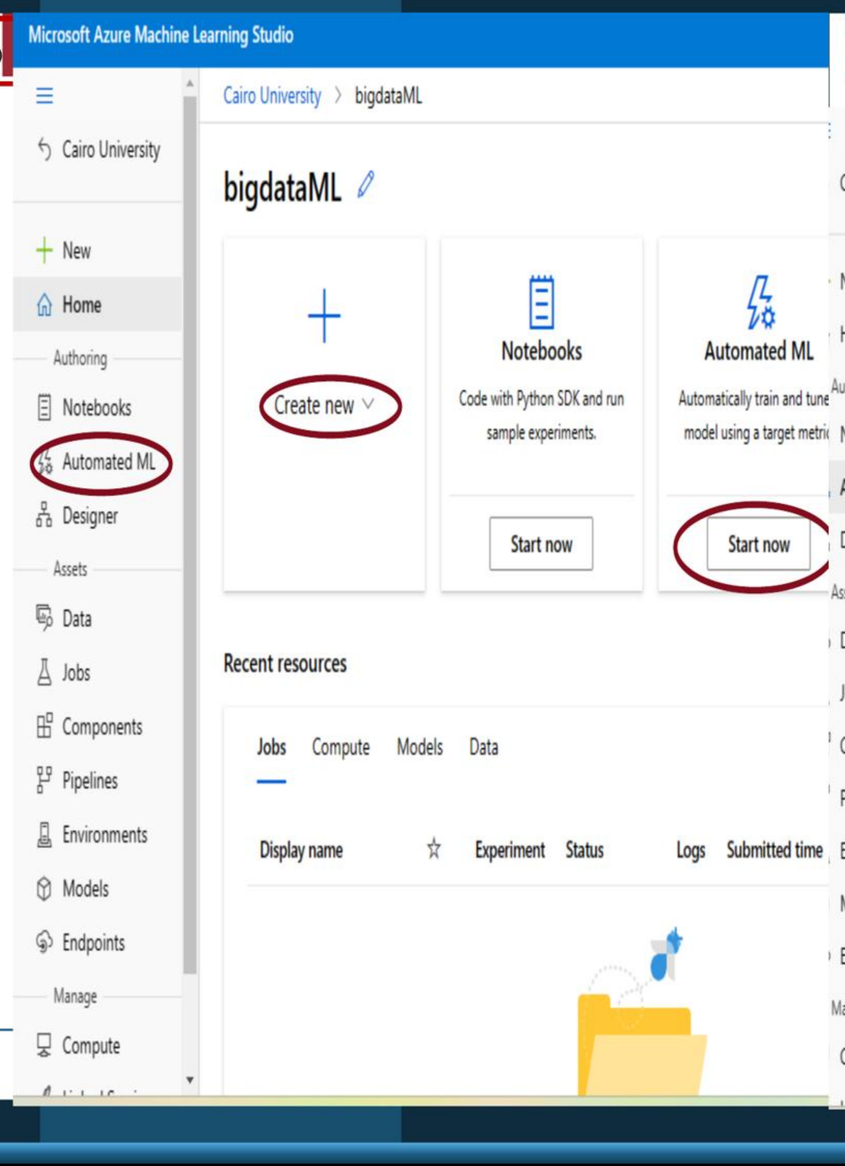
2. Go to Azure Machine Learning studio



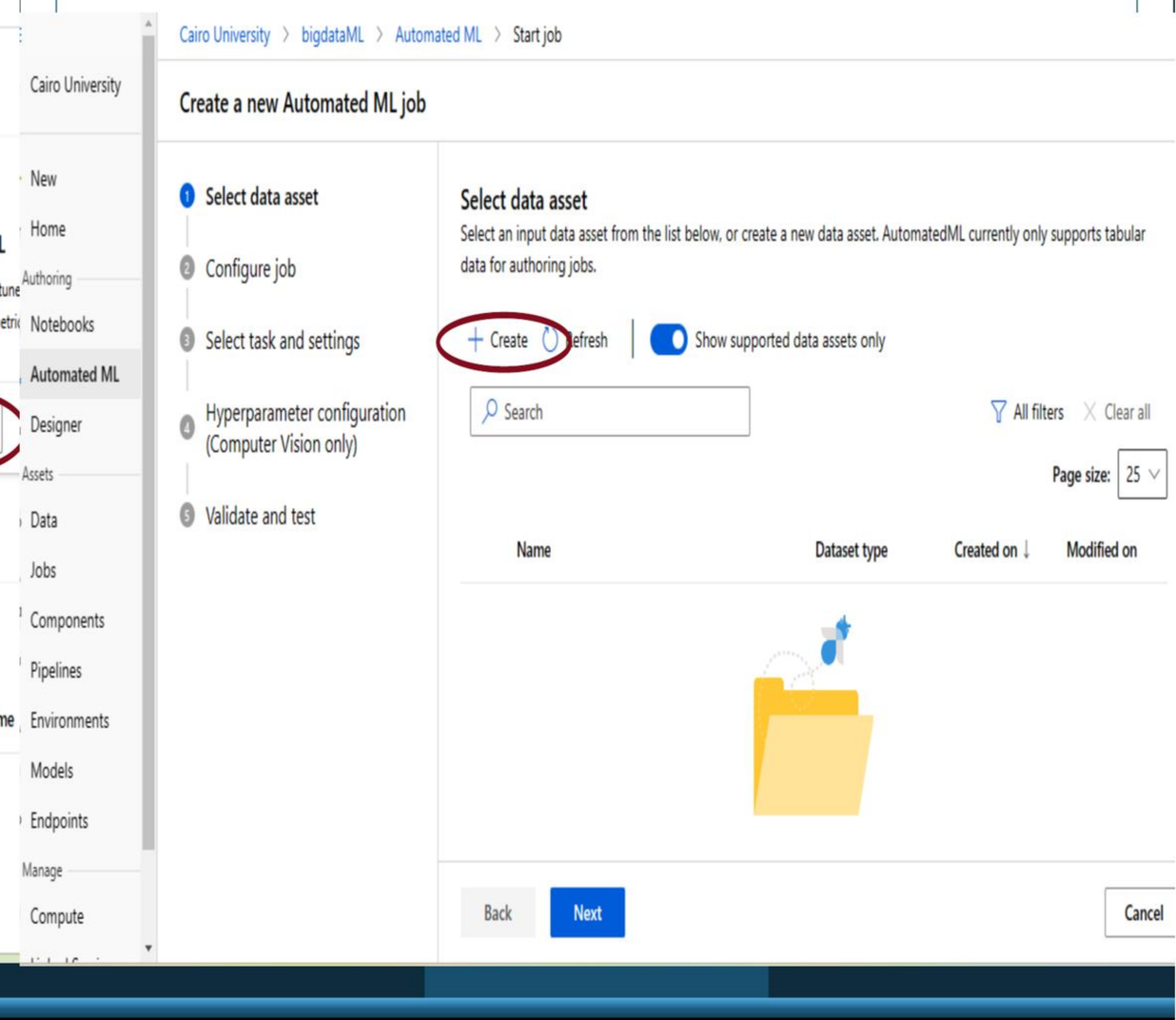
2. Go to Azure Machine Learning studio

2.1 Select your subscription and the workspace you created.

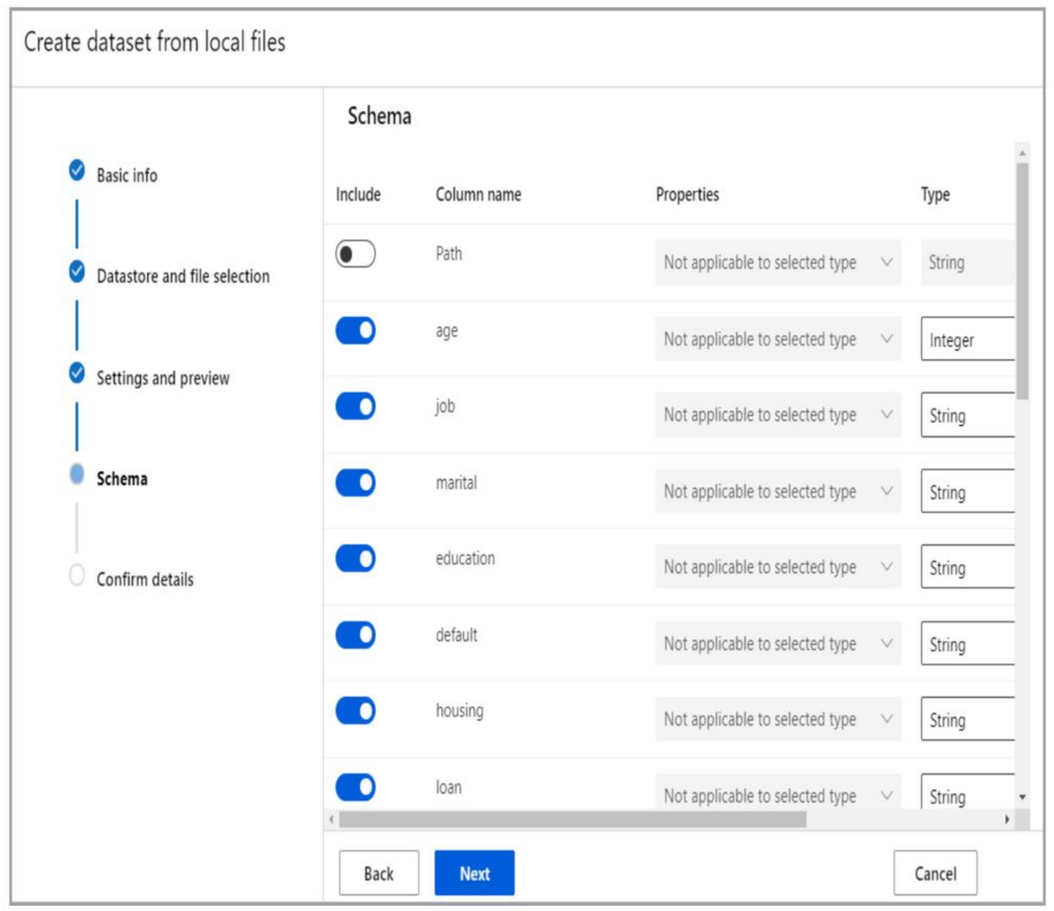
2.2. Create a new AutoML job: In the left pane, select **Automated ML** OR from **Create new** OR Click **Start now** in the Automated ML icon.



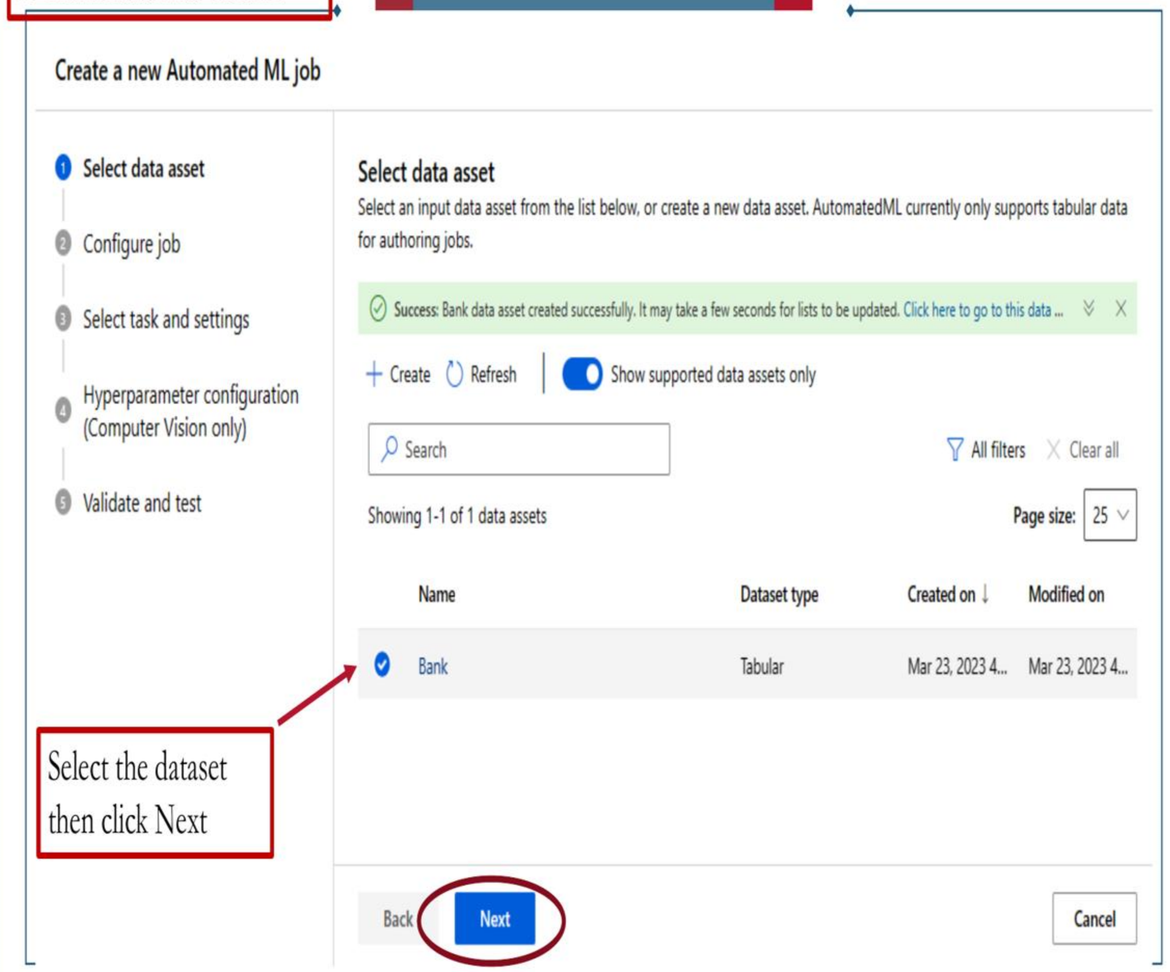
3. Create and load dataset



3. Create and load dataset



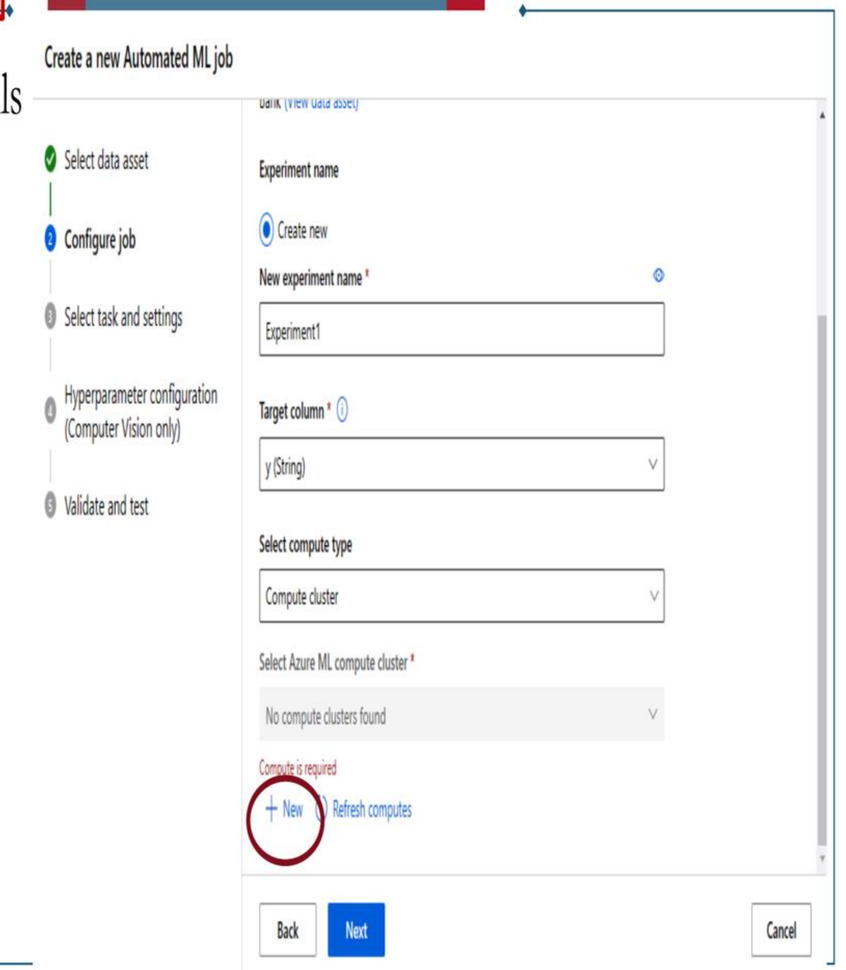
3. Create and load dataset



Select the dataset then click Next

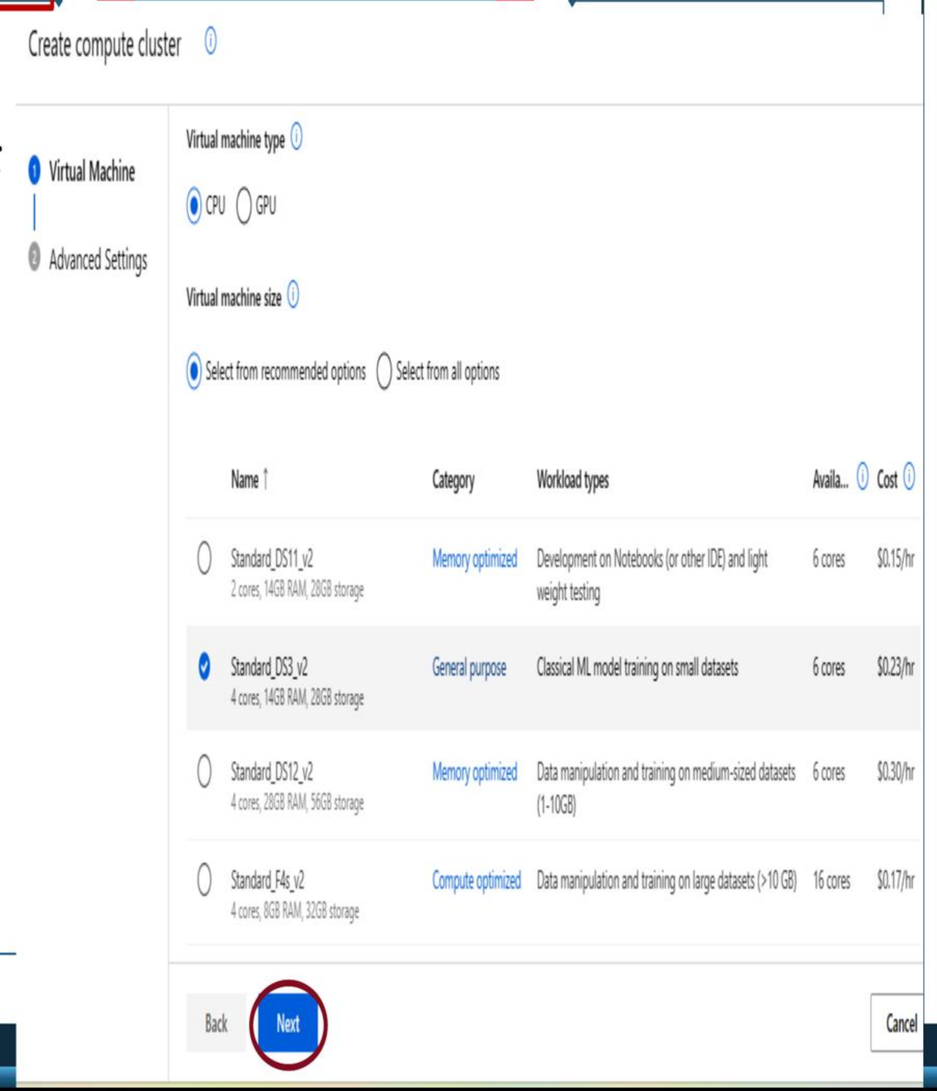
4. Configure job

4.1. Enter the details



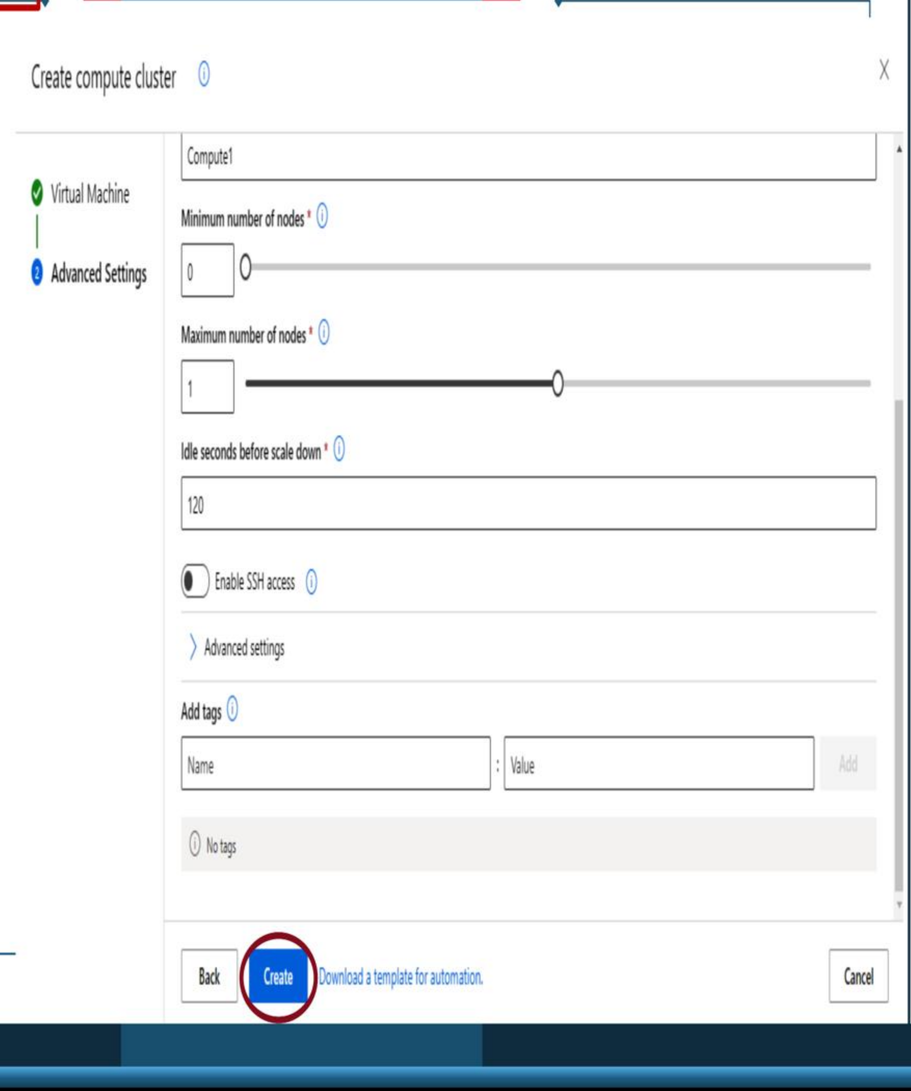
4. Configure job

4.2. Create compute cluster

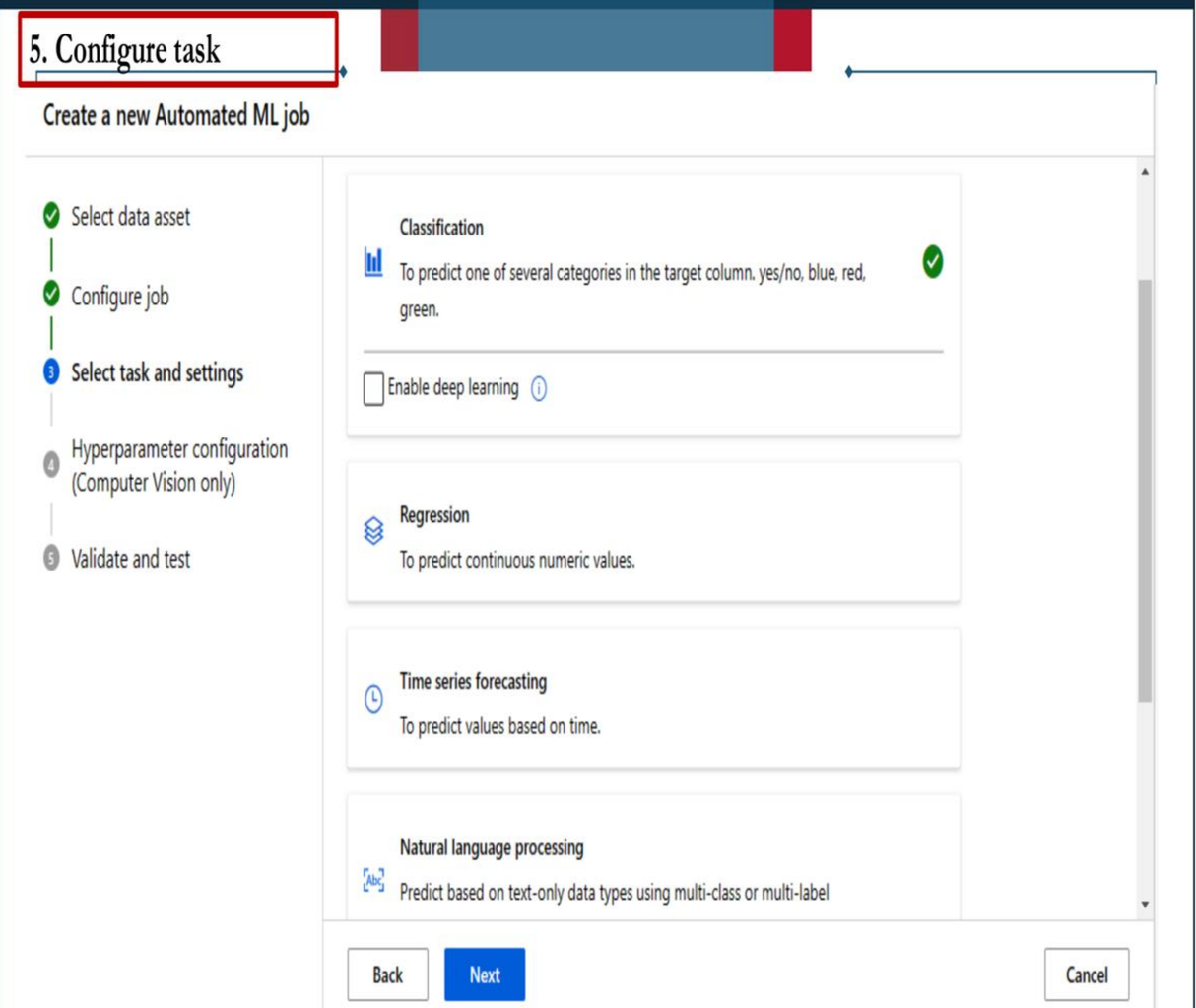


4. Configure job

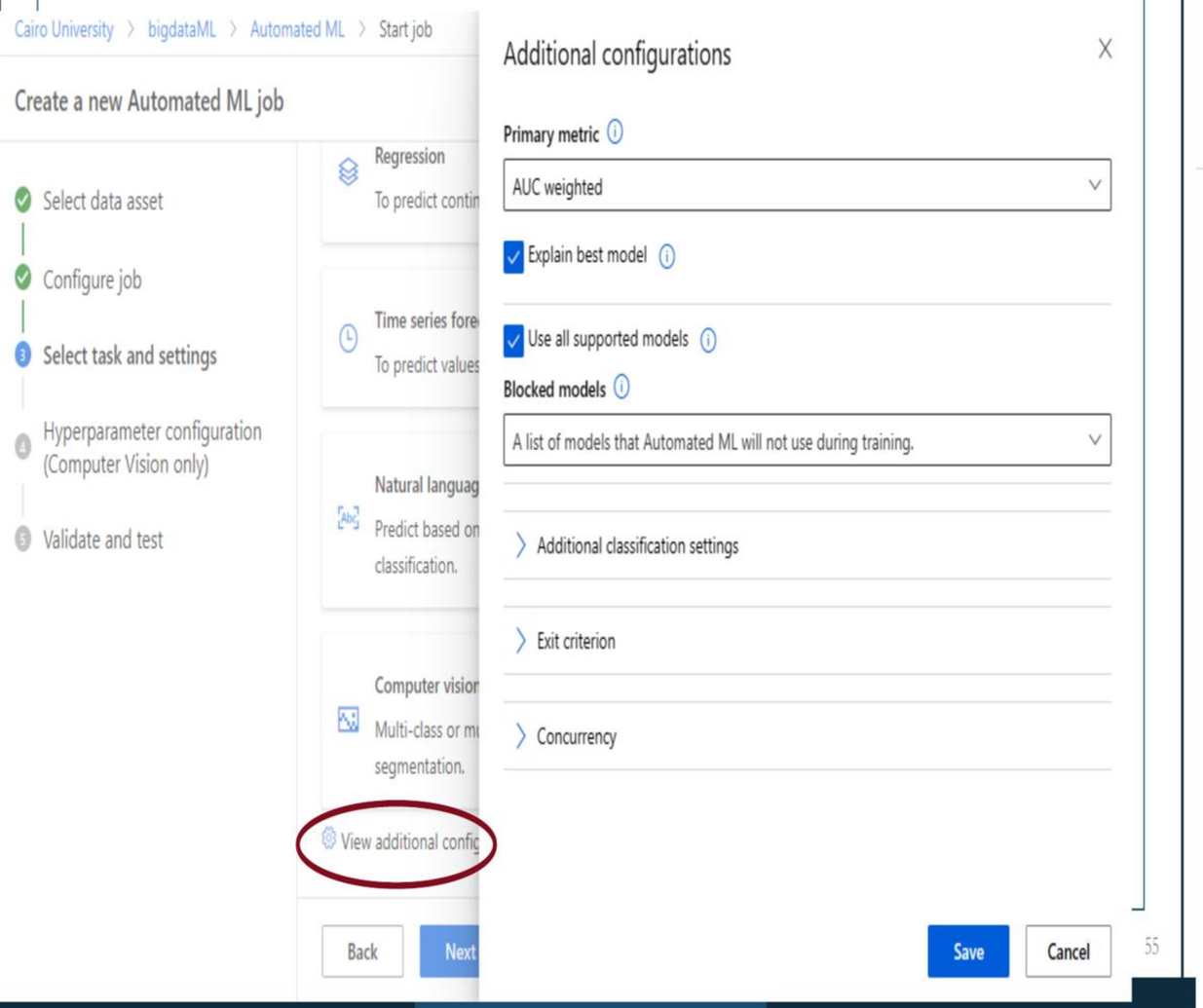
4.3. Configure your cluster and then click create



5. Configure task

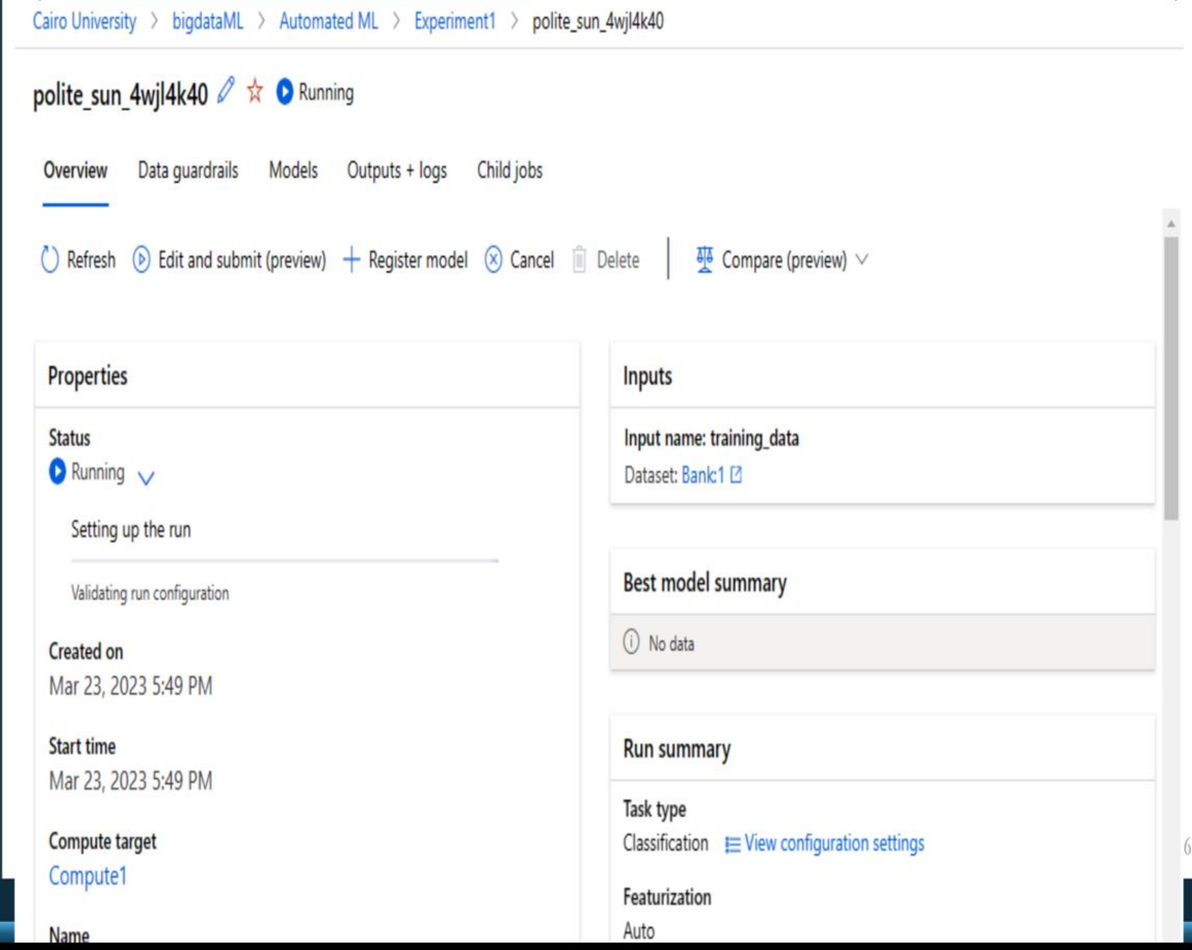


5. Configure task



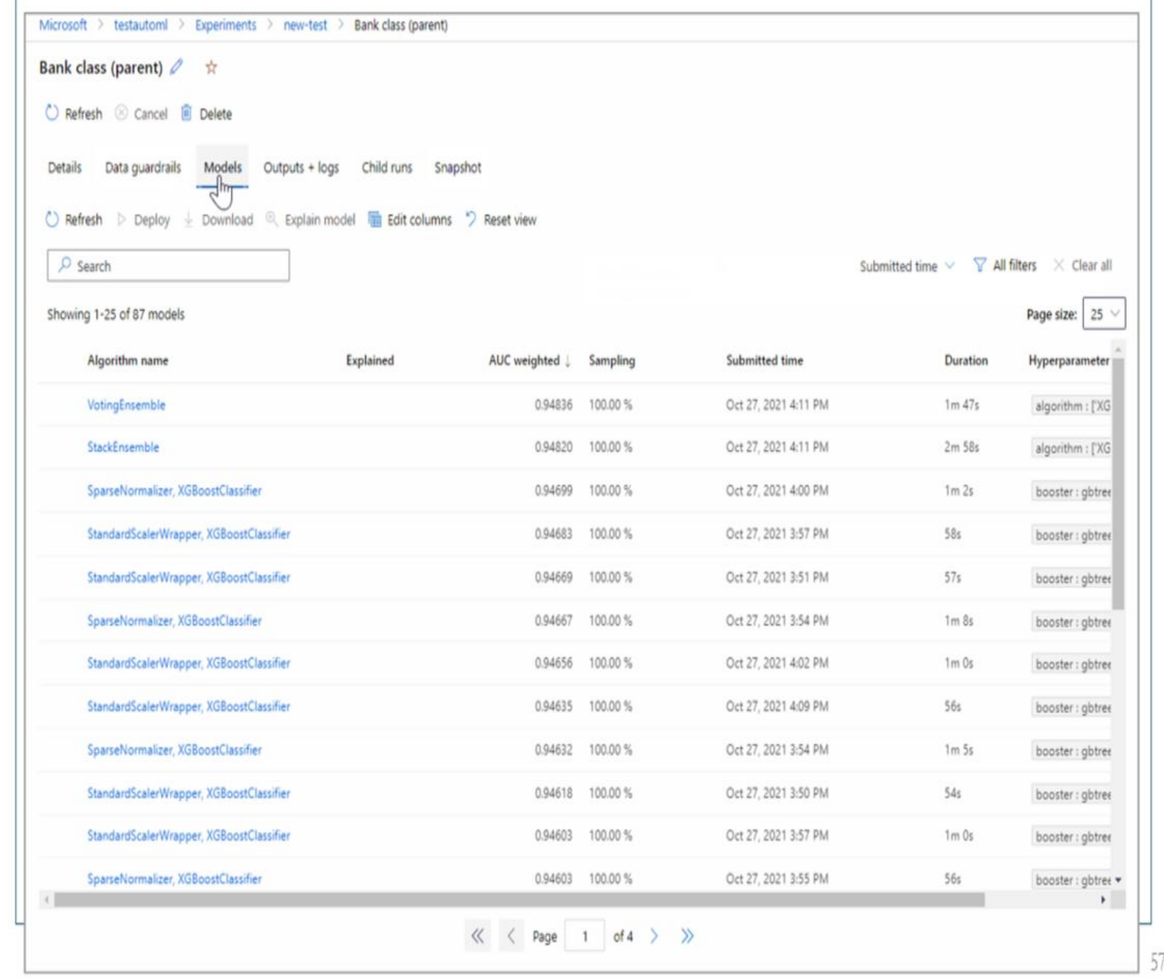
55

The job is currently running



6

6. Explore the models



57

Thank You

58

Lecture 8
Data Warehouse

Dr. Lydia Wahid

Agenda

- Defining Data warehouse
- Defining Data Marts and Data Lakes
- Data warehouse architecture
- Data Modeling for Data Warehouses

Defining Data warehouse

Defining Data warehouse

- A database is defined as a collection of related data. A **data warehouse** is also a **collection of information** as well as a supporting system.
- However, a clear distinction exists: Traditional databases are **transactional** and they support **online transaction processing (OLTP)** which includes insertions, updates, and deletions.
- Data warehouses have the distinguishing characteristic that they are mainly intended for **decision-support applications**. They are optimized for **data retrieval**, not routine transaction processing.

Defining Data warehouse

- A data warehouse also keeps **historical data.**
- To understand this term better, let us consider the database of a University Management System**:**
  - Once a student has left, then his or her database is usually **removed** from OLTP because OLTP is meant to perform day-to-day operations.
  - However, the management may be interested in retaining the data of old students. It can be used for later queries as well as for **analysis purposes**.
  - All such historical records can be moved to a separate data store known as the data warehouse.

Defining Data warehouse

- Historical data is not to be tampered with; no insertion, up-dation and deletion are to be made. Usually, it is used only for retrieval such as verification and data analysis.
- Thus, when data is shifted from OLTP to the data warehouse it is **de-normalized**, because normalization was earlier conducted to remove insert, update and delete anomalies but now only retrieval is important.
  - To **improve the performance of retrieval**, smaller tables are combined together to form larger tables.
- Data warehouses are typically used for **Online Analytical Processing (OLAP)** to support management queries.

Defining Data warehouse

- **OLAP (online analytical processing)** is a term used to describe the analysis of complex data from the data warehouse.

- Data warehouses are designed to support efficient extraction, processing, and presentation for analytic and decision making purposes such as:
  - OLAP
  - Decision-support systems (DSS) or Executive information systems (EIS)
  - Data mining

Defining Data warehouse

- **Data warehouse definition:**
  - According to Inmon, a data warehouse could be defined as: "A **subject-oriented**, **integrated**, **time-variant**, and **non-volatile** collection of data in support of management's decision-making process."
    - **Subject-Oriented:** they are built around the major data entity or subjects of an organization.
    - **Integrated:** integrates (combines) data from multiple systems
    - **Time variant:** data is not always up to date as it contains historical data which is valid or accurate till some point of time
    - **Non-volatile:** the previous data is not erased when new data is added

Defining Data warehouse

Traditional databases	Data warehouse
OLTP applications	OLAP applications
Hold current data	Hold current and historical data
Data is Dynamic	Data is static
Transaction-driven	Analysis-driven
Normalized data	De-normalized data
Retrieve – Insert – Update – Delete	Retrieve only
Application-oriented	Subject-oriented
Pattern of usage is predictable	Pattern of usage is unpredictable

Defining Data Marts and Data Lakes

Defining Data Marts and Data Lakes

- **Data Marts: Analytical** databases similar to data warehouses but with a defined **narrow scope**
  - It is a small localized data warehouse built for a **single purpose**.
  - It is usually built to cater to the needs of a group of users or a department in an organization.
  - A collection of data marts can constitute an enterprise-wide data warehouse.

Defining Data Marts and Data Lakes

- **Data Lakes:**
  - It is a massive storage pool for data in its natural, **raw state** (like a lake).
  - A data lake can handle huge volumes of data without the need to structure it first.
  - On the other hand, a data warehouse stores processed structured data.

Data warehouse architecture

Data warehouse architecture

- Every data warehouse has three fundamental components:
  - Load Manager
  - Warehouse Manager
  - Query manager

Data warehouse architecture

- **Load Manager:**
  - Responsible for **Data collection**
  - Performs data **conversion** into some usable form to be further utilized by the user.
  - Includes all the programs and application interfaces which are required for **extracting data**, it's **preparation** and finally **loading** of data into the data warehouse itself.

Data warehouse architecture

- Warehouse insertions are handled by the warehouse's **ETL (extract, transform, load)** process, which does a large amount of preprocessing.
  - **Extract:**
    - Raw data is copied or exported from source locations
    - Data management teams can extract data from a variety of data sources, which can be structured or unstructured
  - **Transform:**
    - A set of rules are applied over the data, in order to transform it from source format to target format
    - This phase can involve the following tasks: Filtering, cleansing, validating, authenticating the data, performing calculations, translations, or summarizations based on the raw data
  - **Load:**
    - Transformed data is moved to the target data warehouse

Data warehouse architecture

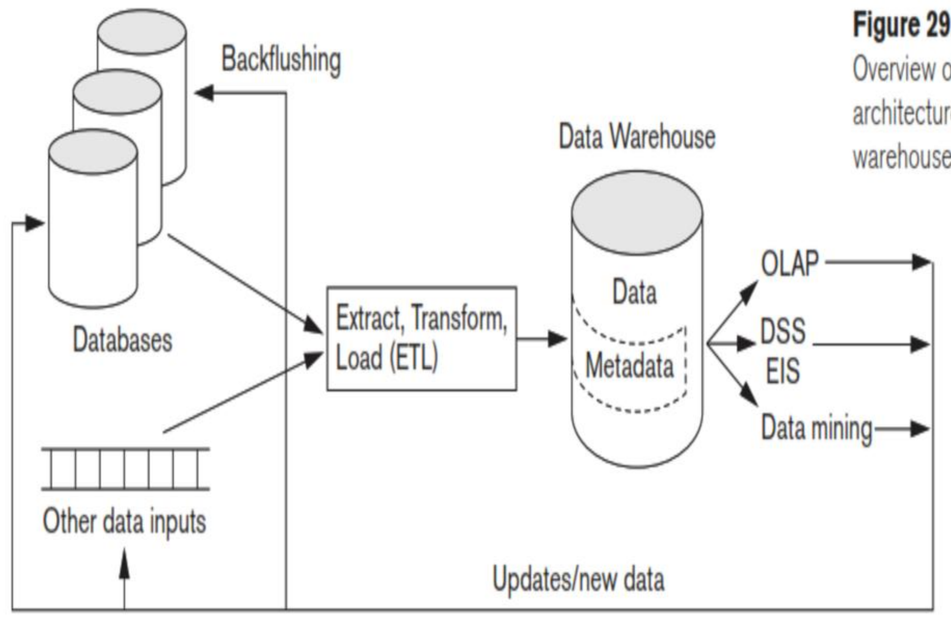


Figure 29.1
Overview of the general architecture of a data warehouse.

Data warehouse architecture

- **Warehouse Manager:**
  - It is the main part of Data Warehousing system as it **holds the massive amount of information.**
  - It **organizes data** to analyze it or find the required information.
  - It maintains three levels of information, i.e, detailed, lightly summarized and highly summarized.
  - It also maintains **metadata**, i.e., data about data.
- **Query Manager:**
  - Query manager is that interface which connects the end users with the information stored in data warehouse through the usage of specialized end-user tools.

Data Modeling for Data Warehouses

Data Modeling for Data Warehouses

- Multidimensional models populate data in multidimensional matrices called **data cubes**. (These may be called **hypercubes** if they have more than three dimensions)
- Query performance in multidimensional matrices can be much better than in the relational data model.
- More than three dimensions **cannot be easily visualized**; however, the data can be **queried directly** in any combination of dimensions, thus bypassing complex database queries.

Data Modeling for Data Warehouses

- The figure shows three-dimensional data cube that organizes product sales data by fiscal quarters and sales regions.
- Each cell contains data for a specific product, specific fiscal quarter, and specific region.

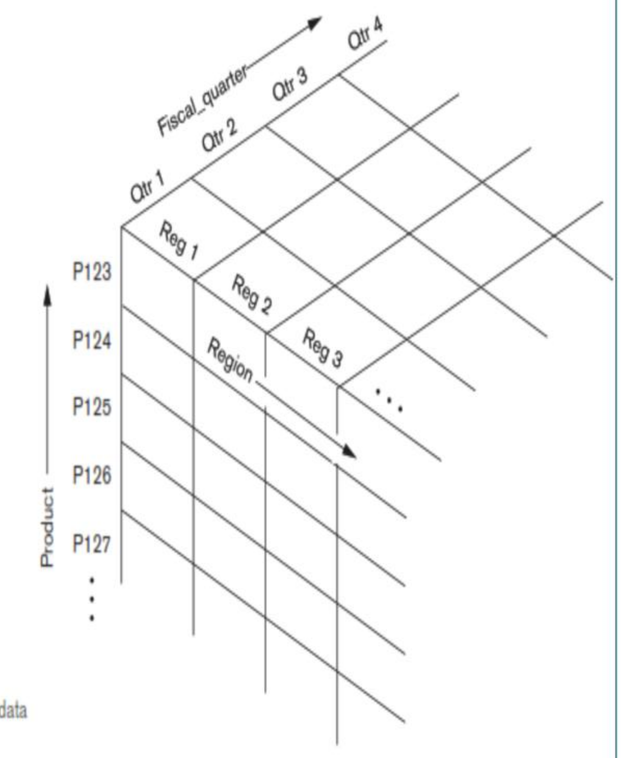


Figure 29.3
A three-dimensional data cube model.

Data Modeling for Data Warehouses

- The term **slice** is used to refer to a two-dimensional view of a three- or higher-dimensional cube.
- The term **"slice and dice"** implies a systematic reduction of a body of data into smaller chunks or views so that the information is made visible from multiple angles or viewpoints.
- Multidimensional models lend themselves readily to hierarchical views in what is known as roll-up display and drill-down display.
- A **roll-up display** moves up the hierarchy, grouping into larger units along a dimension (for example, summing weekly data by quarter)
- A **drill-down display** provides the opposite capability, for example, disaggregating country sales by region and then regional sales by subregion.
  - Typically, in a warehouse, the drill-down capability is limited to the lowest level of aggregated data stored in the warehouse.

Data Modeling for Data Warehouses

- The multidimensional model involves two types of tables: **dimension tables** and **fact tables**.
- A **dimension table** consists of the attributes of the dimension.
- A **fact table** can be thought of as having tuples, one per a recorded fact.
- The fact table contains the data, and the dimensions identify each tuple in that data.

Data Modeling for Data Warehouses

- **Logical descriptions** of database are known as Schema. It is the blueprint of the entire database.
- It defines **how the data are organized** and how the **relations among them** are associated.
- A database uses relational models whereas a data warehouse uses different types of schema, namely, **Star**, **Snowflake**, and **Fact Constellation.**

Data Modeling for Data Warehouses

- The **star schema** consists of a fact table with a single table for each dimension. The fact table is at the center and the dimension tables at the nodes of the star.
- Generally, fact tables are in third normal form (3NF) in the case of star schema while dimensional tables are in **de-normalized** form.

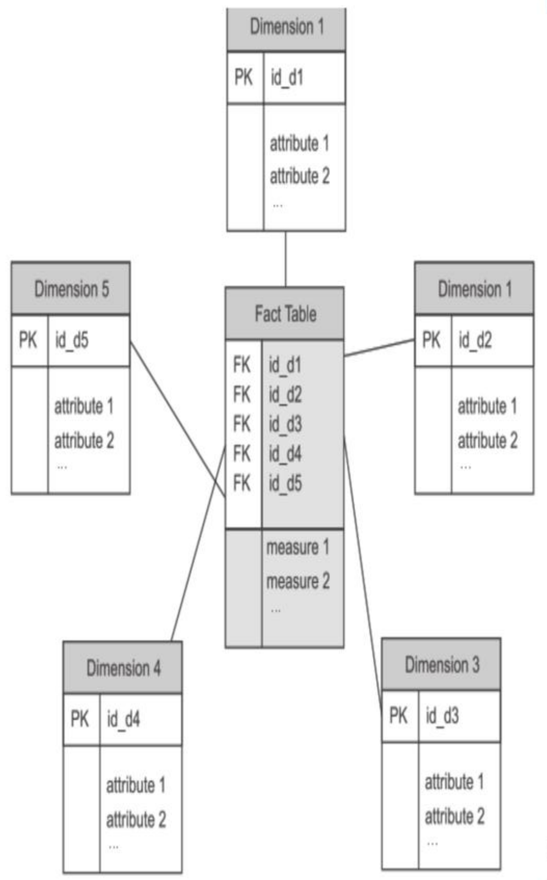


Figure 13.5 Graphical representation of Star schema

Data Modeling for Data Warehouses

- An example of **star schema:**

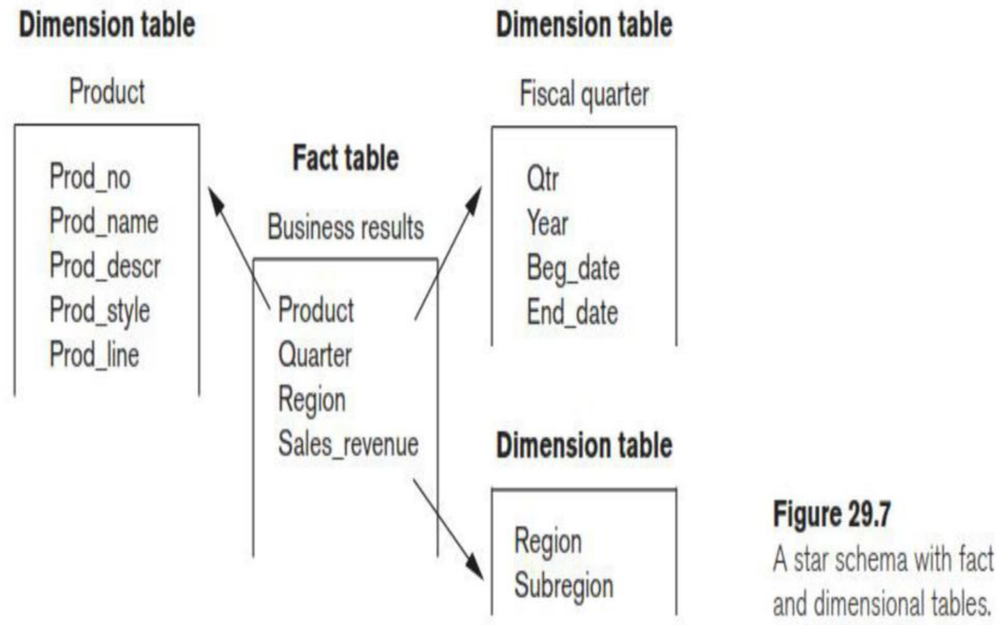
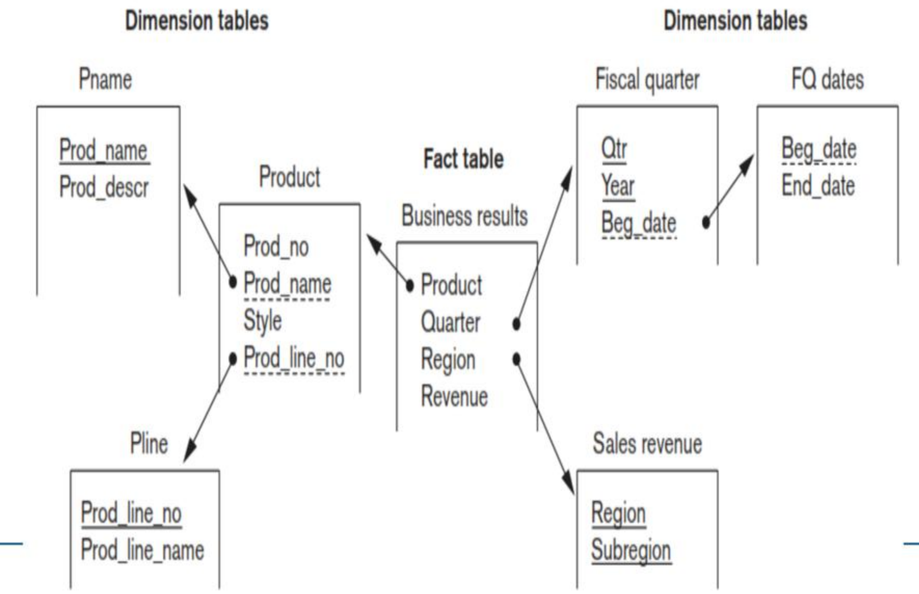


Figure 29.7
A star schema with fact and dimensional tables.

Data Modeling for Data Warehouses

- The **snowflake schema** is a variation on the star schema in which the dimensional tables are **normalized**.

Figure 29.8
A snowflake schema.



Data Modeling for Data Warehouses

- A **fact constellation schema** consists of multiple fact tables. It is a set of fact tables that share some dimension tables.
- Fact constellations limit the possible queries for the warehouse.

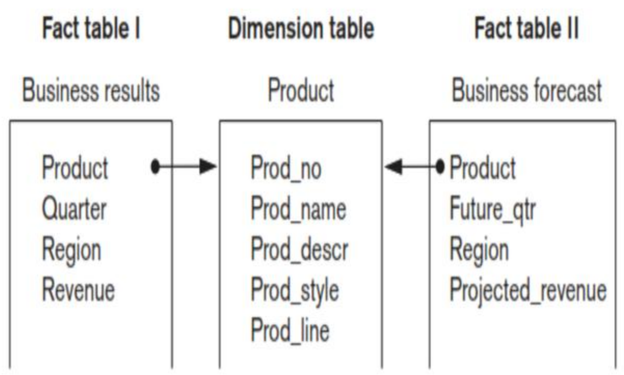
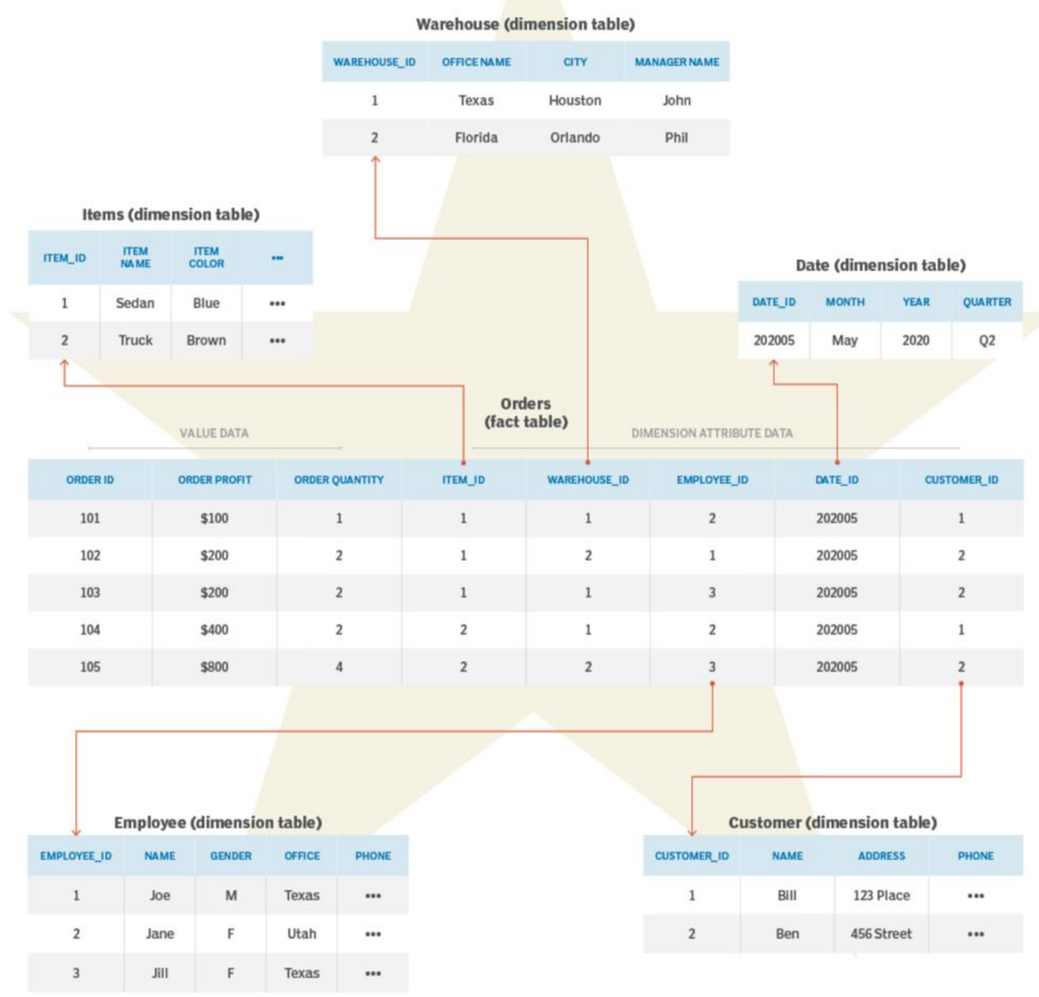


Figure 29.9
A fact constellation.

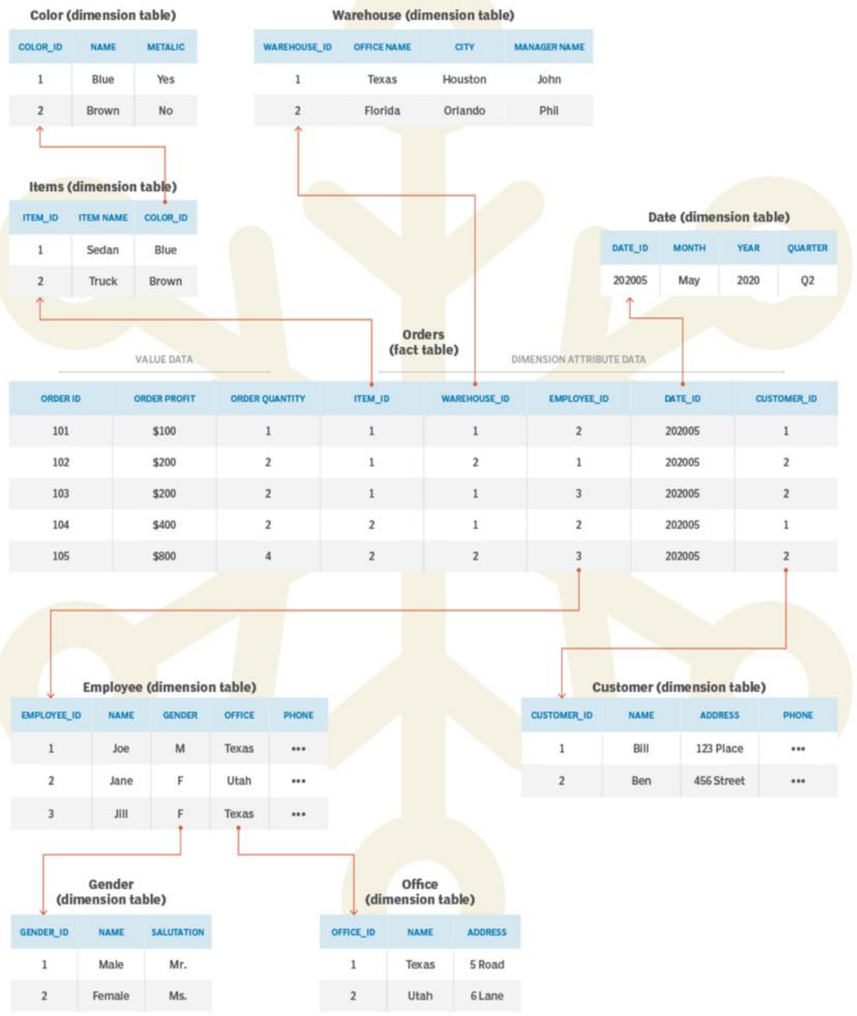
Data Modeling for Data Warehouses

- The fact table contains the specific measurable (or quantifiable) primary data to be analyzed, such as sales records, logged performance data or financial data.
- It may be **transactional** -- in that rows are added as events happen -- or it may be a snapshot of **historical data** up to a point in time.
- The fact table stores two types of information: **numeric values** and **dimension attribute values** (the foreign key value for a row in a related dimensional table)

Star schema



Snowflake schema



Data Modeling for Data Warehouses

Table 13.1 Comparison among Star, Snowflake and Fact Constellation Schema

Parameter	*Star*	*Snowflake*	*Fact constellation*
Query Joins	Require simple joins	Requires complicated joins	Requires complicated joins
Data structure	De-normalized data structure	Normalized data structure.	Normalized data structure.
Number of Fact Tables	Single fact table	Single fact table	Multiple fact tables
Query Performance	It gives faster query results due to fewer join operations.	It is slow in query processing due to larger join operations.	It is slow in query processing due to larger join operations.
Dimension	Dimension table does not split into pieces.	Dimension tables are split into many pieces.	Dimension tables are split into many pieces.
Data Redundancy	Data is redundant due to de-normalization.	Data is not redundant as dimensions are normalized.	Data is not redundant as dimensions are normalized.
Data Integrity	Tough to enforce data integrity due to redundancy of data.	Easy to enforce data integrity due to no redundancy of data.	Easy to enforce data integrity due to no redundancy of data.

Data Modeling for Data Warehouses

- Data warehouse storage also utilizes indexing techniques to support high performance access.
- A technique called **bitmap indexing** constructs a bit vector for each value in a domain (column) being indexed. It works very well for domains of low cardinality.
- Bitmap indexing can provide considerable input/output and storage space advantages in low-cardinality domains.
- With bit vectors, a bitmap index can provide dramatic improvements in comparison, aggregation, and join performance.

Data Modeling for Data Warehouses

- There is a 1 bit placed in the jth position in the vector if the jth row contains the value being indexed.
- For example, imagine an inventory of 100,000 cars with a bitmap index on car size. If there are four car sizes—economy, compact, mid-size, and full-size—there will be four bit vectors, each containing 100,000 bits (12.5kbytes) for a total index size of 50K.

References

- Bhatia, P. (2019). Chapter 12 Data Warehouse. In *Data Mining and Data Warehousing: Principles and Practical Techniques*. Cambridge: Cambridge University Press.
- Bhatia, P. (2019). Chapter 13 Data Warehouse Schema. In *Data Mining and Data Warehousing: Principles and Practical Techniques*. Cambridge: Cambridge University Press.
- Elmasri, R., & Navathe, S. (2016). Chapter 29 Overview of Data Warehousing and OLAP. In *Fundamentals of database systems 7th ed.,* Pearson Education.

Thank You

Lecture 9

Big Data Storage Concepts and Strategies

Dr. Lydia Wahid

Agenda

- Sharding
- Replication:
  - Master-Slave
  - Peer-to-Peer
- Sharding and Replication:
  - Sharding and master-slave replication
  - Sharding and peer-to-peer replication
- CAP theorem and BASE

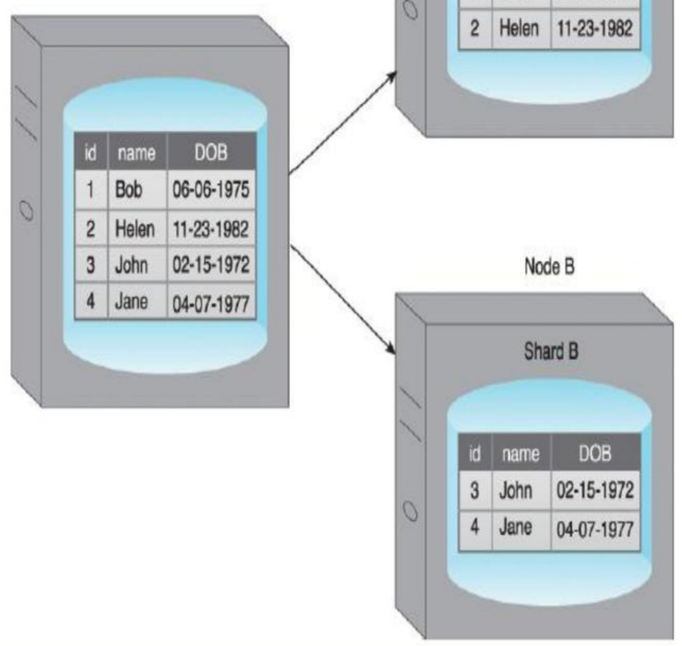
Sharding

Sharding

- Sharding is the process of **horizontally** partitioning a large dataset into a collection of smaller, more manageable datasets called **shards**.
- The shards are **distributed** across **multiple** nodes, where a node is a server or a machine.
- Each shard is stored on a separate node and each node is responsible for only the data stored on it.
- Each shard shares the **same schema**, and all shards collectively represent the complete dataset.

Sharding

- An example of sharding:

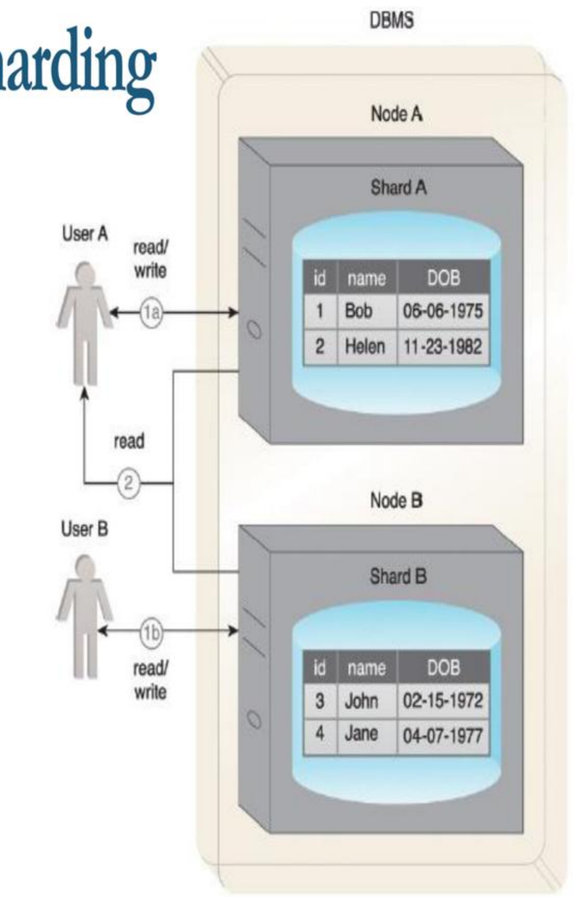


Sharding

- Sharding allows the distribution of processing loads across multiple nodes to achieve **horizontal scalability**.
- Horizontal scaling is a method for **increasing a system's capacity** by **adding similar or higher capacity** resources alongside existing resources.
- Since each node is responsible for only a part of the whole dataset, **read/write times are greatly improved**.

Sharding

- Each shard can **independently** service reads and writes for the specific subset of data that it is responsible for.
- Depending on the query, data may need to be fetched from **both shards**.



Sharding

- A benefit of sharding is that it provides **partial tolerance** toward failures.
- In case of a node failure, only data stored on that node is affected.
- With regards to **data partitioning**, **query patterns** need to be taken into account so that shards themselves do not become **performance bottlenecks**.
- For example, queries requiring data from multiple shards will impose **performance penalties**.
- **Data locality** keeps commonly accessed data co-located on a *single* shard and helps counter such performance issues.

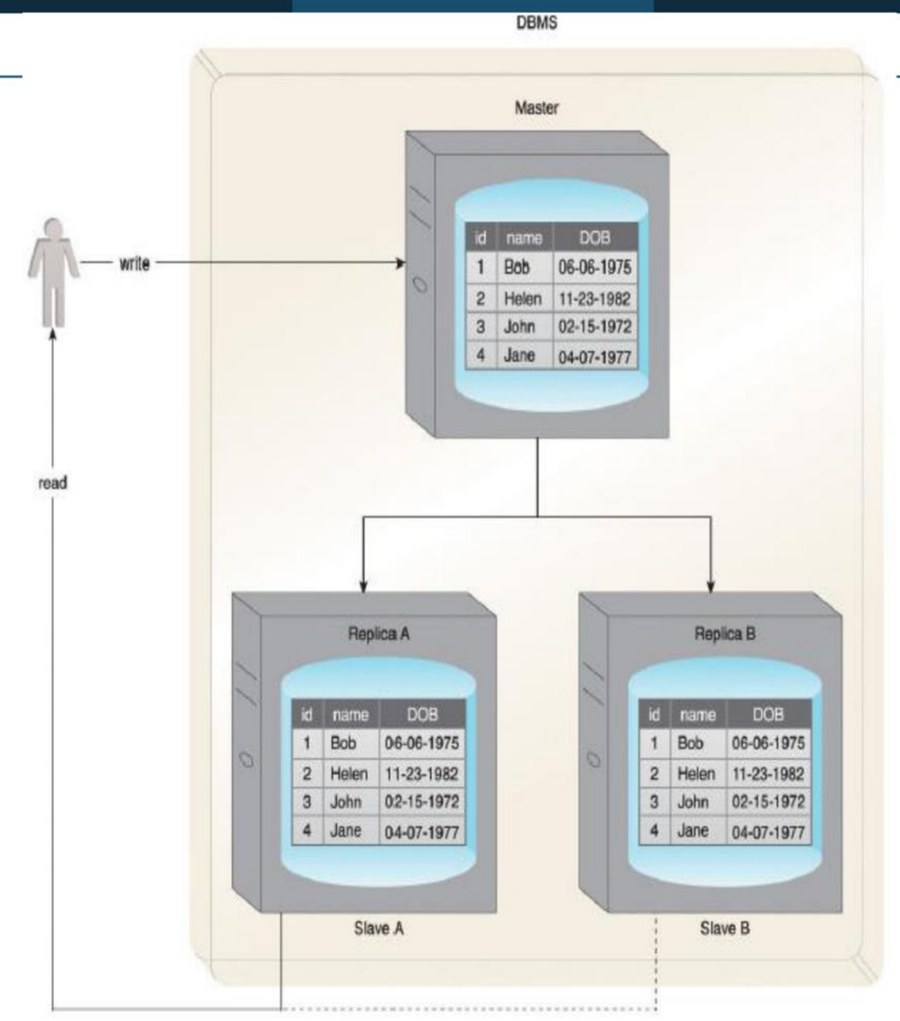
Replication

Replication

- Replication stores **multiple copies** of a dataset, known as **replicas**, on multiple nodes.
- Replication provides **scalability** and **availability** due to the fact that the same data is replicated on various nodes.
- **Fault tolerance** is also achieved since **data redundancy** ensures that data is not lost when an individual node fails.
- There are two different methods that are used to implement replication:
  - Master-Slave
  - Peer-to-Peer

Replication: Master-Slave

- During master-slave replication, nodes are arranged in a master-slave configuration, and **all data** is written to a **master node**.
- Once saved, the data is **replicated** over to **multiple slave nodes**.
- All external **write** requests, including insert, update and delete, occur on the **master node**, whereas **read** requests can be fulfilled by any **slave** node.



Master-Slave replication example

Replication: Master-Slave

- Master-slave replication is ideal for **read intensive loads** rather than write intensive loads since growing read demands can be managed by horizontal scaling to add more slave nodes.
- **Writes are consistent**, as all writes are coordinated by the master node.
- The implication is that **write performance** will *suffer* as the amount of **writes increases**.

Replication: Master-Slave

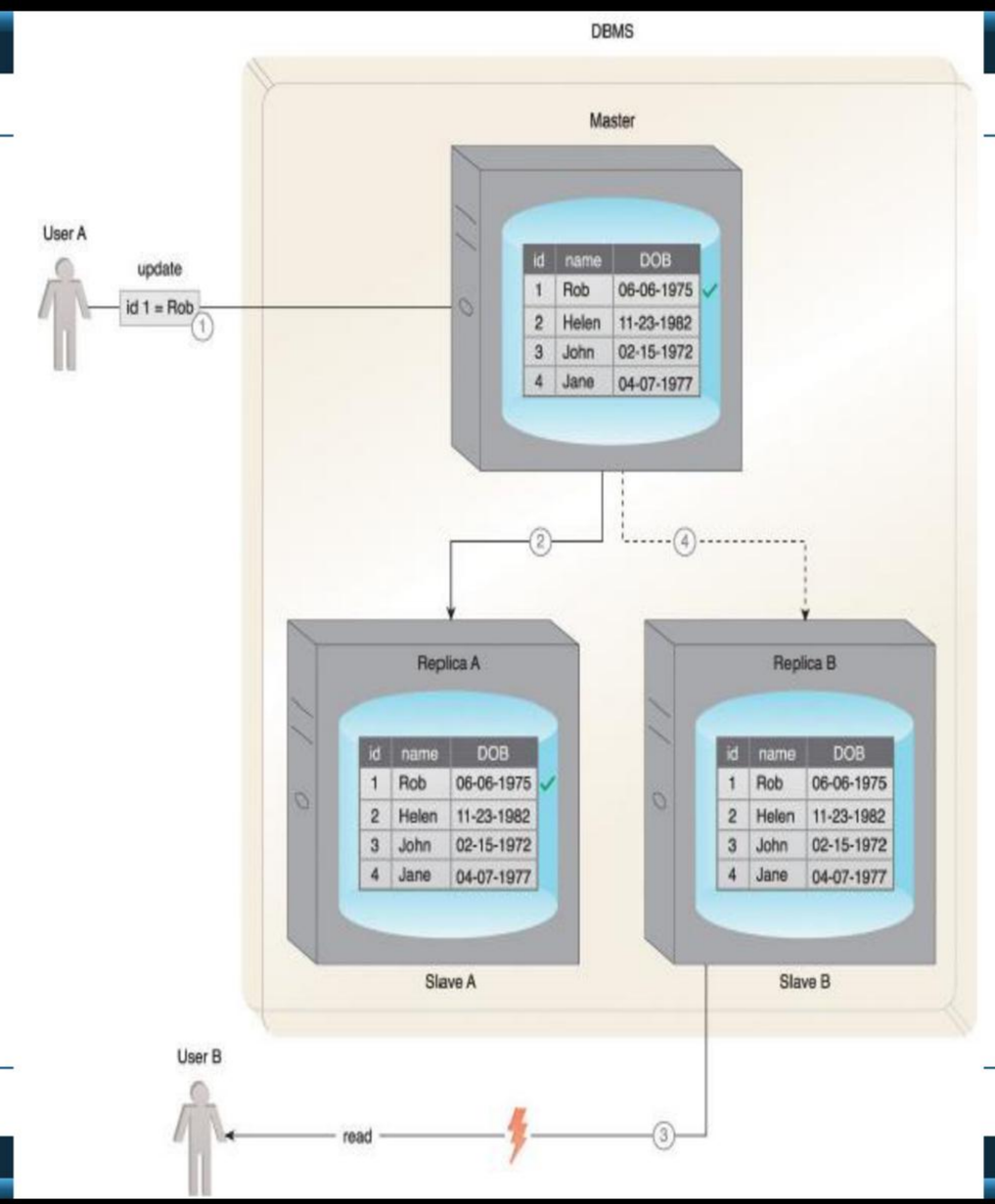
- If the **master node fails**, **reads are still possible** via any of the slave nodes.
- A slave node can be configured as a **backup node** for the master node.
- In the event that the master node fails, writes are **not supported** until a **master node is reestablished**.
- The master node is either **revived** from a backup of the master node, or a **new master node** is chosen from the slave nodes.

Replication: Master-Slave

- One concern with master-slave replication is **read inconsistency**, which can be an issue if a slave node is read prior to an update to the master being copied to it.
- To ensure read consistency, a **voting system** can be implemented where a read is declared consistent if the **majority** of the slaves contain the **same** version of the record.
- Implementation of such a voting system requires a *reliable* and *fast* communication mechanism between the slaves.

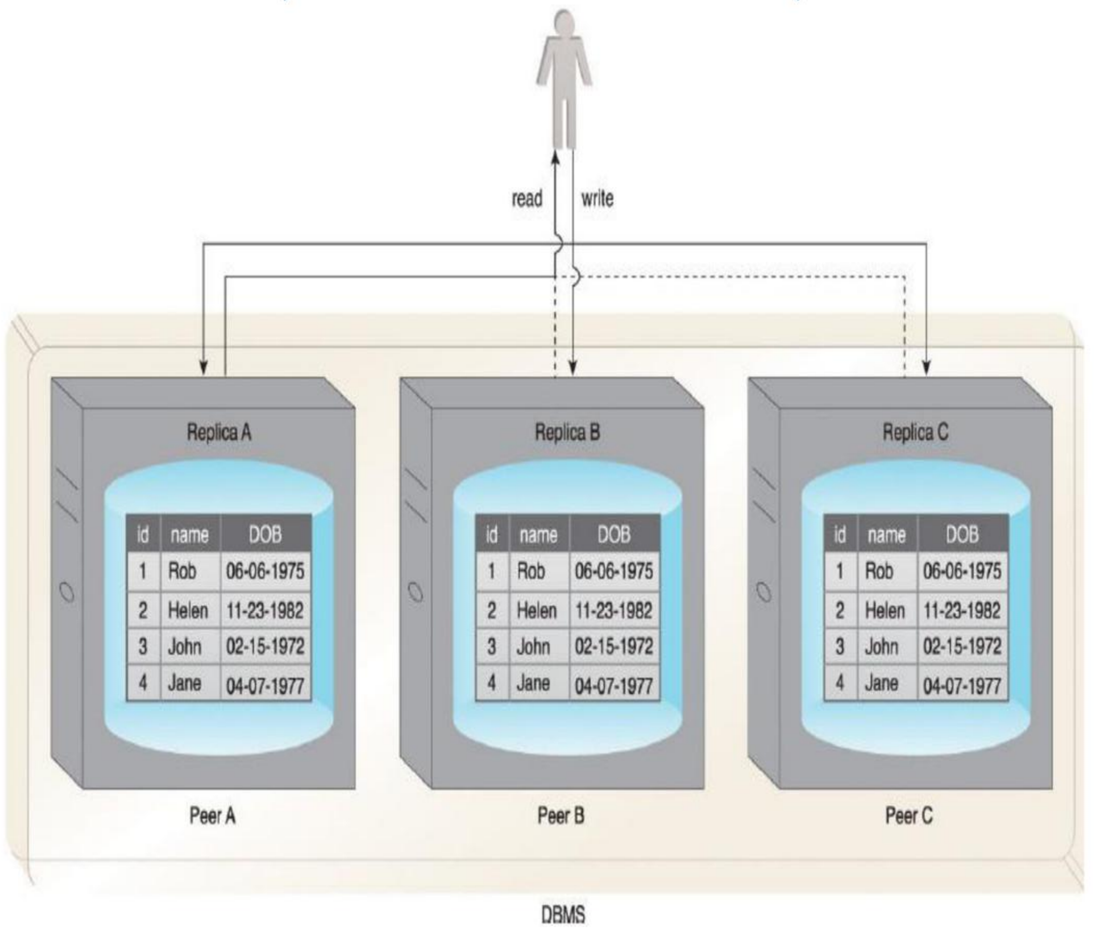
Replication: Master-Slave

- An example scenario of a read inconsistency is illustrated as follows:
  1. **User A updates** data.
  2. The data is **copied** over to **Slave A** by the Master.
  3. **Before** the data is **copied** over to **Slave B**, **User B** tries to **read** the data from **Slave B**, which results in an inconsistent **read**.
  4. The data will eventually become consistent when **Slave B** is **updated** by the Master.



Replication: Peer-to-Peer

- With peer-to-peer replication, **all nodes operate at the same level**.
- In other words, there is not a master-slave relationship between the nodes.
- Each node, known as a peer, is **equally capable** of handling **reads** and **writes**.
- Each write is **copied to all peers simultaneously.**



Replication: Peer-to-Peer

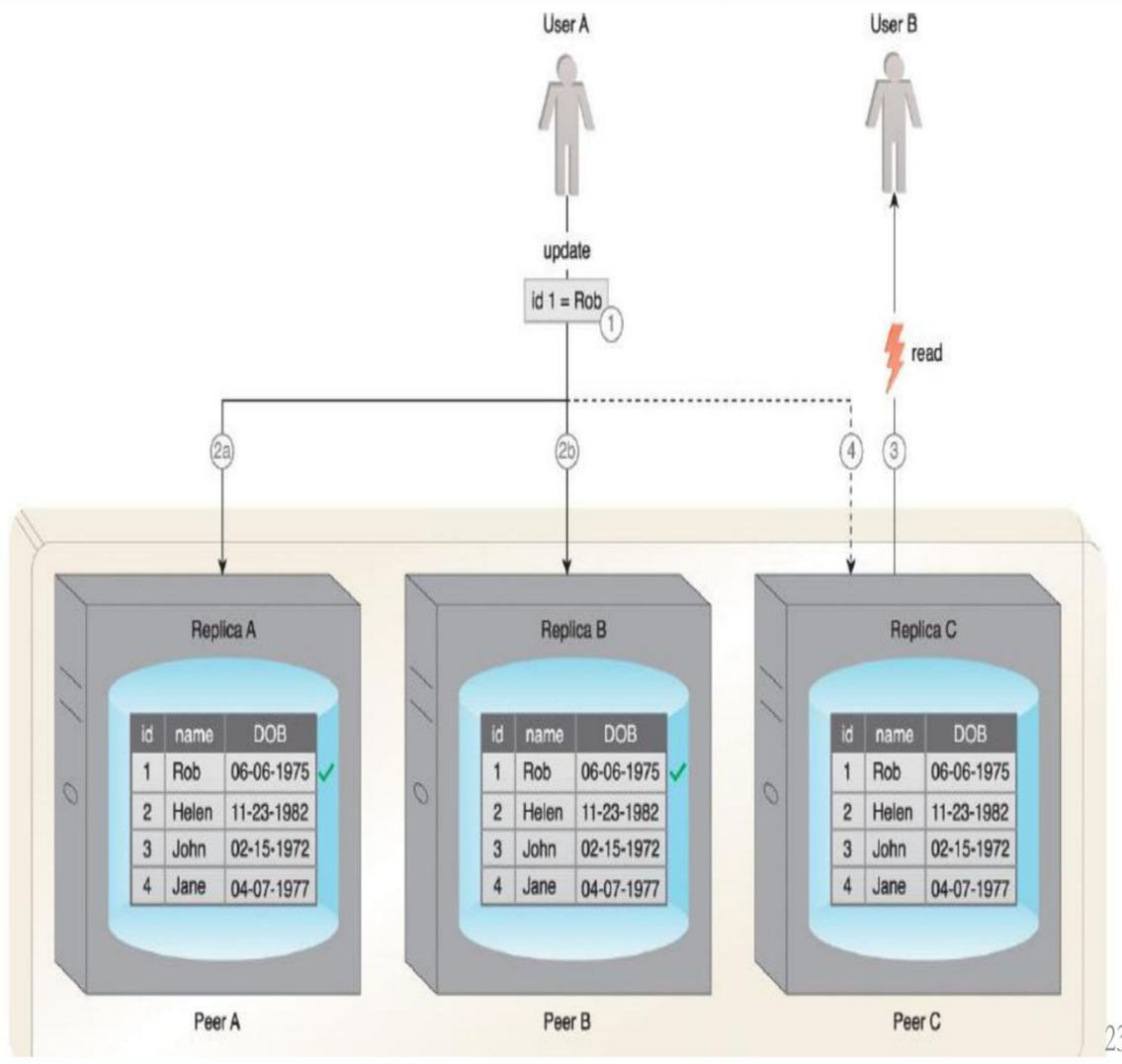
- Peer-to-peer replication is prone to **write inconsistencies** that occur as a result of a simultaneous update of the same data across multiple peers.
- This can be addressed by implementing either a pessimistic or optimistic concurrency strategy:
  - **Pessimistic concurrency** is a proactive strategy that prevents inconsistency. It uses *locking* to ensure that only one update to a record can occur at a time. However, this is unfavorable to availability since the database record being updated remains *unavailable* until all locks are released.
  - **Optimistic concurrency** is a reactive strategy that does not use locking. Instead, it allows inconsistency to occur with knowledge that eventually consistency will be achieved after all updates have propagated.

Replication: Peer-to-Peer

- With optimistic concurrency, peers may remain **inconsistent** for some period of time before attaining consistency.
- However, the database remains **available** as no locking is involved.
- Like master-slave replication, reads can be **inconsistent** during the time when some of the peers have completed their updates while others perform their updates.
- However, **reads eventually** become **consistent** when the updates have been executed on all peers.

Replication: Peer-to-Peer

- An example scenario where an inconsistent read occurs:
  1. **User A updates** data.
  2. a. The data is **copied** over to **Peer A**.
     b. The data is **copied** over to **Peer B**.
  3. Before the data is **copied** over to **Peer C**, **User B** tries to **read** the data from **Peer C**, resulting in an inconsistent read.
  4. The data will eventually be updated on **Peer C**, and the database will once again become consistent.



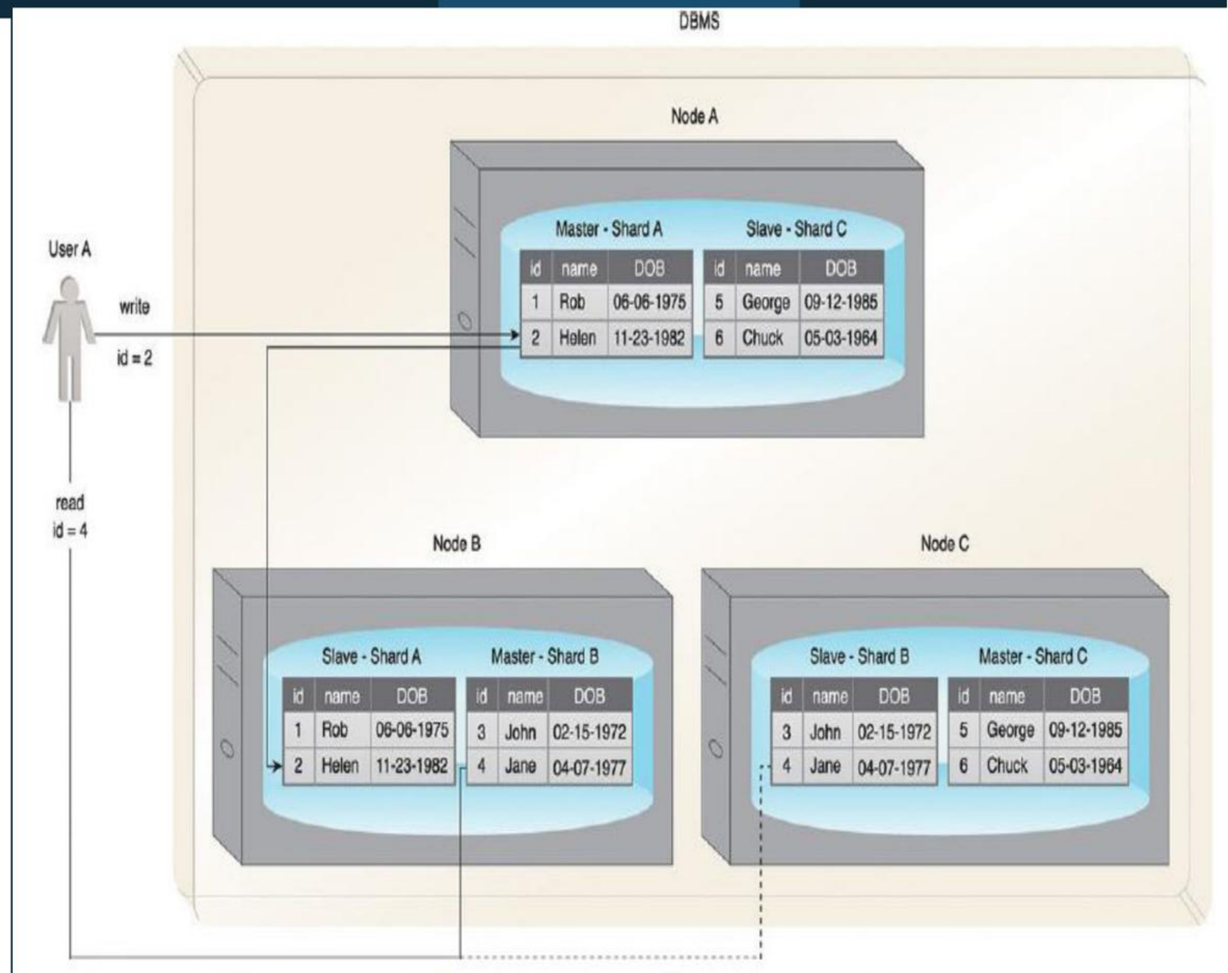
Sharding and Replication

Sharding and Replication

- To improve on the limited **fault tolerance** offered by sharding, while additionally benefiting from the increased **scalability** and **availability** of replication, both sharding and replication can be combined.

Sharding and master-slave replication

- When sharding is combined with master-slave replication, **multiple shards** become **slaves** of a single **master**, and the **master** itself is a **shard**.
- Although this results in **multiple masters**, a single slave-shard can only be managed by a single master-shard.
- **Write consistency** is maintained by the **master-shard**.
- However, if the master-shard becomes **non-operational** or a network outage occurs, fault tolerance with regards to write operations is impacted.
- Replicas of shards are kept on multiple slave nodes to provide fault tolerance for read operations.



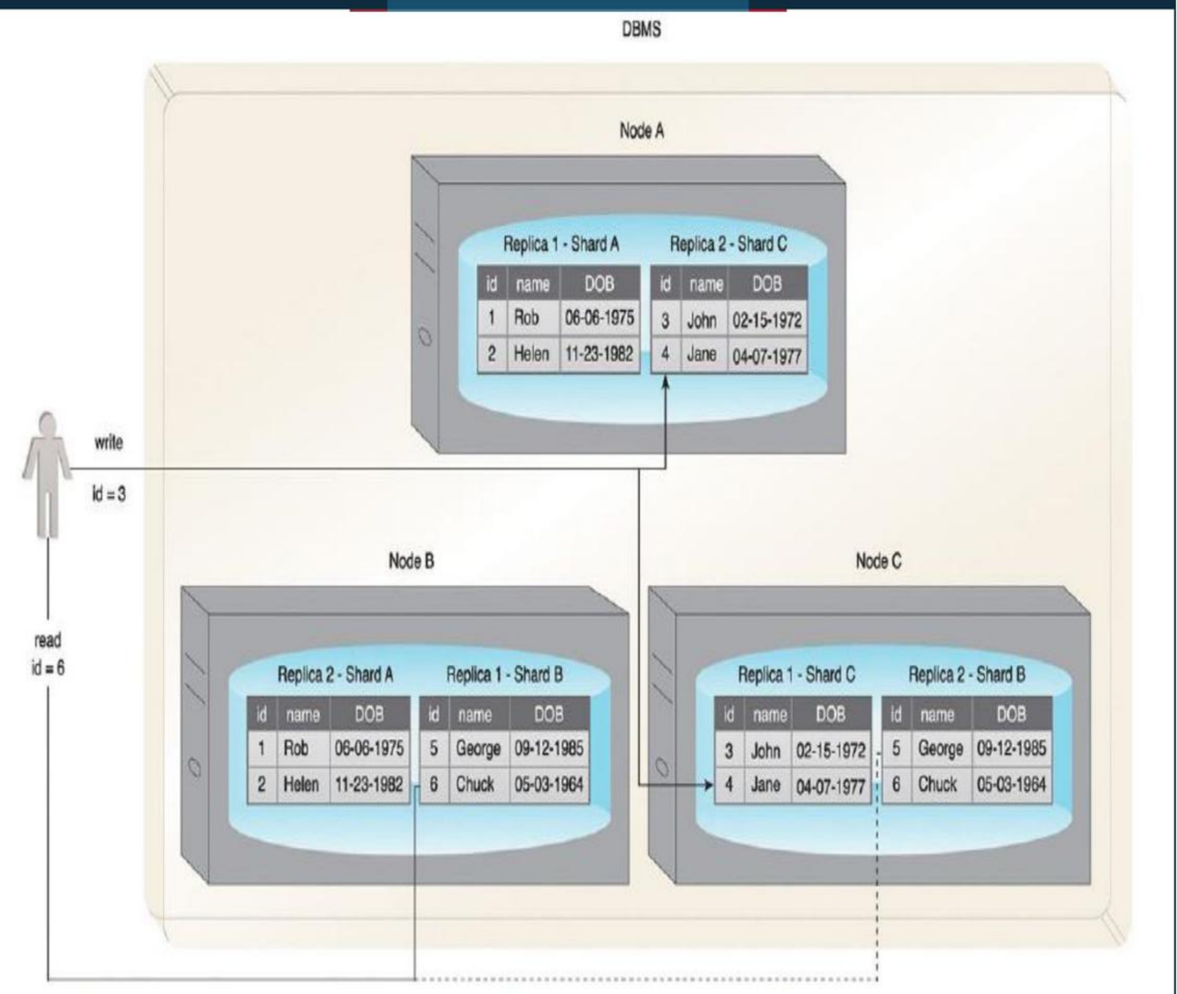
An example that shows the combination of sharding and master-slave replication.

Sharding and master-slave replication

- The previous figure illustrates the following:
  - Each node acts both as a **master** and a **slave** for different shards.
  - Writes (id = 2) to **Shard A** are regulated by **Node A**, as it is the **master for Shard A**.
  - **Node A** replicates data (id = 2) to **Node B**, which is a slave for **Shard A**.
  - Reads (id = 4) can be served directly by either **Node B** or **Node C** as they each contain **Shard B**.

Sharding and peer-to-peer replication

- When combining sharding with peer-to-peer replication, **each shard is replicated** to multiple peers, and each peer is only responsible for a subset of the overall dataset.
- Collectively, this helps achieve increased **scalability** and **fault tolerance**. As there is no master involved, there is no single point of failure and fault-tolerance for both read and write operations is supported.



An example of the combination of sharding and peer-to-peer replication.

Sharding and peer-to-peer replication

- The previous figure illustrates the following:
  - Each node contains replicas of two different shards.
  - Writes (id = 3) are replicated to both **Node A** and **Node C** (Peers) as they are responsible for **Shard C**.
  - Reads (id = 6) can be served by either **Node B** or **Node C** as they each contain **Shard B**.

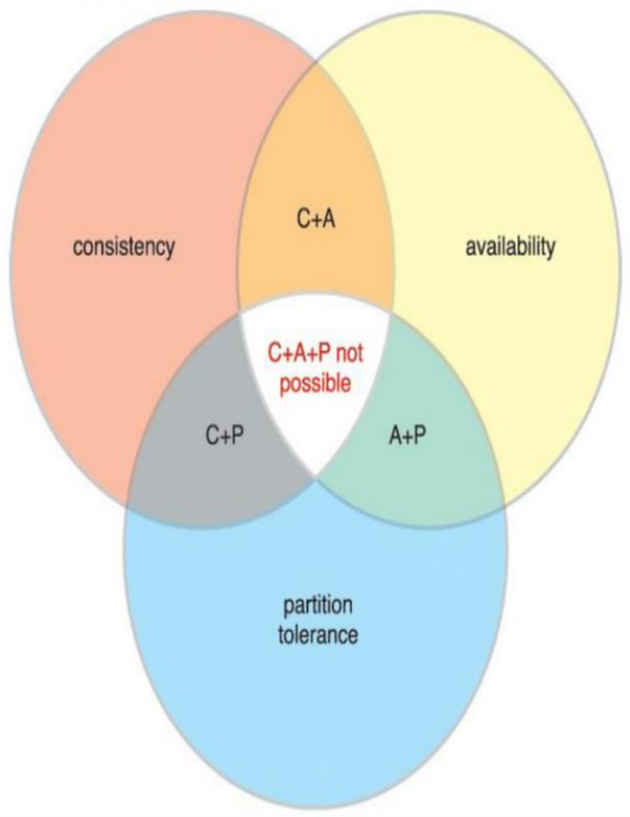
CAP theorem and BASE

CAP theorem

- The Consistency, Availability, and Partition tolerance (CAP) theorem, also known as Brewer's theorem, expresses a triple constraint related to distributed database systems.
- It states that a distributed database system, running on a cluster, **can only provide two** of the following three properties:
  - **Consistency** – A read from any node results in the same data across multiple nodes.
  - **Availability** – A read/write request will always be acknowledged in the form of a success or a failure.
  - **Partition tolerance** – The cluster continues to function even if there is a "partition" (communication break) between two nodes.

CAP theorem

- A Venn diagram summarizing the CAP theorem:



CAP theorem

- The following scenarios demonstrate why only two of the three properties of the CAP theorem are simultaneously supportable:
  - If consistency (C) and availability (A) are required, available nodes need to communicate to ensure consistency (C). Therefore, partition tolerance (P) is not possible.
  - If consistency (C) and partition tolerance (P) are required, nodes cannot remain available (A) as the nodes will become unavailable while achieving a state of consistency (C).
  - If availability (A) and partition tolerance (P) are required, then consistency (C) is not possible because of the data communication requirement between the nodes. So, the database can remain available (A) but with inconsistent results.

CAP theorem

- In a distributed database, scalability and fault tolerance can be improved through additional nodes, although this challenges consistency (C).
- Distributed database systems cannot be 100% partition tolerant (P).
- Although communication outages are rare and temporary, partition tolerance (P) must always be supported by a distributed database; therefore, CAP is generally a choice between choosing either **C+P** or **A+P**. The requirements of the system will dictate which is chosen.

BASE

- BASE is a database design principle based on the CAP theorem and leveraged by database systems that use distributed technology. BASE stands for:
  - basically available
  - soft state
  - eventual consistency
- When a database supports BASE, it favors availability over consistency. In other words, the database is **A+P** from a CAP perspective.
- In essence, BASE leverages optimistic concurrency by relaxing the strong consistency constraints mandated by the ACID properties (recall ACID: **A**tomicity **C**onsistency **I**solation **D**urability).

BASE

- If a database is "**basically available**," that database will always acknowledge a client's request, either in the form of the requested data or a success/failure notification.
- **Soft state** means that a database may be in an **inconsistent** state when data is read; thus, the results may change if the same data is requested again.
  - This is because the data could be updated for consistency, even though no user has written to the database between the two reads.
  - This property is closely related to eventual consistency.

BASE

- **Eventual consistency** is the state in which reads by different clients, immediately following a write to the database, may not return consistent results.
  - The database only attains consistency once the changes have been propagated to all nodes.
  - While the database is in the process of attaining the state of eventual consistency, it will be in a soft state.

BASE

- BASE emphasizes availability over immediate consistency, in contrast to ACID, which ensures immediate consistency at the expense of availability due to *record locking*.
- This soft approach toward consistency allows BASE compliant databases to serve multiple clients **without any latency** in spite of serving **inconsistent** results.
- However, BASE-compliant databases are not useful for transactional systems where lack of consistency is a concern.

Thank You

Lecture 10
Big Data Storage Technologies

Dr. Lydia Wahid

Agenda

- Introduction
- On-Disk Storage:
  - Distributed File System
  - RDBMS Databases
  - NoSQL Databases
  - NewSQL Databases
- In-Memory Storage:
  - In-Memory Data Grids
  - In-Memory Database (Relational, NoSQL, NewSQL)

Introduction

Introduction

- Storage technology has continued to evolve over time.
- The need to **store Big Data** has radically altered the relational, database-centric view that has been embraced by Enterprise ICT.
- The bottom line is that relational technology is simply **not scalable** in a manner to support **Big Data volumes**.
- Also, businesses can find genuine value in processing **semi-structured** and **unstructured** data, which are generally **incompatible** with relational approaches.

Introduction

- Big Data has pushed the storage boundary to unified views of the available memory and **disk storage** of a cluster.
- If more storage is needed, horizontal scalability allows the expansion of the cluster through the addition of **more nodes**.
- Innovative approaches deliver **realtime** analytics via **in-memory storage**.
- Even **batch-based processing** is accelerated by the performance of Solid State Drives (SSDs), which have become less expensive.

On-Disk Storage

On-Disk Storage

- On-disk storage generally utilizes **low cost** hard-disk drives for long-term storage.
- On-disk storage can be implemented via a **distributed file system** or a **database**.

On-Disk Storage: Distributed File Systems

On-Disk Storage: Distributed File Systems

- Distributed file systems, like any file system, support **schema-less** data storage.
- In general, a distributed file system storage device provides **redundancy** and high **availability** by copying data to multiple locations via **replication**.
- A storage device that is implemented with a distributed file system provides *simple*, *fast access* data storage that is capable of storing large datasets that are non-relational in nature, such as **semi-structured** and **unstructured** data.

On-Disk Storage: Distributed File Systems

- Although based on straightforward file **locking** mechanisms for concurrency control, it provides **fast read/write** capability, which addresses the **velocity** characteristic of Big Data.
- Distributed file system is **not ideal** for datasets comprising a **large number of small files** as this creates excessive **disk-seek activity**, **slowing down** the overall data access.
- Due to these limitations, distributed file systems work best with **fewer but larger files** accessed in a **sequential manner**.

On-Disk Storage: Distributed File Systems

- Multiple smaller files are generally **combined** into a single file to enable optimum *storage* and *processing*.
- This allows the distributed file systems to have increased performance when data must be accessed in **streaming mode** with **no random** reads and writes.
- A distributed file system storage device is suitable when large datasets of **raw data** are to be **stored** or when **archiving** of datasets is required.
- It should be noted that distributed file systems **do not provide the ability to search** the contents of files

On-Disk Storage: RDBMS Databases

On-Disk Storage: RDBMS Databases

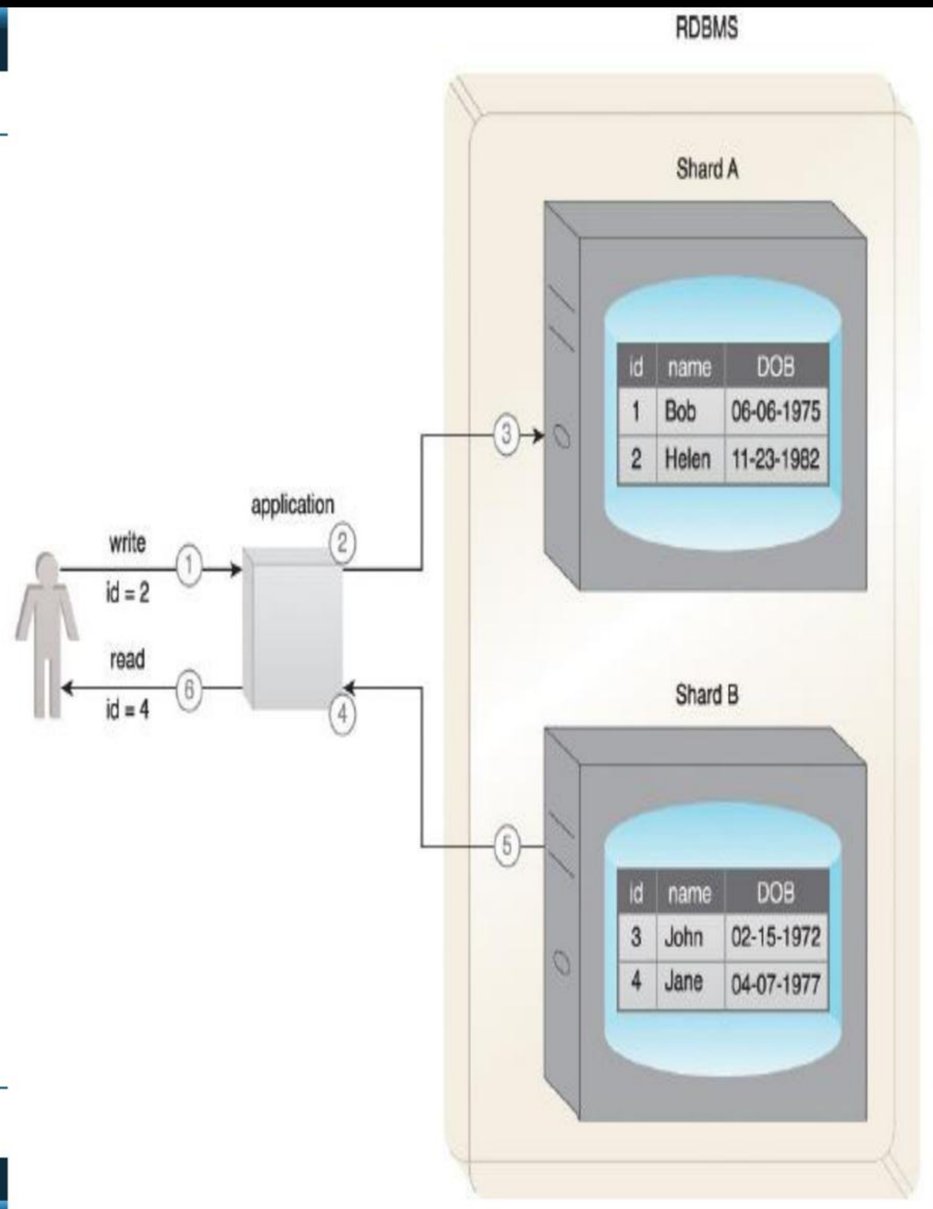
- Relational database management systems (RDBMSs) are good for handling transactional workloads involving **small amounts of data with random read/write properties**.
- RDBMSs are **ACID-compliant**, and they are generally restricted to a **single node**.
- To handle large volumes of data arriving at a fast pace, relational databases generally **need to scale**. RDBMSs employ vertical scaling, not horizontal scaling, which is a *more costly* and *disruptive* scaling strategy.

On-Disk Storage: RDBMS Databases

- Note that some relational databases, for example IBM DB2 pureScale, Sybase ASE Cluster Edition, Oracle Real Application Clusters (RAC) and Microsoft Parallel Data Warehouse (PDW), are capable of being run on clusters.
- However, these database clusters still use shared storage that can act as a single point of failure.
- Relational databases need to be **manually sharded**, mostly using application logic.
- This means that the application logic needs to know which shard to query in order to get the required data. This further **complicates** data processing when data from **multiple shards** is required.

On-Disk Storage: RDBMS Databases

- Consider the following scenario:
  1. A user **writes** a record (id = 2).
  2. The **application logic** determines **which shard** it should be written to.
  3. It is **sent to the shard** determined by the application logic.
  4. The user reads a record (id = 4), and the application logic determines which shard contains the data.
  5. The data is read and returned to the application.
  6. The application then returns the record to the user.



On-Disk Storage: RDBMS Databases

- Relational databases generally require data to adhere to a schema. As a result, storage of **semi-structured** and **unstructured** data whose schemas are non-relational is **not directly supported**.
- Furthermore, with a relational database schema conformance is validated **at the time of data insert or update** by checking the data against the constraints of the schema. This introduces overhead that creates **latency**.
- This latency makes relational databases a less than ideal choice for storing high velocity data that needs a highly available database storage device with *fast* data *write* capability.

On-Disk Storage: NoSQL Databases

NoSQL Databases: Characteristics

- **Not-only SQL** (NoSQL) refers to technologies used to develop next generation **non-relational** databases that are **highly scalable** and **fault-tolerant**.
- Below is a list of the principal characteristics of NoSQL storage devices that differentiate them from traditional RDBMSs:
  - **Schema-less data model** – Data can exist in its raw form.
  - **Highly available** – This is built on cluster-based technologies that provide fault.
  - **Eventual consistency** – Data reads across multiple nodes but may not be consistent immediately after a write. However, all nodes will eventually be in a consistent state.

NoSQL Databases: Characteristics

- Below is a list of the principal characteristics of NoSQL storage devices that differentiate them from traditional RDBMSs (cont.):
  - **Scale out rather than scale up** – More nodes can be added to obtain additional storage with a NoSQL database.
  - **BASE, not ACID** – BASE compliance requires a database to maintain high *availability* in the event of network/node failure, while not requiring the database to be in a *consistent* state whenever an update occurs. NoSQL storage devices are generally *AP* or *CP*.
  - **API driven data access** – Data access is generally supported via API based queries, including *RESTful APIs*, whereas some implementations may also provide *SQL-like query* capability.

NoSQL Databases: Characteristics

- Below is a list of the principal characteristics of NoSQL storage devices that differentiate them from traditional RDBMSs (cont.):
  - **Auto sharding and replication** – To support horizontal scaling and provide high availability, a NoSQL storage device automatically employs sharding and replication techniques.
  - **Polyglot persistence** – The use of NoSQL storage does not mandate retiring traditional RDBMSs. In fact, both can be used at the same time, thereby supporting polyglot persistence, which is an approach of *persisting data using different types of storage technologies within the same solution architecture*. This is good for developing systems requiring structured as well as semi/unstructured data.

NoSQL Databases: Characteristics

- Below is a list of the principal characteristics of NoSQL storage devices that differentiate them from traditional RDBMSs (cont.):
  - **Aggregate-focused** – Unlike relational databases that are most effective with fully normalized data, NoSQL storage devices store de-normalized aggregated data (an entity containing merged, often nested, data for an object) thereby eliminating the need for joins and extensive mapping between application objects and the data stored in the database. One exception, however, is that graph database storage devices (introduced shortly) are not aggregate-focused.

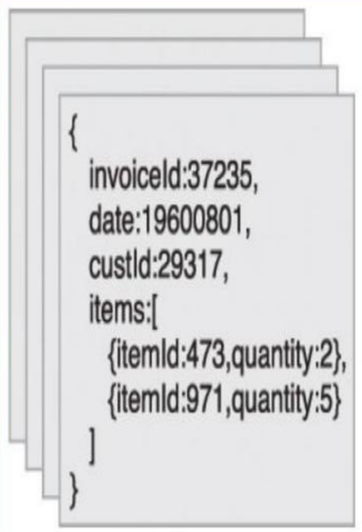
NoSQL Databases: Rational

- The emergence of NoSQL storage devices can primarily be attributed to the *volume*, *velocity* and *variety* characteristics of Big Data datasets:
  - **Volume:**
    - NoSQL storage devices provide scale out capability while using inexpensive commodity servers.
  - **Velocity:**
    - **Write latency does not occur** as no schema check on write.
    - It is highly available.
  - **Variety:**
    - NoSQL storage devices can store these different forms of **semi-structured** and **unstructured** data formats.
    - NoSQL databases support schema evolution.

NoSQL Databases: Types

- NoSQL storage devices can mainly be divided into four types based on the way they store data:
  - Key-value
  - Document
  - Column-family
  - Graph

NoSQL Databases: Types



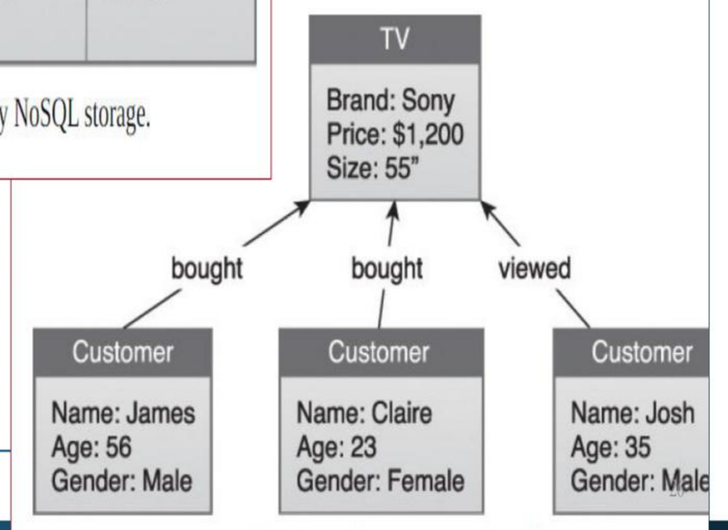
An example of document NoSQL storage.

key	value
631	John Smith, 10.0.30.25, Good customer service
365	1001010111011011110111010101101010010011100110 10
198	<CustomerId>32195</CustomerId><Total>43.25</Total>

An example of key-value NoSQL storage.

studentId	personal details	address	modules history
821	FirstName: Cristie LastName: Augustin DoB: 03-15-1992 Gender: Female Ethnicity: French	Street: 123 New Ave City: Portland State: Oregon ZipCode: 12345 Country: USA	Taken: 5 Passed: 4 Failed: 1
742	FirstName: Carlos LastName: Rodriguez MiddleName: Jose Gender: Male	Street: 456 Old Ave City: Los Angeles Country: USA	Taken: 7 Passed: 5 Failed: 2

An example of column-family NoSQL storage.



An example of graph NoSQL storage.

NoSQL Databases: Key-Value

- A key-value storage device is **appropriate** when:
  - unstructured data storage is required
  - high performance read/writes are required
  - the value is fully identifiable via the key alone
  - value is a standalone entity that is not dependent on other values
  - values have a comparatively simple structure or are binary
  - query patterns are simple, involving insert, select and delete operations only

NoSQL Databases: Key-Value

- A key-value storage device is **inappropriate** when:
  - applications require searching or filtering data using attributes of the stored value
  - relationships exist between different key-value entries
  - a group of keys' values need to be updated in a single transaction
  - multiple keys require manipulation in a single operation
  - schema consistency across different values is required
  - update to individual attributes of the value is required

NoSQL Databases: Document

- The main differences between document storage devices and key-value storage devices are as follows:
  - document storage devices are value-aware
  - the stored value is self-describing; the schema can be inferred from the structure of the value or a reference to the schema for the document is included in the value
  - a select operation can reference a field inside the aggregate value
  - a select operation can retrieve a part of the aggregate value
  - partial updates are supported; therefore a subset of the aggregate can be updated

NoSQL Databases: Document

- A document storage device is **appropriate** when:
  - storing semi-structured document-oriented data comprising
  - schema evolution is a requirement as the structure of the document is either unknown or is likely to change
  - applications require a partial update of the aggregate stored as a document
  - searches need to be performed on different fields of the documents
  - query patterns involve insert, select, update and delete operations
- A document storage device is **inappropriate** when:
  - multiple documents need to be updated as part of a single transaction
  - performing operations that need joins between multiple documents
  - schema enforcement for achieving consistent query design is required

NoSQL Databases: Column-Family

- A column-family storage device is **appropriate** when:
  - data represents a tabular structure
  - support for schema evolution is required
  - certain fields are mostly accessed together, and searches need to be performed using field values
  - query patterns involve insert, select, update and delete operations
- A column-family storage device is **inappropriate** when:
  - relational data access is required; for example, joins
  - ACID transactional support is required
  - SQL-compliant queries need to be executed
  - query patterns are likely to change frequently because that could initiate a corresponding restructuring of how column-families are arranged

NoSQL Databases: Graph

- A graph storage device is **appropriate** when:
  - Consistency is required (generally, graph storage devices provide consistency via ACID compliance)
  - interconnected entities need to be stored
  - querying entities based on the type of relationship with each other rather than the attributes of the entities
  - finding groups of interconnected entities
  - finding distances between entities in terms of the node traversal distance

NoSQL Databases: Graph

- A graph storage device is **inappropriate** when:
  - updates are required to a large number of node attributes or edge attributes, as this involves searching for nodes or edges, which is a costly operation
  - entities have a large number of attributes or nested data—it is best to store lightweight entities in a graph storage device while storing the rest of the attribute data in a separate non-graph NoSQL storage device
  - queries based on the selection of node/edge attributes dominate node traversal queries

On-Disk Storage: NewSQL Databases

NewSQL Databases

- NoSQL storage devices are **highly scalable**, **available**, **fault-tolerant** and **fast** for read/write operations.
- However, they do not provide the same transaction and consistency support as exhibited by **ACID** compliant RDBMSs. Following the BASE model, NoSQL storage devices provide **eventual consistency** rather than **immediate consistency**.
- They therefore will be in a soft state while reaching the state of eventual consistency.
- As a result, they are **not appropriate** for use when implementing large scale transactional systems.

NewSQL Databases

- NewSQL is a class of relational databases that combines the ACID properties of **RDBMS** with the scalability and fault tolerance offered by **NoSQL** storage devices.
- NewSQL databases generally support **SQL** compliant syntax for data definition and data manipulation operations, and they often use a logical relational data model for data storage.

NewSQL Databases

- NewSQL databases can be used for developing **OLTP** (online transaction processing) systems with very high volumes of transactions, for example a banking system.
- They can also be used for **real-time analytics** as some implementations leverage **in-memory storage**.
- Examples of NewSQL databases include Apache Trafodion, Google spanner, VoltDB and NuoDB.

In-Memory Storage

In-Memory Storage

- This section presents in-memory storage as a means of providing options for **highly performant**, and **advanced data storage**.
- An in-memory storage device generally utilizes **RAM**, the main memory of a computer, as its storage medium to provide **fast data access**.
- The growing capacity and decreasing cost of RAM, coupled with the increasing read/write speed of solid state hard drives, has made it possible to develop in-memory data storage solutions.

In-Memory Storage

- Storage of data in memory **eliminates the latency of disk I/O** and the data transfer time between the main memory and the hard drive.
- This **overall reduction** in data read/write **latency** makes data processing **much faster**.
- In-memory storage device capacity can be increased massively by **horizontally scaling the cluster.**
- In-memory storage significantly **reduces the overall execution time** of Big Data analytics, thus **enabling real-time Big Data analytics**.

In-Memory Storage

- When compared with an on-disk storage device, an in-memory storage device is **expensive** because of the higher cost of memory as compared to a disk-based storage device.
- Apart from being expensive, in-memory storage devices do not provide the same level of support for **durable data storage**. The **price** factor further affects the achievable **capacity** of an in-memory device when compared with an on-disk storage device.

In-Memory Storage

- An in-memory storage device is **appropriate** when:
  - data arrives at a **fast pace** and **requires real-time analytics** or **event stream processing**
  - **continuous** or **always-on analytics** is required, such as operational BI and operational analytics
  - **interactive query processing** and real-time data visualization needs to be performed
  - developing **low latency** Big Data solutions

In-Memory Storage

- An in-memory storage device is **inappropriate** when::
  - data processing consists of **batch processing**
  - **very large amounts of data** need to be persisted in-memory for a long time in order to perform in-depth data analysis
  - datasets are **extremely large and do not fit** into the available memory
  - an enterprise has a **limited budget**, as setting up an in-memory storage device may require upgrading nodes, which could either be done by node replacement or by adding more RAM

In-Memory Storage

- In-memory storage devices can be implemented as:
  - In-Memory Data Grid (IMDG)
  - In-Memory Database (IMDB)

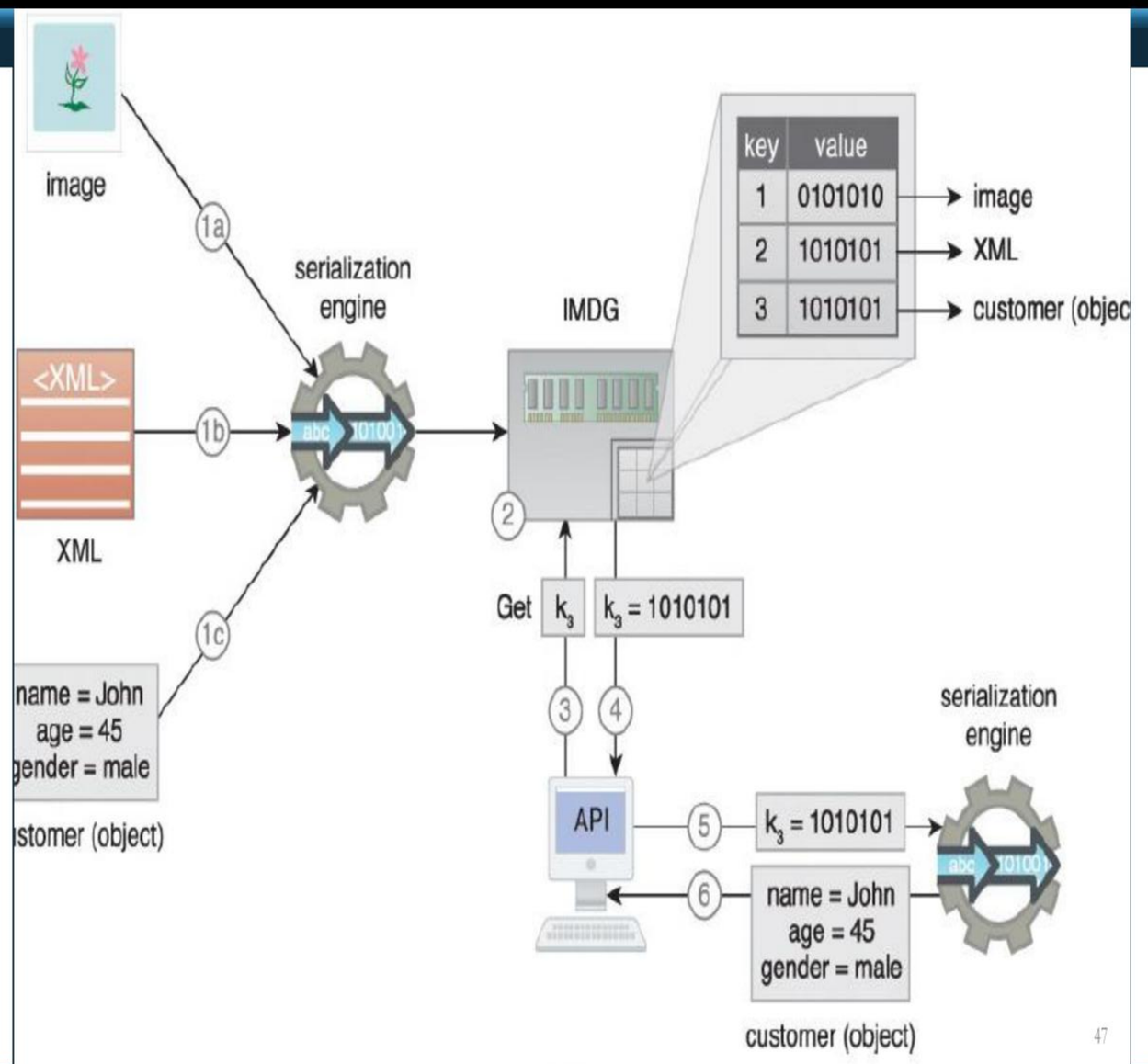
- Although both of these technologies use memory as their underlying data storage medium, what makes them distinct is **the way data is stored in the memory**.

In-Memory Storage: In-Memory Data Grids

In-Memory Storage: In-Memory Data Grid (IMDG)

- IMDGs store data in memory as **key-value pairs** across multiple nodes where the keys and values can be any business object or application data in serialized form.

- This supports schema-less data storage through storage of **semi/unstructured** data. Data access is typically provided via APIs.



An IMDG storage device.

In-Memory Storage: In-Memory Data Grid (IMDG)

- The previous figure describes the following scenario:
  1. An image (a), XML data (b) and a customer object (c) are first serialized using a serialization engine.
  2. They are then stored as key-value pairs in an IMDG.
  3. A client requests the customer object via its key.
  4. The value is then returned by the IMDG in serialized form.
  5. The client then utilizes a serialization engine to deserialize the value.
  6. The original object is returned for manipulation.

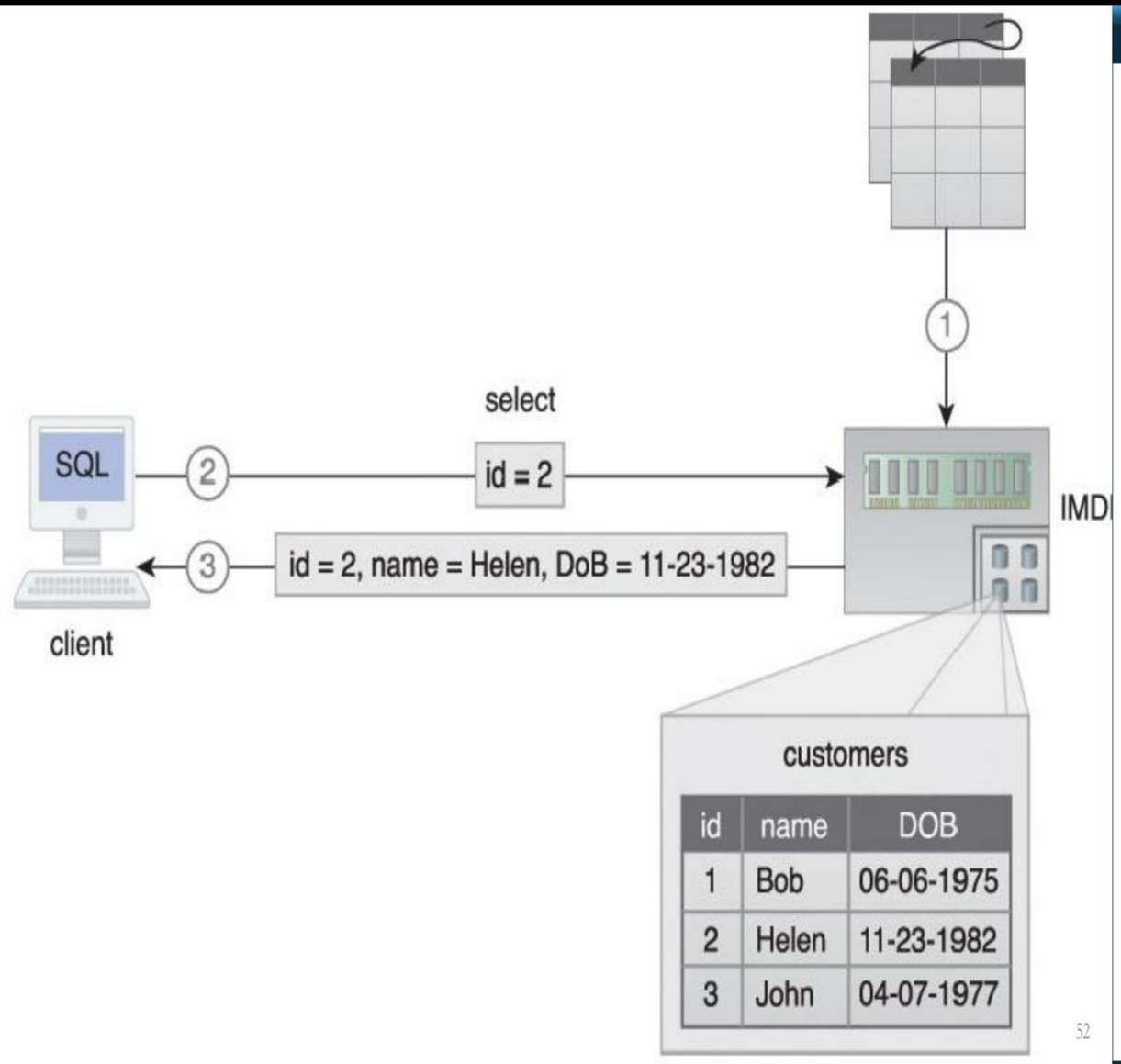
In-Memory Storage: In-Memory Data Grid (IMDG)

- Nodes in IMDGs keep themselves synchronized and collectively provide **high availability**, **fault tolerance** and **consistency**.
- In comparison to **NoSQL**'s eventual consistency approach, IMDGs support **immediate consistency**.
- As compared to relational **IMDBs** (discussed next), IMDGs provide **faster** data access as IMDGs store non-relational data as objects.
- IMDGs may also support **in-memory MapReduce** that helps to reduce the latency of disk based MapReduce processing.
- Examples include In-Memory Data Fabric, Hazelcast and Oracle Coherence.

In-Memory Storage: In-Memory Database

In-Memory Storage: In-Memory Database (IMDB)

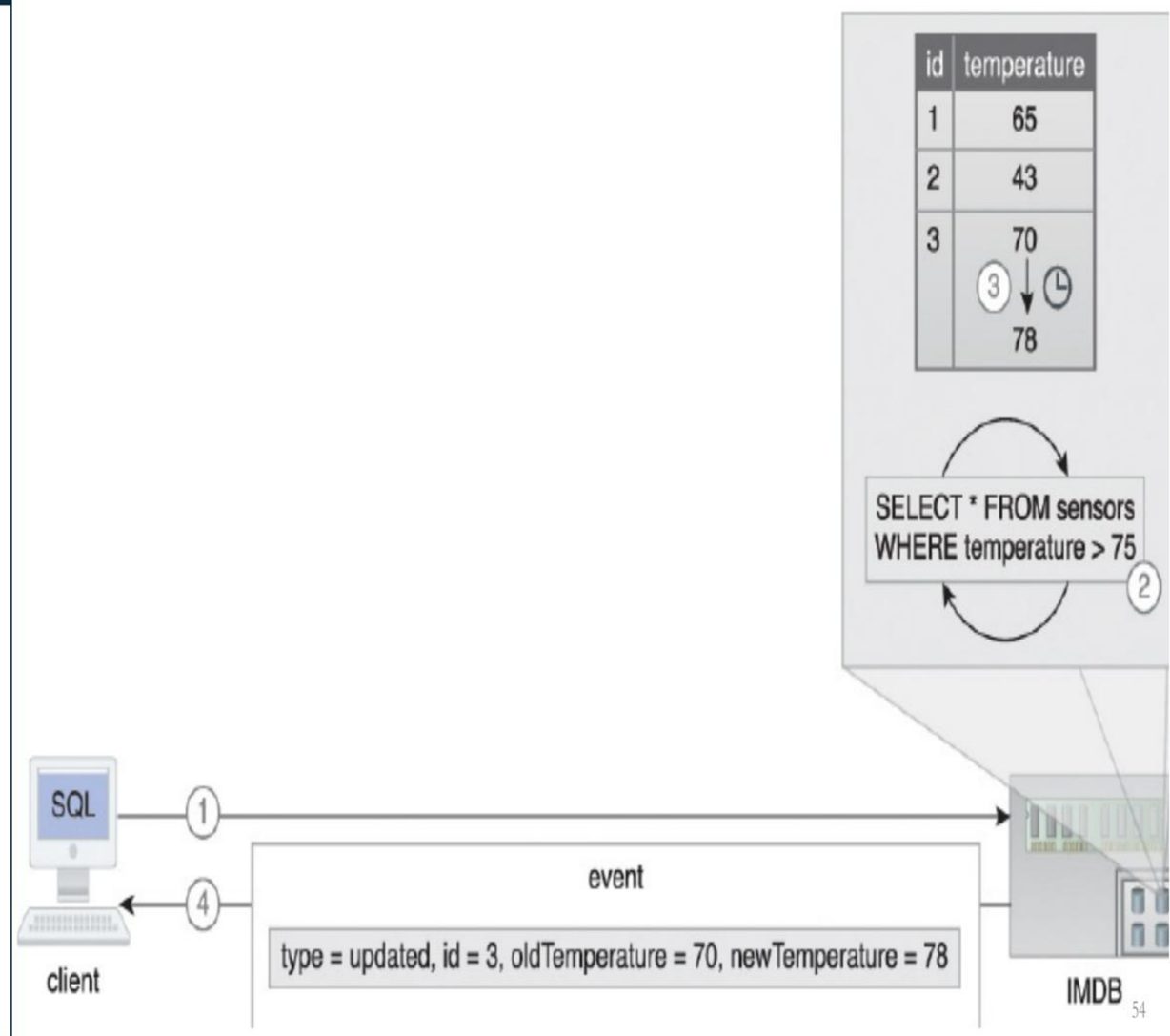
- IMDBs are in-memory storage devices that **employ database technology** and leverage the performance of **RAM** to **overcome runtime latency** issues of the on-disk storage devices.
- An IMDB can be **relational** in nature (**relational IMDB**) for the storage of **structured** data, or may leverage **NoSQL** technology (**non-relational IMDB**) for the storage of **semistructured** and **unstructured** data.
- IMDBs are heavily used in **real-time analytics** and can further be used for developing **low latency applications** requiring full ACID transaction support (relational IMDB).
- Examples include Aerospike, MemSQL, Altibase HDB, eXtreme DB and Pivotal GemFire XD



An example depicting the retrieval of data from an IMDB.

In-Memory Storage: In-Memory Database (IMDB)

- The previous example describes the following scenario:
  1. A relational dataset is **stored into an IMDB**.
  2. A **client requests** a customer record (id = 2) via SQL.
  3. The relevant customer **record is then returned** by the **IMDB**, which is directly manipulated by the client without the need for any deserialization.



An example of IMDB storage configured with a continuous query.

In-Memory Storage: In-Memory Database (IMDB)

- The previous example describes the following scenario:
  1. A client issues a **continuous query** (select * from sensors where temperature > 75).
  2. It is registered in the **IMDB**.
  3. When the temperature for any sensor exceeds 75F …
  4. … an **updated event** is sent to the subscribing **client** that contains various details about the event.

In-Memory Storage: In-Memory Database (IMDB)

- In the case of replacing an RDBMS with a relational IMDB, **little or no application code** change is required due to SQL support provided by the relational IMDB.
- However, when replacing an RDBMS with a NoSQL IMDB, **code change may be required** due to the need to implement the IMDB's NoSQL APIs.
- In the case of replacing an on-disk NoSQL database with a relational IMDB, **code change** will often be required to establish SQL-based access.
- However, when replacing an on-disk NoSQL database with a NoSQL IMDB, **code change** may still be required due to the implementation of new APIs.

Thank You