UNIT 4

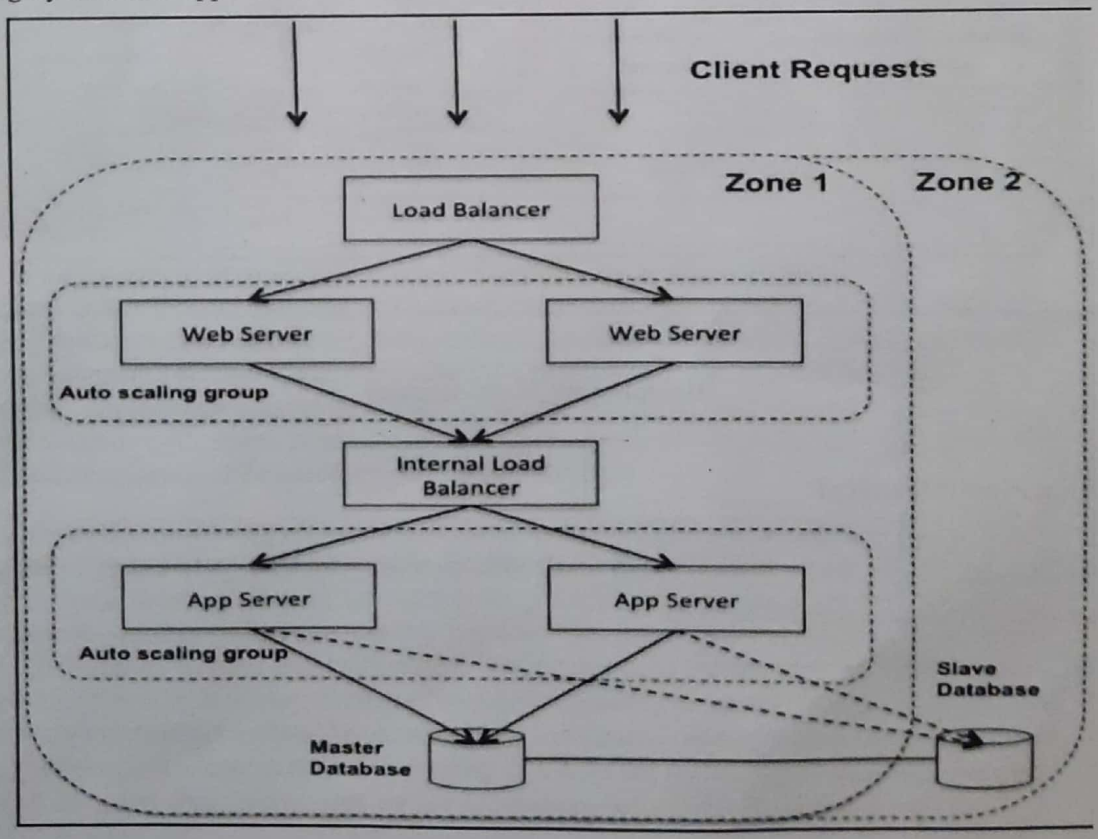
Designing Cloud Applications: Introducing cloud-based multitier architecture , Designing for multi-tenancy , Understanding cloud applications design principles Understanding emerging cloud-based application architectures , Estimating your cloud computing costs , A typical e-commerce web application, AWS Components, Managing costs on AWS Cloud, Application development environments

CHAPTER : Designing Cloud Applications

As an architect, you should have come across terms such as *loosely coupled*, *multitier*, *service oriented*, *highly scalable*, and so on. These terms are associated with architectural best practices, and you will find them listed in the first couple of pages of any system architecture document. These concepts are generally applicable to all architectures, and the cloud is no exception. In this chapter, we want to highlight how these are accomplished on the cloud.

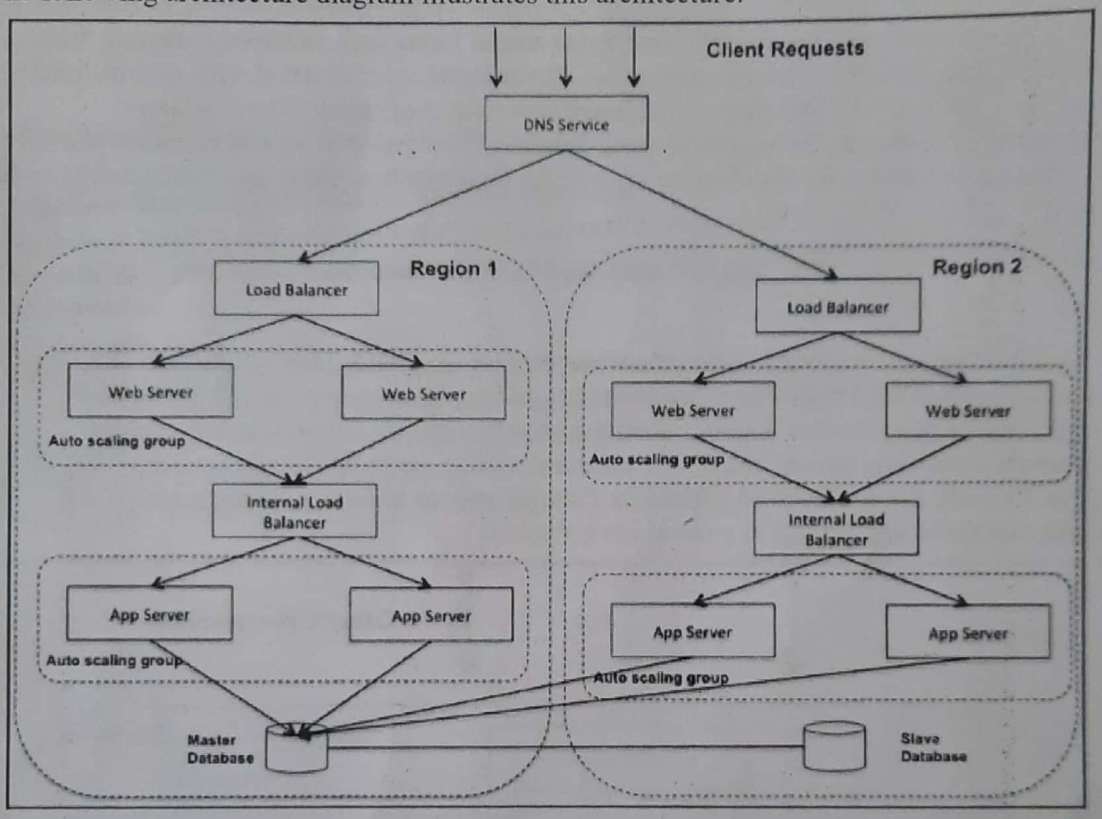
cloud-based multitier architecture

The following figure illustrates tiered architecture on the cloud. This architecture supports auto scaling and load balancing of web servers and application servers. Further, it also implements a master-slave database model across two different zones or data centers (connected with high speed links). The master database is synchronously replicated to the slave. Overall, the architecture represents a simple way to achieve a highly scalable and highly available application in a cloud environment.



Cloud applications can be deployed at multiple locations. Typically, using AWS terminology, these locations are regions (that is, separate geographical areas) or zones (that is, distinct locations within a region connected by low-latency networks). It is also possible to separate out the tiers across two different zones regions to provide for a higher level of redundancy including data center wide or zone-level failures or unavailability. We need to consider network traffic flow and data synchronization issues between the regions while designing high-availability architectures across multiple regions.

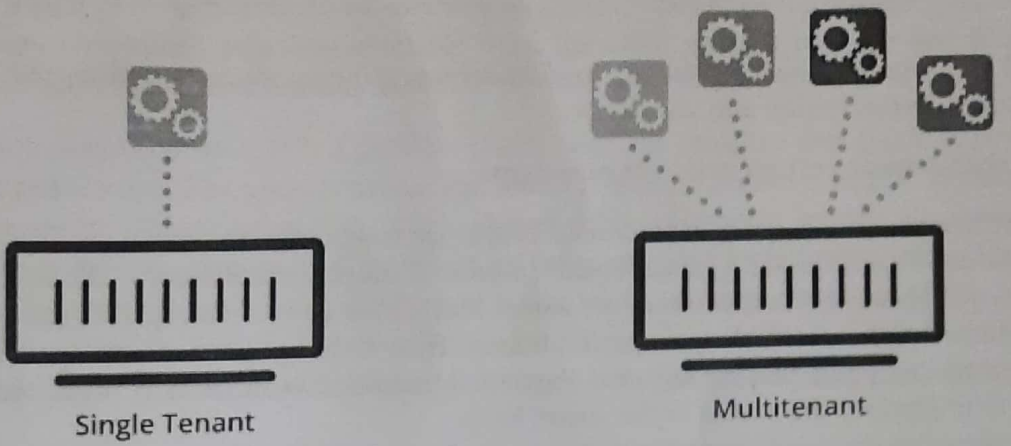
The following architecture diagram illustrates this architecture:

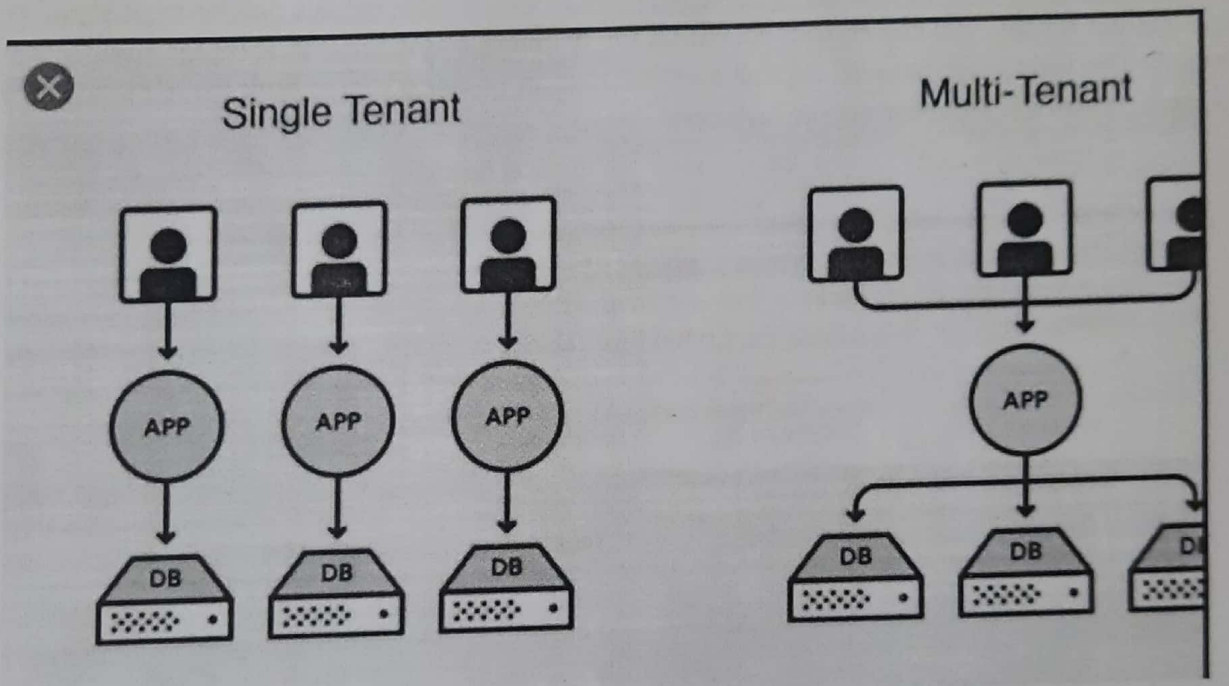


Designing for multi-tenancy

What is multitenancy?

In cloud computing, multitenancy means that multiple customers of a cloud vendor are using the same computing resources.





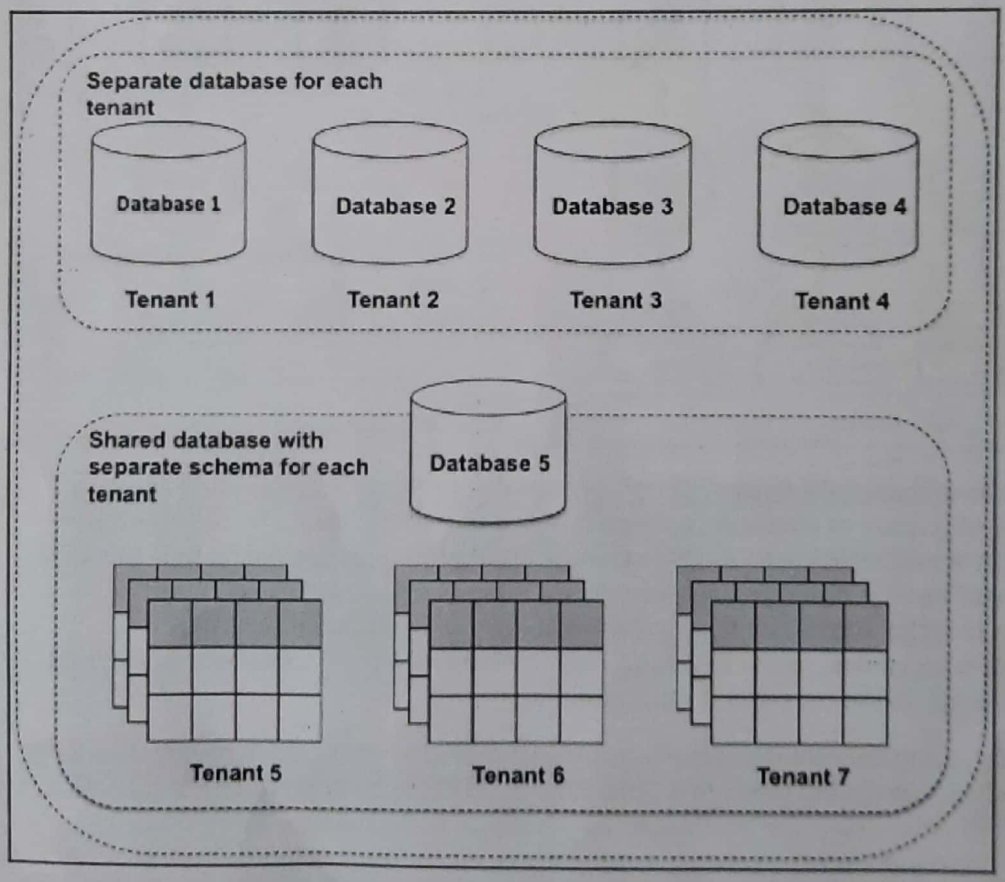
The major benefit of multi-tenancy is cost savings due to shared infrastructure and operational efficiency of managing a single instance of the application across multiple customers or tenants. However, multi-tenancy introduces complexity, and issues can arise when a tenant's action or usage can affect the performance and availability of the application for other tenants on the shared infrastructure. In addition, security, customization, upgrades, recovery, and so on, requirements of one tenant can create issues for other tenants and/or introduce further complexity.

Additionally, customers are concerned about their business-critical data residing on public cloud infrastructure (in the hands of a third-party SaaS application provider). It is of vital importance to ensure that the data architecture is robust and secure to satisfy the security standards, privacy policies, regulatory requirements, etc. in place at large enterprises (as well as smaller businesses). For example, businesses in the financial or healthcare sector are governed by a host of very strict regulatory requirements in terms of privacy, and location and usage of customer data. We need to have strict access control mechanisms to ensure a tenant does not access resources belonging to a different tenant. Simultaneously, we also need to ensure that mechanisms we use are operationally cost-effective and easy to administer.

In cloud architectures, the main factors to consider while implementing multi-tenancy are data security, extensibility, and scalability.

Addressing data-at-rest security requirements

There are two levels of security to be considered; at the tenant level (typically, an organization) and at the end-user level, of a given tenant. In order to implement a security model you need to create a database access account at the tenant level. This account can specify (using ACLs) the database objects accessible to a specific tenant. Then at the application level, you will need to prevent users from accessing any data they are not entitled to. A security token service can be used to implement the access at the tenant level.
In the approaches that implement multi-tenancy by having either separate databases or separate schema per tenant, you can restrict access tat the database or the schema level for a particular tenant. The following diagram depicts a very common scenario where both these models are present in a single database server instance:



If the database tables are shared across tenants then you need to filter data access by each tenant. This is accomplished by having a column that stores a tenant ID per record (to clearly identify records that belong to a specific tenant).

Aside from database level security, organizational policies or regulatory requirements can mandate securing your data at rest. The options for implementing encryption to protect your

data can range from fully automated solutions to manual ones to be implemented on the client side. There are several solutions available from the cloud service provider and third party vendors to implement these security models.

Regardless of the approach, it is a good practice to encrypt sensitive data fields in your cloud database and storage. Encryption ensures that the data remains secure even if a non-authorized user accesses it. This is more critical for shared database/schema models. In many cases, encrypting a database column that is part of an index can lead to full table scans. Hence, try not to encrypt everything in your database as it can lead to poor performance. Therefore, it is important to carefully identify sensitive information fields in your database, and encrypt them more selectively. This will result in the right balance between security and performance

---

Addressing data extensibility requirements

Having a rigid database schema will not work for you across all your customers. Customers have their specific business rules and supporting data requirements. They will want to introduce their own customization to the database schema.

One approach to achieving extensibility in the database schema is to pre-allocate a bunch of extra fields in your tables, which can then be used by your customers to implement their own business requirements. All these fields can be defined as string or varchar fields. You can also create an additional metadata table to further define a field label, data type, field length, and so on, for each of these fields on a per tenant basis

This approach is depicted in the following figure. Fields 1 to 4 are defined as extra columns in the customer table. Further, the metadata table defines the field labels and data types.

tbl_customers

tenant_id	customer_id	customer_fname	customer_lname	field1	field2	field3	field4
1	23	John	Smith	123, Holly St, NY, NY	11-07-1974	Null	(212)-555-7651
2	45	Harry	Snow	45, Barclay St	Los Angeles	CA	(310)-555-8956

tbl_customers_metadata

tenant_id	field1_label	field1_ datatype	field2_label	field2_ datatype	field3_label	field3_ datatype	field4_label	field4_ datatype
1	Address	String	Birthdate	Date	Null	Null	Home Phone	String
2	Street	String	City	String	State	String	Home Phone	String

A second approach takes a name-value pair approach, where you have a main data table that points to an intermediate table containing the value of the field and a pointer to a meta data table that contains the field label, data type, and so on, information. This approach cuts out potential waste in the first approach but is obviously more complicated to implement.

tbl_customers

tenant_id	customer_id	customer_fname	customer_lname	record_id
1	23	John	Smith	11
2	45	Harry	Snow	12
3	67	Tom	Builder	Null

record_id	metadata_id	field_value
11	111	123, Holly St, NY, NY
11	112	11-07-1974
11	113	(212)-555-7651
12	121	45, Barclay St
12	122	Los Angeles
12	123	CA
12	124	(310)-555-8956

tbl_customers_metadata

tenant_id	metadata_ id	field_label	field_ datatype
1	111	Address	String
1	112	Birthdate	Date
1	113	Home Phone	String
2	121	Street	String
2	122	City	String
2	123	State	String
2	124	Home Phone	String

A third approach is to add columns per tenant as required. This approach is more suitable in the separate database or separate schema per tenant models. However, this approach should generally be avoided as it leads to complexity in application code, that is, handling an arbitrary number of columns in a table per tenant. Further, it can lead to operational headaches during upgrades.

Understanding cloud applications design Principles

In this section, we will cover key guiding principles that are useful while designing cloud based applications. More specifically, we will introduce designing for scale, automated infrastructure, failures, parallel processing, performance, and eventual consistency.

Designing for scale

Traditionally, designing for scale meant carefully sizing your infrastructure for peak usage and then adding a factor to handle variability in load. At some point, when you reached a certain threshold on CPU, memory, disk (capacity and throughput) or network bandwidth, you would repeat the exercise for handling increased loads and initiate a lengthy procurement and provisioning process. Depending on the application, this could mean a scale up (vertical scaling) with bigger machines or scale out (horizontal scaling) with more machines being deployed. Once deployed, the new capacity would be fixed (and run continuously) whether the additional capacity was being utilized fully or not.

In cloud applications, it is easy to scale—both vertically and horizontally. additionally, the increase and the decrease in the number of nodes (in horizontal scalability) can be done automatically to improve resource utilization and manage costs better.

Automating cloud infrastructure

During failures or spikes in load you do not want to be provisioning resources, identifying and deploying the right version of the application, configuring parameters (for example, database connection strings), and so on. Hence, you need to invest in creating ready-to launch machine images, continuously monitor your system metrics to dynamically take action such as auto scaling, develop scripts for automated deployments, centrally store application configuration parameters, and boot new instances quickly by bootstrapping your instances, and so on.

It is possible to automate almost everything on the cloud platform via APIs and scripts, and you should attempt to do so. This includes typical operations, deployments, automatic recovery actions against alerts, scaling, and so on. For example, your cloud service may also provide an auto healing feature. You should leverage this feature to ensure failed/unhealthy instances are replaced and restarted with the original configurations.

Designing for failure

Assume all things will fail. Ensure you carefully review every aspect of your cloud architecture, and design for failure scenarios against each one of them. In particular, assume hardware will fail, cloud data center outages will happen, database failure or performance degradation will occur, expected volumes of transactions will be exceeded, and so on. In addition, in an auto-scaled environment, for example, nodes may be shutdown in response to load getting back to normal levels after a spike. Nodes may also be rebooted by the cloud platform. There can be unexpected application failures. In all these cases, the design goal should be to handle such error conditions gracefully, and minimize any impact to user experience.

There should be a strong preference to minimize human or manual intervention. Hence, it is better to implement strategies using services made available by the cloud platform to reduce the chances of failures or automate recovery from such failures.

Designing for parallel processing

It is a lot easier to design for parallelization on the cloud platform. You need to use parallel designs throughout your architecture from data ingestion to its processing. So, use multithreading for parallelizing your cloud service requests, distribute load using load balancing, ensure multiple processing components or service endpoints are available via horizontal scaling, and so on.

Designing for performance

When an application is deployed to the cloud, latency can become a big issue. There is sufficient evidence that shows that latency leads to loss in business. It can also severely impact user adoption.

You will need to attack the latency issue through approaches that can improve the user experience by reducing the perceived and real latency. For example, some of the techniques you can use include rightsizing your infrastructure, using caching and placing your application and data closer to your end users.

Designing for eventual consistency

Depending on the type of applications you have designed in the past, you may or may not have come across the concept of eventual consistency (unless you have worked extensively on distributed transactions oriented applications). However, it is fairly common in the cloud world. After a data update, if your application can tolerate a few seconds delay before the update is reflected across all replicas of the data then eventual consistency can lead to better scalability and performance.

Normally, eventual consistency is the default behavior in a cloud data service. In case the application requires consistent reads at all times then some cloud data services provide the flexibility to specify strongly consistent reads. However, there are several cloud data services that support the eventually consistent option only.

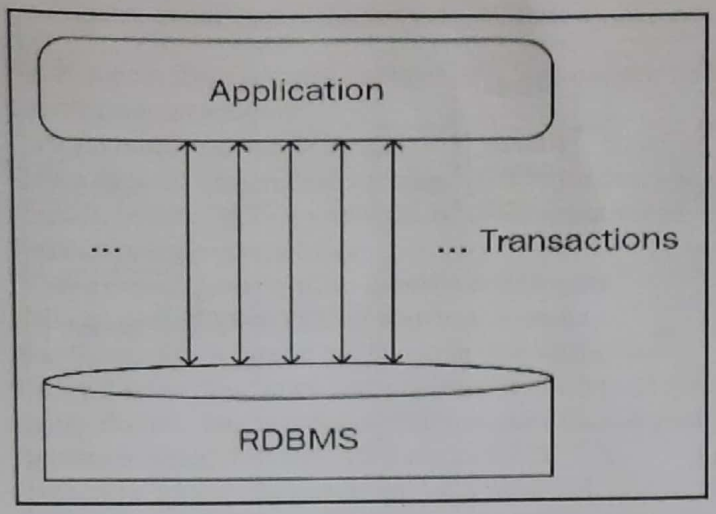
==========

Understanding emerging cloud-based application architectures

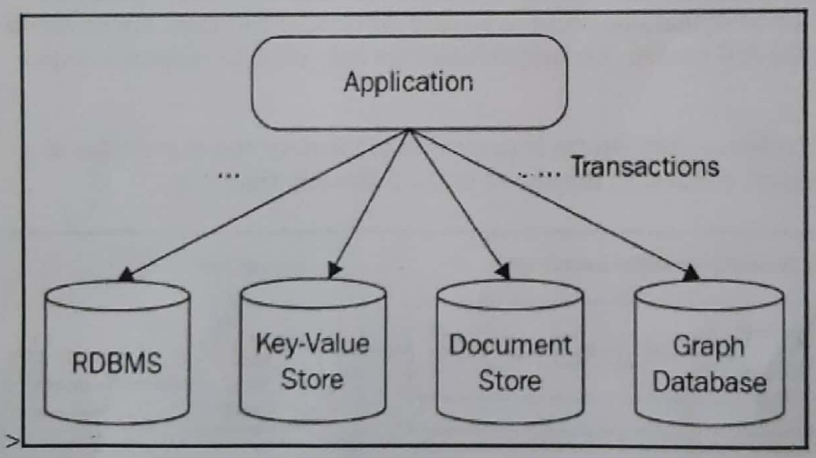
In this section, we will describe common architecture patterns and deployment of some of the main processing models being used for batch processing, streaming applications, and machine learning pipelines. The underlying architecture for these processing models are required to support ingesting very large volumes of various types of data arriving at high velocities at one end, while making the output data available for use by analytical tools, reporting and modeling software, at the other.

Understanding polyglot persistence

As organizations start employing big data and NoSQL-based solutions across a number of projects, a data layer comprising of RDBMSs alone is no longer the best solution for all the use cases in a modern enterprise application. The following figure illustrates a situation that is rapidly disappearing across the industry:

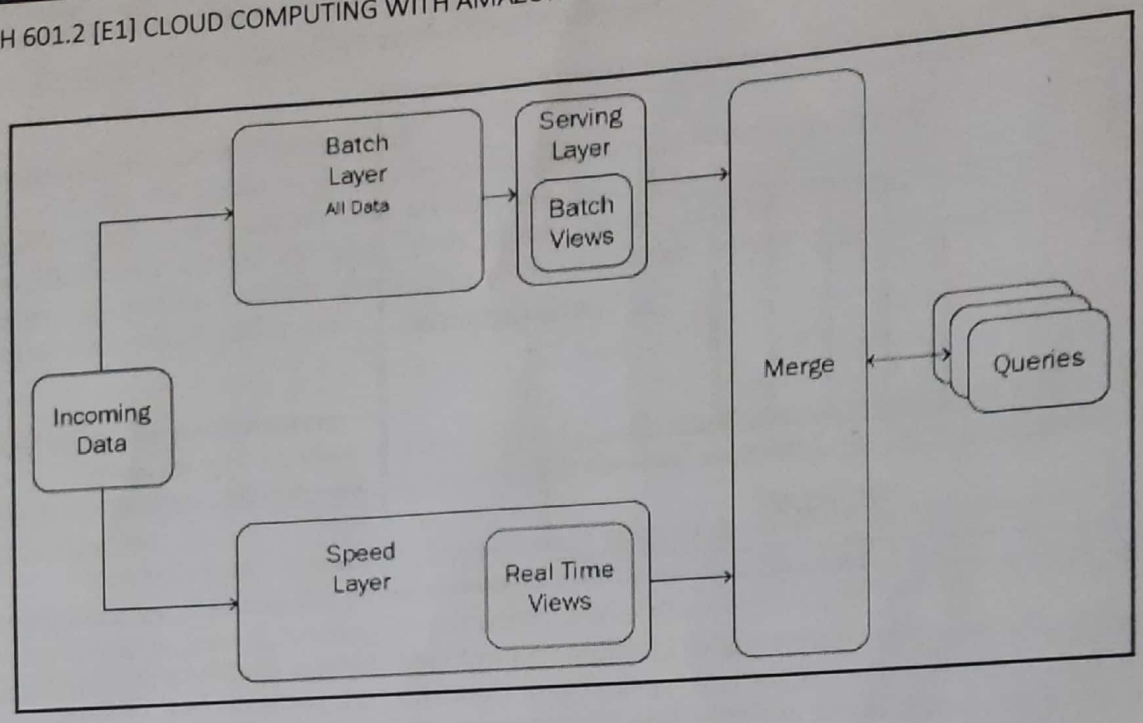


A more typical scenario comprising of multiple types of data stores is shown in the following figure. Applications today use several types of data stores that represent the best fit for a given set of use cases. Using multiple data storage technologies, chosen based upon the way data is being used by applications, is called polyglot persistence. For example, Apache Spark is an excellent enabler of this and other similar persistence strategies in the cloud or on-premise deployments:



Understanding Lambda architecture

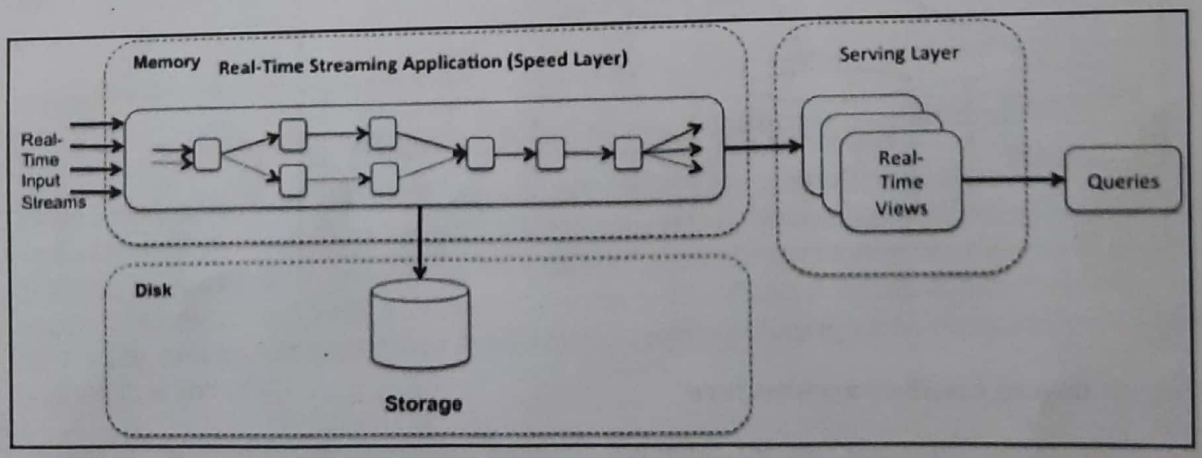
The Lambda architectural pattern attempts to combine the best of both worlds—batch processing and stream processing. This pattern consists of several layers: **Batch Layer** (ingests and processes data on persistent storage such as HDFS and S3), **Speed Layer** (ingests and processes streaming data, that has not been processed by the batch layer yet), and the **Serving Layer** that can combine outputs from the batch and speed layers to present merged results. This is a very popular architecture in Spark-based cloud environments because it can support both batch and speed layer implementations with minimal code differences between the two. The following figure depicts the Lambda architecture as a combination of the batch processing and stream processing:



Understanding Kappa architecture

Kappa architecture is simpler than the Lambda pattern as it comprises of the speed and serving layers only. All the computations occur as stream processing and there are no batch recomputations done on the full dataset. Recomputations are only done to support changes and new requirements.

Typically, the incoming real-time data stream is processed in memory and is persisted in a database or HDFS, to support queries, as illustrated in the following figure:



Kappa architecture can be realized using a queueing solution such as Apache Kafka or Kinesis. If the data retention times are bound to several days to weeks then Kafka could also be used to retain the data for the limited period of time.

IMP QUESTIONS

a) With a neat diagram explain Multi tier architecture on the cloud/ Explain cloud-based multitier architecture
b) Explain multi-tenancy in cloud
c) Briley explain implementation issues of multi-tenancy in cloud
d) Explain in detail design principles of cloud applications
e) Explain polyglot persistence
f) With a neat diagram explain Lambda architecture
g) With an neat diagram explain Kappa architecture
h) Briefly explain A typical e-commerce web application
i) Briley discuss functional requirements of a A typical e-commerce web application
j) Briley discuss non functional requirements of a A typical e-commerce web application
k) Explain in detail different AWS components
l) Explain how to manage costs on AWS cloud
m) Explain in detail different application development environments

Introducing AWS Components

This chapter will introduce you to some of the key AWS components and services. We will also cover strategies to lower your cloud infrastructure costs and their influence on your architectural decisions.

AWS components mainly divided into following categories

- Amazon compute-related services
- Amazon storage-related services
- Amazon database-related services

etc

Amazon compute-related services

Amazon EC2

Amazon EC2 is a web service that provides compute capacity in the AWS cloud. You have several choices of instance types, operating systems, and software packages available. You will need to choose an appropriate mix of instance types based on your specific use cases. These instance types include general purpose, accelerated computing, compute-optimized, memory-optimized, and storage-optimized instances. For example, you would choose compute-optimized instances for your compute-intensive workloads and accelerated computing instances for GPU-based processing, typically, used for deep learning applications. In addition, each instance type includes one or more instance sizes to match the scalability requirements of your specific workloads.

Amazon EC2 allows you to configure the memory, CPU, instance storage, and your choice of operating system and applications. You can bundle the operating system, application software, and associated configuration settings into an Amazon Machine Image (AMI). Then, use it to provision or decommission multiple virtualized instances using web service calls. EC2 instances can be resized and the number of instances scaled, up or down, to match your requirements or demand.

AWS Lambda

The **AWS Lambda** service supports executing your code in response to certain events within your application. Such events could include website clicks, image uploads, updates to certain data fields, document transformation, indexing, anomaly detection, errors detected in log files, sensitive or auditable events, unusual readings from sensors, and so on. You can also send notifications using SNS in response to these events.

Amazon EC2 container service

The **Amazon EC2 container service** is a cluster management and configuration management service. This service enables you to launch and stop container-enabled applications via API calls.

Amazon storage-related services

Amazon S3

Amazon S3 is a highly durable and distributed data store. Using a web services interface, you can store and retrieve large amounts of data as objects in buckets (containers). The stored

objects are also accessible from the web via HTTP. It supports the implementation of stringent security and compliance policies on the stored data to ensure the security of dataat-rest. S3 is also an economical storage option for massive amounts of data typically used by analytics, IoT, and machine learning applications.

Amazon EBS

Amazon EBS is a highly available and durable persistent block-level storage volume for use with Amazon EC2 instances. You can configure EBS with SSD (general purpose or provisioned IOPS) or magnetic volumes. Each EBS volume is automatically replicated within its Availability Zone (AZ).

Amazon Glacier

Amazon Glacier is a low-cost storage service that is typically used for archival and backups. The retrieval time for data on Glacier, based on the option selected, varies from a few minutes to several hours.

Amazon database-related services

Amazon DynamoDB

Amazon DynamoDB is a NoSQL database service offered by AWS. It supports both document and key-value pairs data models and has a flexible schema. Integration with other AWS services, such as Amazon Elastic MapReduce (Amazon EMR) and Redshift, provide support for big data and BI applications, respectively. In addition, the integration with AWS Data Pipeline provides an efficient means of moving data into and out of DynamoDB.

Amazon Redshift

Amazon Redshift is a highly scalable data warehouse service offered by AWS. You can leverage your existing investments in BI tools because Redshift can work with them.

Amazon Relational Database Service (RDS)

Amazon Relational Database Service (RDS) provides an easy way to set up, operate, and scale a relational database in the cloud. Database options available from AWS include MySQL, Oracle, SQL Server, PostgreSQL, and Amazon Aurora. With RDS, you can launch a DB instance and get access to a full-featured MySQL database and not worry about managing or administering it. Amazon RDS can significantly reduce effort on common database administration tasks, such as backups and patch management.

Launching a database requires you to select a database engine, license type, an instance class, and storage capacity. The RDS instances are preconfigured for the DB instance you choose. It is equally easy to monitor and scale your database instance (for both compute and storage capacities).

And other important components of aws are

Amazon SQS

Amazon Simple Queue Service (Amazon SQS) is a reliable, highly scalable, hosted distributed queue to store messages as they travel between computers and application components.

Amazon SNS

Amazon Simple Notification Service (SNS) provides a simple way to notify applications or people from an AWS cloud application. It uses the publish-subscribe protocol.

Amazon SES

Amazon Simple Email Service (SES) is a cloud-based email sending and receiving service. You can use Amazon's SMTP interface or integrate SES directly into your applications using AWS SDKs.

AWS WAF

AWS WAF is a web application firewall that helps protect your web applications from incidents that could affect application availability, compromise security, or consume excessive resources. AWS WAF gives you control over which traffic to allow or block to your web applications by defining appropriate rules.

Amazon EMR

Amazon Elastic MapReduce (EMR) provides a hosted Hadoop framework running on Amazon EC2 and Amazon S3 that allows you to create customized MapReduce jobs.

AWS CloudFormation

The **AWS CloudFormation** service helps in creating and managing a collection of related AWS resources. We can create templates to describe the AWS resources and any associated dependencies or runtime parameters required to run your application. In addition to provisioning AWS resources, CloudFormation can also be used to update them in an orderly and predictable manner.

Amazon CloudWatch

CloudWatch is a monitoring service for your AWS resources. It enables you to retrieve monitoring data, set alarms, troubleshoot problems, and take actions based on the issues in your cloud environment.

Amazon Machine Learning

Amazon Machine Learning provides developers with visualization tools and wizards to help create ML models without having to learn complex ML algorithms or manage the underlying infrastructure. It allows you to start small and then scale, as your application grows. Typically, after the modeling phase is completed, Amazon Machine Learning makes it easier to obtain the results using simple APIs. These APIs assist in serving these results in real time. You can also use Amazon Batch for processing predictions based on large batches of data.

Managing costs on AWS cloud

There are broadly three areas for cost optimizations on the cloud: operational, infrastructural, and architectural optimizations. It is important to note that costs should not focus on infrastructure alone. You should include code changes in your deliberations because sometimes, it makes sense to focus on improving the code rather than infrastructure alone. There are many architectural decisions and trade-offs to be made to achieve the best results from a cost saving's perspective

Costs are a big motivation to use cloud infrastructure, and AWS provides many different ways of saving on your AWS bills. However, it is up to you to take advantage of all the saving pportunities available. As a simple guideline start with minimal infrastructure and iterate from there to optimize your infrastructure costs.

Some of the tools available to manage costs in AWS cloud are

AWS Cost Explorer

AWS Cost Explorer has an easy-to-use interface that lets you visualize, understand, and manage your AWS costs and usage over time. Get started quickly by creating custom reports (including charts and tabular data) that analyze cost and usage data, both at a high level (e.g., total costs and usage across all accounts) and for highly-specific requests (e.g., m2.2xlarge costs within account Y").

AWS Budgets

AWS Budgets gives you the ability to set custom budgets that alert you when your costs or usage exceed (or are forecasted to exceed) your budgeted amount. You can also use AWS Budgets to set RI utilization or coverage targets and receive alerts when your utilization drops below the threshold you define. RI alerts support Amazon EC2, Amazon RDS, Amazon Redshift, and Amazon ElastiCache reservations.

Budgets can be tracked at the monthly, quarterly, or yearly level, and you can customize the start and end dates. You can further refine your budget to track costs associated with multiple dimensions, such as AWS service, linked account, tag, and others. Budget alerts can be sent via email and/or Amazon Simple Notification Service (SNS) topic.

Budgets can be created and tracked from the AWS Budgets dashboard or via the Budgets API.

AWS Cost & Usage Report

The AWS Cost & Usage Report is a single location for accessing comprehensive information about your AWS costs and usage.

The AWS Cost & Usage Report lists AWS usage for each service category used by an account and its IAM users in hourly or daily line items, as well as any tags that you have activated for

cost allocation purposes. You can also customize the AWS Cost & Usage Report to aggregate your usage data to the daily or monthly level.

Strategies to lower AWS costs

Monitoring and analyzing costs

AWS platform provides a set of tools to help monitor and analyze your costs. These include the AWS TCO Calculator, the simple monthly calculator, AWS billing console, which shows you an itemized bill, and AWS Cost Explorer, which gives you costs trends information across different time periods.

The TCO calculator can be used to compare on-premise versus on-cloud costs. The AWS simple monthly calculator is a useful tool to select the various AWS services and options to compute your costs.

You can use the AWS billing console to drill down deeper into the AWS bill to see an itemized list of components and their costs. This can help us identify optimization targets. You can also set AWS billing alerts, which send you automatic notifications when your bill hits some preset threshold.

Choosing the right EC2 Instance

The EC2 instances you choose are directly dependent on your application characteristics. Based on the application characteristics shortlist a few instance types available from AWS. EC2 types include several families of related instances available in sizes ranging from micro to extra large. These classes include general purpose, compute-optimized, memoryoptimized, storage-optimized, and accelerated computing instances.

You should then do a few tests to analyze the performance of the shortlisted instances against increasing loads. It is a good idea to understand the upper limit of these instances in terms of number of concurrent users or throughput they can support. Then Compare the costs against different throughput levels.

Turn-off unused instances

Typically, there is an inordinate amount of focus on production spends but a lot of that spend (in some cases, more than production) is on dev and test. It is surprising how many times you find unused instances adding to your bills. This usually happens when someone provisions an instance to conduct an experiment or check out a new feature and then fails to delete the instance, when done. It also happens a lot during testing when the test environment is, carelessly, left running through the weekend, or after the testing activity has been completed. It is important to check your bills and usage data to minimize such costs.

Using Auto Scaling

Auto Scaling scale your compute instances to the extent required otherwise scale down, automatically. Auto Scaling aligns your deployed infrastructure to the demand at any given point in time