|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data Ingestion and Transformation | | | | | | | | | | | | | |
| Data Sources | | | | | | Databases, Objects, IoT, Mobile. | | | | | | | |
| AWS Data Lake | | | | | Raw Data in S3 (from data source) -> AWS Glue (for transformation) -> Transformed Data in S3 -> published downstream for Data Analytics  Glue transforms data by cataloging, cleaning, and crawling the data to infer the schema | | | | | | | | |
| Data Analytics | | | | | Amazon QuickSight: BI and data visualization  Amazon SageMaker: deploying ML models | | | Amazon OpenSearch Service: search & analytics  Amazon Athena: running SQL queries | | | | | |
| Data Lakes vs Data Warehouses | | | | | Data Lakes: Centralized repo for raw data in its native format. – Stores structured, semi-structured, unstructured data. – Don’t require a predefined schema (schema-on-read). – Lower cost for storage | | | | | | | | |
| Data Warehouses: Centralized repo for historical data from various sources. – Mainly store structured and organized data. – Require predefined schema (schema-on-write). – Best for transactional systems, CRM, ERP systems. - Higher cost for storage. – More effort to maintain, higher operational cost | | | | | | | | |
| AWS Glue | | | | | Serverless solution for ingesting, discovering, prepping and combining data  Collection of capabilities (ETL jobs, Data Catalog, Glue Studio, Data Quality and DataBrew) | | | | | | | | |
| For ETL: - allow multiple data sources like S3, DynamoDB, and RDS  - Can build ETL jobs using diff mtds like Python Shell, Glue Streaming, and Cloud Studio  4 types of Data Processing Units (DPUs):  1) G.1X (memory-intense). Mapped to 1 DPU  2) G.2X (workloads like ML or extreme transfers of data like ML Transforms). Mapped to 2 diff DPUs  3) G.025X (low volume and or sporadic data). Mapped to a quarter of a DPU. Cheap  4) Standard (for older version of Glue. Not recommended for Glue later than V1.0) | | | | | | | | |
| Data Catalog: manage metadata of data assets. – auto infer schema of DB. – can have multiple sources | | | | | | | | |
| Glue Studio: provide visual interface of ETL workflows for easy development and management  - e.g. drag and drop functions to build transformation logic. – enables job orchestration & monitoring | | | | | | | | |
| DataBrew: visual data preparation/profiling tool (can use SQL to process data). Gain insights to data  - allow exploration and interaction w dataset. – built-in data transformations fns to apply to datasets | | | | | | | | |
| Benefits of AWS Glue | | | | | 1) Fully managed service. 2) Allows integration of large amounts of data to multiple sources  3) Can consume process and buffer data in real time. 4) Can do real-time metrices & reporting | | | | | | | | |
| Event-Driven Triggers | | | | | Configure Glue workflows to start automatically in response to events  E.g. data uploads to S3, changes in DynamoDB tables, messages in Amazon Simple Notification Service (SNS) topics, EventBridge | | | | | | | | |
| Time-Based Triggers | | | | | | | Schedule workflows at specific intervals or particular dates and times. Cron notations | | | | | | |
| On-Demand Trigger | | | | | | | Start workflow when you want. | | | | | | |
| Amazon Kinesis | Fully managed data streaming service to process and analyze streaming data  - Family of services (Data Streams, Data Firehose, Video Streams and Managed Apache Flink) | | | | | | | | | | | | |
| Kinesis Data Streams: serverless way to stream and store data  - each Stream is composed of a seq of Shards containing data records  - A data record contains a seq num/ID, partition key, and a data blob up to 1MB  - Use partition key to group Shards tgt. – Data Producer -> Kinesis (Data Stream + Firehose) -> Consumer  1) Data Producers: app that ingests incoming data into a Stream in near real-time  2) Data Consumers: app that takes collected data and processes or stores it (S3, Redshift) | | | | | | | | | | | | |
| Data Firehose: Delivery Stream = allows you to store / send data to services to process it | | | | | | | | | | | | |
| Apache Flink: To batch process data using programming languages or SQL near real time.  - Can create Apache Zeppelin notebooks to run CMDs against data. If not batch, use Kinesis Data Analytics | | | | | | | | | | | | |
| Use cases: anomaly detection in IoT, log processing for ML, pattern detection, click stream analysis | | | | | | | | | | | | |
| Benefits of Kinesis | | | | | 1) Fully managed service. 2) Serverless. 3) Capture streaming data in near real-time.  4) Auto scaling | | | | | | | | |
| Kinesis vs SQS | | Kinesis default stores records for 24 hrs to 7 days  Multiple consumers of the same data records  Support in-order processing with a shard, however no ordering btw shards is guaranteed  Related records can be routed to same processor | | | | | | | SQS message retention period: 1 min to 14 days  1 consumer at a time, once read, delete from queue  Standard SQS queue don’t guarantee in-order processing, but FIFO support it w limitations  Don’t support routing | | | | |
| Kinesis vs Kafka | | Kinesis configurability is limited in how it writes data to end resources  Cloud native and pay-as-you-go = lower costs  Designed for easy use, few clicks | | | | | | | Kafka more configurable in how it writes data (type of file format data saved in)  More engineering hours to get started = higher cost  Require more to implement, use and maintain | | | | |
| Amazon Redshift | | - Fully managed data warehouse based on PostgreSQL. - For OLAP. - Supports distributed workload  - Stores data in columns instead of rows. – Use Redshift Query Editor for querying (console or Redshift API)  - Query petabytes of data (structured, semi-structured). – Runs within a VPC.  – Support Multi-AZ deployment. – Offers serverless option | | | | | | | | | | | |
| Data Warehouse allows combining diff data sources into a single place to run custom analytics  Can use Log analysis to gain insights into click stream behavior, what sensors they use …  Redshift uses a cluster architecture. – Each cluster composed of nodes. (up to 128 compute nodes)  Single-Node Cluster: Leader node + Compute node. For small datasets or testing  Multi-Node Cluster: Leader node + ≥ 2 Compute nodes. For large datasets and production environments  Leader Node: receives queries from client apps -> parses queries and develop query execution plans -> coordinates parallel execution of these plans -> aggregate results of queries and return to client  Compute Node: partitions the job into slices (slice management) -> runs query execution plans -> sends intermediate results back to leader node.  Massively Parallel Processing (MPP) framework | | | | | | | | | | | |
| Dense Compute (DC) Node: For compute-intensive workloads. E.g. High query performance, concurrent queries or real-time analytics, memory intensive workloads  - compute and storage charges coupled tgt. Cost structure more straightforward  Dense Storage (DS) Node: For large data warehouses. E.g. Dealing with TB or PB of data, need balance btw compute and storage resources, slower performance  RA3 Nodes: Scales compute and storage independently. Recommended by AWS over DS nodes. Auto offloads data to S3 if node grows beyond capacity of local SSD  - compute and storage charges separately. Keep track of cost of compute and data stored in S3 | | | | | | | | | | | |
| Scaling Options: 1) Add Redshift Clusters (Concurrency Scaling): Auto adds more compute power temporarily to meet demand of high concurrency  2) Query Data in S3 via Redshift Spectrum: Analyze data in S3 w/o loading data into Redshift or ETL jobs  3) Resize Clusters by Updating Nodes (horizontal or vertical). Scale in/out by add/remove nodes. Scale up/down by changing node types | | | | | | | | | | | |
| To resize clusters:  1) Elastic Resize: - For add/remove nodes, does an in-place resize; cluster is unavailable during resize  - For upgrading node type within same cluster type, Redshift takes snapshot and restore it to new cluster (source cluster in read-only mode). - Don’t support downgrading. – Faster to resize. – Retains system logs  2) Classic Resize: -copies tables and metadata from source cluster to a new target cluster.  – For add/remove nodes, source cluster is in read-only mode. - For changing node types, (e.g. DS2 to RA3)  - For changing cluster type. - Takes more time to complete. – Don’t retain system log tables | | | | | | | | | | | |
| Loading Data to Redshift | | Sources can include: S3, Kinesis, EMR, DynamoDB, DMS (DB Migration Service), RDS, …  Option 1) Loading from S3. Data source -> S3 -> Use COPY command to copy data to Redshift  Option 2) Loading from DB. Directly use COPY command to copy data from DB to Redshift | | | | | | | | | | | |
| Option 1: 1) Create Redshift cluster. 2) Split file to upload into multiple files to enable COPY command to leverage MPP architecture and load data in parallel. Then, upload files to S3  3) Create IAM role, assign S3 read permissions, attach role to cluster. 4) Use COPY to load data from S3 | | | | | | | | | | | |
| *Copy target\_table from “s3://my\_bucket/my\_table/table\_1.txt” iam\_role ‘arn:aws:iam…’*  *UNLOAD (‘SELECT \* FROM my\_table’). TO “s3://…” iam\_role. ‘arn:aws:iam…’ FORMAT PARQUET*  *VACUUM FULL table\_name* : Takes longer & more space as it locks the table down and make a full copy to reclaim space (from deleted/updated rows) and re-sort rows based on default threshold  *VACUUM sort only table\_name to 74 percent* : Re-sort rows only if < 75% of the rows are already sorted  *VACUUM reindex table\_name* : Reindex and then vacuum | | | | | | | | | | | |
| AWS Lambda | | AWS Managed service. AWS manages OS and security. Pay for running code measured in milliseconds.  Can rollback version of lambda functions. 1) Create fn. 2) Write/upload code to fn. 3) Specify trigger event  To run a Lambda Service, you need: - the Lambda function, - IAM Execution Role,  - Memory size specification, - Execution Timeout (limit is 15 mins), - Event source mapping/trigger  Can trigger lambda fn by a trigger event or invoke directly using the lambda fn ARN | | | | | | | | | | | |
| Triggering Data Ingestion | | S3 Event Trigger = invoke action for a S3 bucket. Triggers work w SNS, SQS, Lambda, and EventBridge. | | | | | | | | | | | |
| EventBridge Triggers = if an event matches the pattern definition, EventBridge sends the event to the target specified in the rule. Events can come from AWS services or custom apps | | | | | | | | | | | |
| Redshift Events = log of things happening within the cluster (new data, changes in data/table). Events can have notification subscriptions and can be held for several weeks | | | | | | | | | | | |
| Kinesis vs EventBridge vs Lambda | | | | Kinesis Data Stream can be used as an event source for a trigger. E.g. action of pushing data through Kinesis to a singular mapped destination can be a trigger for Lambda  EventBridge follows event patterns defined within rules. When event occurs that match pattern, this triggers an action in one or multiple destinations  Lambda provides the most option to what events can trigger a fn, these always go to 1 mapped dest | | | | | | | | | |
| Consuming Data APIs | | | | SOAP (Simple Object Access Protocol) APIs. Client and server exchange messages using XML  More for secure transmissions as all request must follow a fixed format but can be too verbose | | | | | | | | | |
| RPC (Remote Procedure Calls) APIs. Client completes a fn/action on the server and server sends the output back to client | | | | | | | | | |
| WebSocket API. API development that uses JSON objects to pass data. Supports 2-way communication btw client and server. Use for bidirectional communication like chat apps or live updates | | | | | | | | | |
| REST (Representational State Transfer) APIs. Most flexible API, client sends requests to server as data | | | | | | | | | |
| AWS Data Exchange API: - move data btw services. – can set up subscriptions to view and access data  - As a provider, can create and manage data before you publish it  Parts of Data Exchange API: 1) Data Set (a collection of data). 2) Asset (any piece of data)  Data Exchange API allows access to Redshift; don’t require a persistent connection to DB.  Provides a secure HTTP endpoint and integration w SDK | | | | | | | | | |
| Steps to work w Redshift Data API: 1) Check caller/you are authorized to call DB  2) Determine if you have the authentication credentials (either temporary or via Secrets Manager)  3) Call API using AWS CLI from code or query editor in Redshift console | | | | | | | | | |
| Lambda API | | | | User w serverless apps. Uses the API gateway which supports API routing, serving HTML pages or binary files, issuing redirects, … | | | | | | | | | |
| Security and Connec-tions | | Network Access Control Lists (NACLs). - Provides stateless (check all traffic regardless of direction or source) access controls in VPCs and subnets. - Layer of security in VPC subnet acting as a firewall  - Can allow or deny access through protocols and port ranges | | | | | | | | | | | |
| Security Groups. – Filter who can access resources (ec2 instance, DB) in a VPC. – Provides stateful access control. – Can allow or block traffic based on a block IP addr, other security groups, or running services | | | | | | | | | | | |
| Security for RDS. Best practices: - Isolate DB. – Implement security groups to limit IPs that can access.  - Limit what resources share this isolated VPC so that you can monitor traffic flow. - Set up NACLs | | | | | | | | | | | |
| ETL | | | Data sources -> extract -> Data Lake (transform) -> load -> Database -> Publish -> Reports -> Business  Transformation: - Filtering data: Removing cols or rows before inserting data into final dest  - Mapping data: taking input from data source and changing/conforming it into another data format  - Deriving variables. – Aggregating data across a particular dimension. – Deduplicating Data  - Splitting Data: refine unstructured data by splitting certain fields before insert into final table | | | | | | | | | | |
| EMR vs Glue | | | Elastic Map Reduce: Designed for Integration of ETL  More customized options for 3rd party support  You must load your data connectors  Harder to set up, but can be cheaper | | | | | | | | Glue: Designed for big data  More built-in capabilities  Auto migration and allow other ETL providers  Easier to use, but more expensive due to serverless | | |
| Apache | Hive: querying and managing large datasets residing in distributed storage | | | | | | | | | | | | |
| Hadoop: distributed processing of large datasets across clusters of computers. Can handle unlimited data | | | | | | | | | | | | |
| AVRO: data serialization system for efficient data storage and transmission | | | | | | | | | | | | |
| Spark: fast large scale data processing in memory. RAM limitations | | | | | | | | | | | | |
| Airflow: build, schedule and monitor batch workflows or data pipelines  Amazon Managed Workflows for Apache Airflow (AMWAA): enable Airflow to provide managed orchestration  - Creates environment that has you cluster (include scheduler, workers, web servers)  - Upload DAG code into S3 (include DAG folder, Plugin zip file, requirements file). – Run DAGs in Airflow | | | | | | | | | | | | |
| AWS Step Functions | | | Serverless orchestration via a GUI to view app workflows as event-driven steps  - Standard workflows execute each step exactly once  - Express workflows have an at-least-once step execution but will run for at least 5 mins | | | | | | | | | | |
| AMWAA vs Step Functions | | | AMWAA allows auto-scale w integration w Fargate  Parses the queries and develop execution plans  Suited for managing complex, code-based workflows | | | | | | | | | | Allows you to scale to meet demand  Pay as you go model  Suited for simplifying complex apps |
| Glue Workflows | | | Can create and see complex ETL jobs, which can incorporate multiple crawlers, jobs, and triggers  Can be built from a Glue blueprint or manually build w 1 component at a time | | | | | | | | | | |
| Using Containers in Data Pipelines | | | | Using containers allow you to write in polyglot (any language), be more agile, and scale more easily  Main container types: AWS ECS and EKS (elastic Kubernetes service) | | | | | | | | | |
| ECS orchestrates deployment of containerized apps and mapping of resources. Can launch, manage, and scale as you grow. – 3 layers to ECS: 1) Capacity: infrastructure layer that all your containers run on  2) Controller: deployment layer to manage apps and containers.  3) Provisioning: layer that provide tools to interact w schedulers for deploying and managing containers | | | | | | | | | |
| EKS is a managed service that controls and maintains your Kubernetes control panel  Allows you to create clusters of Fargate containers or EC2 workers to run Kubernetes inside those cluster  More complex to set up. | | | | | | | | | |
| ECS uses AWS orchestration engine  Fine-tune auto scalability  Microservice architecture | | | | | | | | EKS manage your Kubernetes experience  Allows for multi-cloud deployments  Easy integration w AWS services (better security) | |
| Data coming from everywhere | | | | EMR Multi-Source. EMR = AWS big data solution for huge scale data processing. (Uses Hadoop & Spark)  Usually run in a cluster w a Master Node and several Core Nodes and task Nodes  Code Nodes: software components that run tasks and store data within your various systems (HDFS)  Task Nodes [Optional]: Run tasks specifically to utilize that data and push to other sources  For Multi-Source: multiple Master/Primary Node to take data from multiple sources | | | | | | | | | |
| Redshift Multi-Data Warehouse  Requirements: - Connection to a datashare. – Available on (serverless workgroups, ra3.4xl clusters, ra3.16xl clusters). – Metadata discovery. – Encryption  Prerequisites: - Create a Data Warehouse. – Load sample data w Query Editor. – Load data from S3. - SQL  Cons: - Cannot connect via Data API, need datashare. – Unable to see permission granted by their datashare to their user base. – Consumer writing to datashare objs will not trigger an auto analyse option (must run SQL command after adding data to analyze that new data) | | | | | | | | | |
| Optimizing Costs | | | DPUs: make processing jobs easier by providing more units to offload demand on CPUs  Cost per DPU are per hour | | | | | | | | | | |
| Spark is run inside of Glue and processes data in batches. Can run Spark ETL scripts w the job cmd *glueetl*  Glue supports extension for PySpark for scripting ETL jobs | | | | | | | | | | |
| Python shell option in Glue can be cheaper than running Glue on its own  Using a Python shell allows you to build out ETL jobs while being in a Spark environment  Python shell runs inside of Glue w the job cmd *pythonshell*. Only need 1 DPU to get the job running | | | | | | | | | | |
| Python Shell for Glue jobs default allocate 1 DPU  Billed per hour w 1 min minimum  Designed to run light ETL jobs | | | | | | Spark default allocates 10 DPUs  Billed per hour w 1 – 10 min min (depends on version)  Designed for scaling horizontal and or vertical | | | | |
| Infra as Code (IaC) | | Build entire environments at once versus piece by piece. Standardize deployments to autoscale  IaC main options: 1) CloudFormation (AWS built-in option for building environments of any size)  2) Cloud Development Kit (CDK; abstraction layer on top of CloudFormation). 3) Third-party (Terraform) | | | | | | | | | | | |
| 1) CloudFormation - Setup environment via templates written in YAML and JSON.  - Each time you deploy resources, they create a Stack, allowing resources to be connected and built tgt  - Also has Change Sets to identify changes in a template before deploying to avoid drift | | | | | | | | | | | |
| 2) CDK. - Provides way to push builds to CloudFormation. – Can use more common languages like Python, Java. – More freedom in code w Declarative or Imperative statements, i.e. can be abstract w best practices while still being granular. - Also provides library of Constructs (building blocks that contain everything CloudFormation needs to create the component) | | | | | | | | | | | |
| CDK provides 3 diff levels on how code is built out:  1) CloudFormation constructs: same syntax as CloudFormation; explicitly define all aspects of the resource  2) Abstracted constructs: add cloud resources w some aspects abstracted away by CDK  3) Pattern constructs: constructs are pre-built or custom pattern that define multiple resources from 1 or more AWS services | | | | | | | | | | | |
| CloudFormation allows for quick deployments and easy infra replication.  Allows for safe deployment, drift detection, ability to rollback | | | | | | | | CDK geared towards developers (less code, deploy more). Harder to debug  Sends data to CloudFormation to build resource quickly and efficiently | | | |
| SQL | | SQL is used in AWS RDS, AWS RedShift, AWS Athena. New Commands: WHERE … OR … | | | | | | | | | | | |
| SQL Syntax Arguments: ‘$’ = useful when representing string constants inside other constants  ‘^’ = exclusive OR (true only when 1 condition is true, but not both). ‘%’: match seq of ≥ 0 wildcard characters. ‘&’: logical AND to combine operations as well as invoke bitwise (bitwise = assign binary values to your results, translating things to a 0 or 1 for each character) | | | | | | | | | | | |
| Query Optimization: - Use CASE expression to perform aggregation instead of selecting from same table multiple times. – Use INNER joins, not LEFT or OUTER joins. – Avoid using UNION  - Be as specific as possible for conditionals. – Use EXPLAIN to show query execution plan and cost | | | | | | | | | | | |
| SQL PIVOT | | SELECT col1, key1, key2 FROM (SELECT col1, col2 FROM table) AS SourceData  PIVOT (COUNT(col3) FOR col2 IN (key1, key2) ) AS PivotTable; # Essentially pivot longer  At least 1 col to act as a Row identifiers.  1 col as Col identifiers to be used to create new column headers  1 col to contain values to fill pivoted table = values to aggregate based on row and col identifiers | | | | | | | | | | | |
| Tumbling Window | | When a windowed query processes data inside each specific window in a non-overlapping manner  Services that allow for Tumbling Windows:  1) Lambda: can set windows to open up functions to pull data tgt in bundles of 5 MBs w range of 0 – 900s for the window to be open. 2) DynamoDB: when used as a trigger option within Lambda  3) Kinesis: Can ensure only sending data within the tumbling window | | | | | | | | | | | |
| Time-Based Window or Row-based Windows (num of rows). Window types:  1) Stagger windows: aggregate data according to when data arrives. Allows for overlapping windows, reducing late or out of order data  2) Tumbling window: open and close at scheduled intervals, only accepting data at certain times  3) Sliding window: continuously uses a fixed time or row count before closing window | | | | | | | | | | | |
| Connec-ting to SQL DB | | Java Database Connectivity (JDBC). Code <-> JDBC API + JDBC Driver <-> DB. - Types of JDBC Drivers:  1) JDBC-ODBC Bridge Drivers translate and convert Java calls to ODBC function calls  2) Native-API Drivers use client-side libraries of the target DB  3) Network Protocol driver uses middleware to convert JDBC calls into a DB call  4) Database Protocol Drivers / thin drivers changes JDBC calls directly into vendor-specific DB protocol | | | | | | | | | | | |
| Open Database Connectivity (ODBC). App <-> ODBC Driver <-> DB/Data source. – Components of ODBC:  1) Apps process and call ODBC functions then submit the SQL statements  2) Driver manager loads drivers for each app sending a call  3) Drivers handle actual ODBC call and send request to a data source. 4) Data source is accessed | | | | | | | | | | | |
| JDBC can only be programmed in Java  Can be used on any platform  Is object-oriented | | | | | | | | ODBC can be programmed in any language  Can only be used on Windows  Is procedural | | | |
| Data Lifecycle | | Data Sources -> Real-time Ingestion (Kinesis Firehose) or Data Storage (S3) -> Batch Processing (Glue) or ML or Stream Processing (Flink/Kinesis Data Analytics) -> Analytics Data Store (RDS/RedShift) -> Reporting & Dashboards (Quicksight).  – High Level: Ingestion & Transfer -> Security & Storage -> Transformation & Use -> Analyze & Monitor | | | | | | | | | | | |

Storage Gateway: integrates on-prem platforms w S3 by offering file shares that store data as S3 objs. Ideal for connecting legacy apps w/o native S3 support to S3 for analysis

Apache Hadoop DistCp command: transfers data from on-prem Hadoop clusters to S3 buckets using a distributed copy mechanism. Ideal for bulk migration from existing Hadoop environments to S3

AWS Direct Connect: dedicated network connection for secure data transfer from on-prem to S3. Ideal for high-bandwidth, low-latency data transfer for large dataset

AWS Cognito: secure and scalable way to manage user identities, including authentication, authorization and user data storage (profile)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data Store Management | | | | | | | |
| Amazon S3 Security | | order of responsibility on you: Infra services (EC2) , Container svcs (RDS) , Managed svcs (S3, DynamoDB)  We handle Client-Side Encryption. IAM is a Shared responsibility  AWS handle Server-Side Encryption, Network Protection, Platform, OS, Network, Firewall.  1st security layer is Network (VPC). 2nd = Authentication and access control (IAM). 3rd = Encryption | | | | | |
| S3 is accessible over the internet (operates independently of VPCs). Can still select ‘Block all public access’ option. To access when all public block, use:  VPC Endpoints: grants you isolated access by providing a private connection btw your VPC and S3.  - Allows you to access S3 directly from within your VPC w/o traversing the public internet | | | | | |
| Access Control: 1) Identity-Based: AWS IAM policies are attached to IAM users, groups or roles  2) Resource-Based: - Access Control Lists (ACLs) configured on the object/bucket level  - Bucket Policies: JSON-based policies that you attach to control access at bucket level  Policy: Effect (allow/deny), Principal (who), Action (do what), Resource (what object)  3) S3 bucket policies allows to grant access across accts, deny unencrypted uploads, define access based on IP addrs, IAM user/role  4) S3 Access Points: define access policies applying only to data access through that particular endpoint  - Multiple Access Channels (multiple apps/teams need access to diff subset of data in same S3 bucket) | | | | | |
| S3 Server-Side Encryption (SSE)/at rest encryption: in AWS. Either client or AWS encrypts  For SSE: 1) Customer Provided Keys (SSE-C): you use your managed keys during uploading objs to S3  2) AWS Key Management Service (KMS):  a) AWS Owned key (default, AWS owns and manages key, no cost) OR S3 Managed Keys (SSE-S3)  - Each obj encrypted w a unique key. – Uses 256-bit Advanced Encryption Standard (AES)  - When creating an obj by API, include header: *‘s3:x-amz-server-side-encryption’: ‘AES256’*  b) AWS Managed Key (Key stored in your acct/own by you, AWS manages key, AWS KMS charges)  - Track key usage in AWS CloudTrail.  – When creating by API, include headers: *‘s3:x-amz-server-side-encryption’: ‘aws:kms’* &  *‘’x-amz-server-side-encryption-aws-kms-key-id’: ‘KEY\_ID’*  c) Customer Managed Key (You create, own, and manage key, AWS KMS charges)  - When creating by API, include headers: *‘s3:x-amz-server-side-encryption-customer-algorithm’: ‘AES256’* &  *‘’x-amz-server-side-encryption-customer-key’: ‘BASE64\_ENCRYPTION\_KEY’*  Client-Side Encryption: outside AWS. Client in charge of encryption before sending data to AWS  In Transit encryption: S3 encrypts data in transit by default using SSL/TLS protocols to secure connections | | | | | |
| S3 Features | | S3 Versioning. S3 Replication (replicate obj from 1 S3 bucket to another). S3 Event Notifications | | | | | |
| Keeps track of deletions through delete markers. 2 options for configuration:  1) Object Level versioning. 2) Bucket level versioning.  Once enabled, versioning cannot be disabled, only suspended (will not delete previous versions) | | | | | |
| Cross-Region Replication (CRR): for Disaster Recovery (DR), compliance & geo redundancy, reduced latency  Same-Region Replication (SRR): DR within a single-AZ, Testing (btw prod and test), Caching  1) Asynchronous replication: files copied to replica after written to primary bucket (near real-time)  2) Synchronous replication: files copied to replica while being written to primary bucket (real-time)  S3 replication operate asynchronously. To start replication, must enable versioning in src and dest bucket & grant IAM permissions to your S3 bucket. Only **newly** added objects are replicated  To replicate existing objects: use S3 batch replication  Delete markers are not replicated by default. Option to replicate delete markers | | | | | |
| S3 Events: create, remove, restore, replicate objects, expired S3 lifecycle events (auto delete old objs), Transitioned S3 lifecycle events (change storage class), auto archival events from S3 Intelligent-Tiering, tagging objs, PUT ACL objs (changing ACL of objs)  Can filter objs by name, e.g. \*.png. Usually takes a few seconds  Can send notifications to SNS, SQS, Lambda, EventBridge (which can then send to other places) | | | | | |
| Storage Class | | 1) Amazon S3 Standard – General Purpose: Frequently access data w low latency and high throughput  2) Amazon S3 Standard Infrequent Access: Infrequently access data and ok with slightly higher latency, but still need high throughput when accessed  - S3 Standard-Infrequent Access (Standard-IA): 99.9% availability in multiple zones. For DR, backups  - S3 One Zone-Infrequent Access (One Zone-IA): 99.5% availability in single zone. For secondary backup  3) Amazon S3 Glacier: Archive the data w high latency (retrieval times of several hours) and can tolerate low throughput when accessed  - Instant Retrieval: Milliseconds retrieval. For medical images, news media assets, data accessed quarterly  - Flexible Retrieval: Min storage duration of 90 days. Retrieval options: Expedited (1-5 mins), Standard (3-5 hrs), Bulk (5-12 hrs; free)  - Deep Archive: Min storage duration of 180 days. Retrieval options: Standard (12 hrs). Bulk (48 hrs) | | | | | |
| 4) S3 Intelligent-Tiering: auto moves objs btw the access tiers. Incurs small monthly fee  Frequent Access Tier (default). Infrequent Access Tier (objs not access for 30 days). Archive Instant Access Tier (not accessed for 90 days). Archive Access tier (manually set, configurable from 90-730 days).  Deep Archive Access Tier (manually set, configurable from 180-730 days) | | | | | |
| 1) Use S3 Select to access data in most S3 storage class (Standard and Intelligent Tiering)  2) Glacier Select to access Glacier. Both allows simple SQL statements which Amazon performs server-side filtering which is up to 400% faster and 80 % cheaper | | | | | |
| S3 Lifecycle Rules | | Configure when objs are moved btw storage classes and access tiers (e.g. \*.png move to infrequent access)  1) Transaction Actions: when and how objs should be moved from 1 access tier to another  2) Expiration Actions: at what age should objs expire/permanently deleted (for compliance or older version)  As soon as obj becomes eligible for lifecycle action, billing charges apply (but if expiration action supposed to occur, but S3 still has not expire it, won’t charge for storage after expiration time)  - Exception if lifecycle rule is to transition to S3 Intelligent-Tiering storage class (billing only when obj has transitioned to S3 Intelligent-Tiering) | | | | | |
| S3 Analytics | | | Generates CSV reports w recommendations on how to transition objs based on:  - Access patterns for S3 objs (access freq, last access time, total data scanned). – Cost analysis  This feature is not available for objects stored in One-Zone IA or Glacier | | | | |
| EC2 | Rent virtual servers (compute instances) in AWS. Pay for what you use, no commitments. Flexible in growing and shrinking capacity. Minutes to provision and launch instance.  Instance Types depends on : how powerful machine needs to be, how many vCPUs, how much memory | | | | | | |
| General Purpose Instances: - Power = balanced performance of compute, memory and networking  - vCPUs = balanced ratio of vCPUs to memory. – Memory = balanced ratio of memory to vCPUs. – E.g. t3, t3a, t4  Compute Optimized: - Power = high computational power. – vCPUs = higher vCPUs relative to memory  - Memory = sufficient to support most workloads. – E.g. c6g, c5, c5a, c5n  Memory Optimized: - Power = large amt of RAM. – vCPUs = lower num of vCPUs compared to other instances.  – Memory = Highest memory capacities. – E.g. r5, r5a, r5n  Storage Optimized: - Power = high-speed storage. – vCPUs = less vCPUs. - Memory = Moderate to high memory  - Great for low-latency storage for data-intensive workloads. – E.g. I3, I3en, d2, h1  Note EC2 storage is not persistent. Use EBS or EFS to persistently store data beyond lifecycle of EC2 instance | | | | | | |
| Block storage attached to individual EC2 instances. – Instance Store (more like cache, meant to enhance performance) or Elastic Block Storage (EBS) (for persistent storage)  - Types of EBS volumes: 1) General Purpose (gp2) = balanced; for small to medium size DB  2) Provisioned IOPS (io1) = specify consistent level of IOPS; for transactional processing system or real-time analytics. 3) Throughput Optimized (st1) = freq accessed workloads; big data analytics like Hadoop, Spark, date warehouse. 4) Cold HDD (sc1) = infrequent access.  5) Magnetic = low cost storage for large amt of data that is rarely accessed  File storage offers shared file storage, can be mounted to multiple EC2 concurrently. – EFS for Linux or FSx for Windows. Use cases = content sharing, distributing files | | | | | | |
| Pricing Models: 1) On Demand = pay by hour or second, depends on type of instance  2) Reserved = reserves capacity for 1 or 3 years depending on contract for up to 72% discount on hourly charge  3) Spot = purchase unused capacity at discount of up to 90%, not guaranteed  4) Dedicated = physical EC2 server dedicated for you, most expensive | | | | | | |
| Data Formats | | Structured data: follows predefined model or schema, organized into tables, rows and cols. R/s btw data elements clearly defined. Easily queryable  Semi-Structured: falls btw structured and unstructured. Has some organizational elements to provide some hierarchy using tags, keys, attributes. Don’t have rigid schema, can have schema evolution. E.g. JSON, XML  Unstructured: Don’t have a specific schema. Not organized into consistent format. Lacks clear r/s btw elems. Not easily queryable (needs preprocessing). E.g. images, audio, video, social media posts, word doc | | | | | |
| Common Data Formats | | | 1) CSV (comma-separated values): text-based tabular data. Structured data  2) JSON: text-based data organized hierarchically using key-value pairs. Used to configure files and settings OR exchange data btw web servers and browsers, language agnostic  3) Avro: represent data in a way that is easy to serialize (convert to binary) and deserialize. JSON-like syntax + schema defined. Use to achieve a compact binary format (smaller payload) OR with big data processing frameworks (Hadoop, Spark, Kafka), data that have regular schema changes OR apps that require high interoperability btw diff systems while enforcing data type validation  4) Parquet: open-source columnar storage file format optimized for big data analytics. Similar to Avro, but schema and data in 1 file unlike Avro which stores them separately and is row-based  Used parquet for querying OR w big data processing frameworks and Redshift Spectrums OR apps that undergo regular schema changes OR selective reading of specific columns | | | | |
| Transfor-ming Data Formats | | Glue ETL Jobs: 1) Python Shell Jobs. 2) Spark ETL Jobs. 3) Batch & Streaming ETL Jobs using Pyspark or Scala  1) Python Shell Jobs: for simple ETL tasks. Has pre-built libraries for transforming data to other formats, aggregate data, reading data from files  2) Spark ETL Jobs: workloads requiring high compute power. Spark is a distributed computing system. Can perform complex transformations over large-scale data. Filtering, aggregate, joining  3) Batch and Streaming: PySpark or Scala to batch process or stream process (process in micro-batches) | | | | | |
| DynamoDB | | | | NoSQL, fully managed, non-relational db. - Supports key-value and document data (JSON, HTML, XML).  – Don’t support joins or analytical queries. – Access patterns must be known. – Unlimited storage size  - Single-digit millisecond latency at any scale. – Microsecond latency w DynamoDB Accelerator (DAX)  - Infinite scaling, smoothly handle terabytes of data | | | |
| Features: - DynamoDB Global Tables for globally distributed apps  - DynamoDB Streams: captures time-ordered seq of item-level modifications in a DynamoDB table  - Auto distribute data across partitions. – Auto replicate across 3 AZ, providing high availability  - Time to Live (TTL): compares current time to defined TTL attr of an item. If current time > item’s TTL value, item is marked for deletion -> DynamoDB auto removes expired items from tables, LSI (local secondary indexes) and GSI (global secondary indexes) within 48hrs of expiration  - Use filter operations to exclude items marked for deletion but not deleted yet | | | |
| Used for Media and Metadata Stores, Retail and Shopping experiences, Large-scale gaming platforms  Use for OLTP workloads, hierarchical data structure, fluctuating workload, mission-critical apps | | | |
| SQL: Tables -> Rows/records -> Cols/Fields.  PKs can consist of multiple cols (optional) | | | NoSQL: Tables -> Items -> Attributes  - Single item in DynamoDB can’t exceed 400 KB. PKs can consist of 1 or 2 attrs (mandatory) |
| DynamoDB Rate Limits & Throttling | | | | Throughput capacity depends on read and write capacity modes:  1) Provisioned Capacity Mode: provision w capacity units: Read capacity units (RCUs) & WCUs (writes)  - Risk of over or under provisioning (subject to throttling, but can leverage auto scaling)  - More for apps w predictable traffic and capacity needs. – Can switch to provisioned mode any time  - Enable auto scaling and define min and max capacity or disable auto scaling  2) On-demand Capacity Mode: don’t need to specify throughput, charge in terms of Read request units (RRUs) and WRUs  - Auto scales according to demand. – More expensive than provisioned. – Subject to throttling if exceed 2x the previous peak within 30 mins. – More for apps w unpredictable traffic. – Just make API calls  - Can switch to on-demand mode once every 24 hours  Both modes have 40,000 RCUs/RRUs/WCUs/WRUs limit per table | | | |
| Burst capacity: when consumed capacity > provisioned  - When throughput is not fully used, DynamoDB retains up to 5 mins of that unused capacity for later use | | | |
| Exponential backoff is used in retry logic | | | |
| DynamoDB: PartiQL | | | | Query DynamoDB in console/AWS CLI/DynamoDB APIs/NoSQL Workbench using SQL-like syntax. Support most statements like INSERT, UPDATE, SELECT, DELETE. – Can obtain output in Table view format or JSON view  - PartiQL don’t always translate queries to efficient operations | | | |
| Scans: similar to SELECT \* FROM table. *aws dynamodb scan –-table-name*  Expensive as scan entire table even with filter expressions, and can use up provisioned throughput  Queries: find items in table using PK. *aws dynamodb query –- table-name* | | | |
| To prevent PartiQL statements from being translated to scan operations:  - Use AWS IAM to deny *dynamodb:scan* operation in the identity used to run PartiQL statements  - Create secondary indexes and write queries using those indexes  - Monitor performance of queries and analyze any full scans that occurs | | | |
| Redshift Distributn  Styles | | | 1) Even: rows distributed evenly btw cluster nodes regardless of values. - Don’t need to use table for JOINs  2) Key: rows distributed based on 1 col. Identical key values placed on same node  - Need to use table for JOINs  3) ALL: entire table replicated in each node. – Small fact/static tables  4) Auto: Redshift identifies the best-suited distribution style | | | | |
| Redshift Workload Management (WLM) | | | | | Categorize queries and run them in diff queues  Main purpose to prevent long queries from holding up short fast-running queries  WLM set up through params groups. Can have up to 8 queues. – Each queue will have a concurrency level to determine num of concurrent queries | | |
| WLM modes: 1) Auto = comes w default param group. – Redshift manages concurrency level  - Can identify relative importance of queries through setting a priority value (CRITICAL, HIGHEST, HIGH, NORMAL, LOW, LOWEST)  2) Manual = custom param group. – You manage concurrency level (max 50; sum of all concurrency levels for all queues also 50). – Default, Redshift configures 2 queues (1 w concurrency level = 5, and 1 predefined Superuser queue w concurrency = 1) | | |
| Redshift Spectrum & Materialized Views | | | | | Redshift Spectrum query exabytes of data in S3 w/o loading data into redshift. Don’t need to worry about scaling. – But still need a Redshift cluster to interface w Redshift Spectrum (cluster must be in same region as S3 bucket. Cluster will send query to thousands of Redshift Spectrum nodes. Can have more than 1 cluster querying the same data) | | |
| Redshift Cluster -> Spectrum -> IAM -> Data Store -> Spectrum -> Redshift Cluster  - Need to create an external read-only table per dataset in S3: supports SELECT \* INSERT ops. Don’t support UPDATE or DELETE ops  Tables can reside in a data store, such as AWS Glue, Athena, EMR | | |
| Redshift Federated Query: query data across DBs, data warehouse, data lakes  Can use to perform join queries to combine data from Redshift w data from external DBs (S3, RDS, Aurora). – Apply quick transformation on data, eliminating need for ETL pipeline | | |
| Regular Views: virtual table that consists of a saved query. Results aren’t stored in views, only query  Materialized Views: snapshot/physical copy of the result of a query. For predictable/recurring queries  To keep view in sync w source table, can: - turn AUTO REFRESH on to update data each time source table is updated. – Manually issue command (REFRESH MATERIALIZED VIEW view\_name) | | |
| Transfer/  Migrating Data | | | AWS DataSync: Transfer files and objs btw on-prem to storage service like S3. – Constant syncing  AWS Transfer Family: Set of fully managed file transfer services. – Suited for standard file transfer protocols like SFTP, FTPS, or FTP. – Integrates w existing authentication systems like Active Directory/ LDAP | | | | |
| AWS Application Discovery Service (ADS): A component of AWS Migration Hub. – Scans on-prem environment and identifies dependencies btw apps to determine the optimal migration approach  ADS Discovery Types: 1) Agentless Discovery (Remote) = utilize AWS Agentless Discovery Connector to gather info about servers, apps, configs, resource utilization (CPU, memory, disk usage).  – Suited for where installation of additional software (agents) is not possible  2) Agent-based Discovery = Deploys lightweight software/discovery agents. – Continuously monitors and collects more granular and real-time data. – When deep level of control and customization needed | | | | |
| Application Migration Service: Focuses on app-level migration. – Ideal for rehosting apps running on physical servers, VM or other cloud environments to AWS. – Can do Migration or Replication. – Minimized downtime. - Migration lifecycle: Discovery -> Planning -> Testing -> Cutover | | | | |
| AWS Snow Family: Physical devices for migrating large amt of data or network bandwidth can’t handle  1) Snowball Device: ≥ 10 TB to migrate. 2) Snowball Edge: ≥ 10 TB. Have onboard compute resources to process data. 3) Snowmobile: ≥ 10 PB | | | | |
| DB Migration Service (DMS) | | | Migrate on-prem DB to AWS. Can use for discover eligible source DB, migrate source -> DMS -> target DB, consolidate source DBs to target DB. – Source DB remains available during migration  Possible DMS migrations: - into the cloud (AWS/Azure), - From AWS to on-prem. – Btw on-prem instances (through an AWS Cloud setup) | | | | |
| 1) 1 time migration: Once migration is complete, DMS can be stopped, and target DB = primary one  2) Continuous replication: Continually replicates changes made to source to target DB in near real-time | | | | |
| Migration Methods: 1) Full load (used for 1-time migration). – Can result in downtime. – Initial migration OR target table don’t exist or needs to be completely overwritten  2) CDC (Change data capture): only replicate data changes. – Minimal downtime. – Initial load already migrated or restoring from snapshot  3) Full load + CDC: Full load then replicate ongoing changes via CDC | | | | |
| Streaming CDC into S3: 1) On-prem DB -> DB CDC -> AWS DMS -> S3  2) On-prem DB -> DB CDC -> AWS DMS -> Kinesis Data Streams -> Kinesis Data Firehose – (transformation) -> S3 | | | | |
| Migration Types: 1) Homogenous = no need for schema conversion. Btw 2 compatible engines  2) Heterogenous = schema conversion + data migration. Btw 2 non-compatible engines  - Need to do schema conversion w Schema Conversion Tool (SCT): a standalone GUI app | | | | |
| DMS is a server that runs replication software. Runs on EC2 instance = replication instance. – Runs in VPC  You need to create: 1) Replication instance. 2) Source and target connections (using endpoints).  3) Target tables manually or using SCT to create some or all of target tables, indexes, views, triggers, ….  4) DB migration/replication task (identify data to be migrated). 5) Service access role in IAM for DMS to run | | | | |
| Table Mappings: specify r/s btw cols from src to target tables  Transformation Rules: convert data types, perform calculations, altering strings | | | | |
| Components of a Data Catalog | | | | | Data Catalog keeps track of: - where data is store, - what’s in the data, - who uses the data, - data quality, - data lineage. E.g. AWS Glue Data Catalog, Hive Metastore | | |
| Data Catalog components: 1) Metadata Repo (Hive Metastore).  2) Search and discovery = search using DB/table/col names. – Tags & annotations provide additional context for searching  3) Data Lineage = Hive Metastore stores flow of data. FK & metadata r/s shows r/s btw cols in diff tables  4) Data Asset descriptions = table’s purpose, owner, creation data  5) Access and security = access permissions set on table/col level. – Use security labels to categorize data based on sensitivity of confidentiality. Can query permissions using Hive Metastore | | |
| Apache Hive = data warehousing and SQL-like query language tool (HiveQL).  – Create data warehouses on top of large-scale distributed storage systems like Hadoop/S3  - Can install and run Hive on EMR clusters  - Hive Metastore stores metadata about: Hive tables, schema, partition info, storage location  - Can use Glue Data Catalog to act as a Hive Metastore | | |
| Elastic MapReduce (EMR): cloud-based big data processing service (fully managed)  - Distribute and parallelize data processing tasks on EC2 using Hadoop, Spark, …  Src system -> transport using EMRFS (EMR file system) -> EMR Cluster -> nodes/EC2 for transformation OR Data Catalog System (Hive Metastore or Glue Data Catalog) | | |
| Metadata | | | Provides context and info about the data: Location, Schema + Data Types, Data Lineage  Ensures quality and integrity of data: correct data type + version control | | | | |
| Types of Metadata: 1) Structural = organization (lineage, r/s, table/col names, data types, constraints)  2) Descriptive = content and purpose of data (table description/comment, annotations)  3) Administrative = management aspect (ownership, access permissions, versioning)  4) Technical = storage format, serialization/deserialization, indexing | | | | |
| Importance of data lineage: understand data flow, enable impact analysis, compliance, debugging | | | | |
| Data Quality Metrics: 1) Accuracy. 2) Completeness (not null). 3) Consistency (consistent across diff sources or over time). 4) Validity (conform to predefined rules, e.g. positive num). 5) Uniqueness  Benefits of Data Quality Metrics: better decision making, save costs to preprocess, compliance | | | | |
| Collaboration: - annotation and comments.  - Use AWS CloudTrail to monitor API changes/who made changes, including metadata updates and when  - Can use CloudWatch Event rules to monitor changes made to Glue Data Catalog or S3 and trigger notifications via SNS | | | | |
| AWS Glue vs Apache Hive | | AWS Glue -> Crawl Data -> Glue Data Catalog -> Transform -> Load to Target DB  - Fully managed data catalog that auto crawl and organized metadata about datasets  - Uses built-in or customer logic to transform data and load to target | | | | | |
| Apache Hive: Data processing and analytics. Uses Hive-QL to run queries against HDFS or S3 | | | | | |
| AWS fully managed ETL service  AWS settle provisioning, scaling, maintenance  Batch and streaming processing  Translates ETL jobs in AWS  Integrates w AWS services | | | | Runs on Hadoop clusters  Requires setup, config, maintenance  Batch processing  Translates Hive-QL into MapReduce jobs  Integrates w Hadoop system (HDFS, YARN, MapReduce) | |
| Self-Discovering Schemas in AWS | | | | Data Lakes: central storage repo to hold raw data in native format | | | |
| When configuring AWS Glue crawler, can enable Auto Schema Discovery. (use if data changes frequently) OR  Provide Schema Manually. Use if data don’t change frequently or need to enforce some schema | | | |
| Optimization Techniques for improving Query Performance | | | | | 1) Indexing: Find info w/o full table scan | | |
| 2) Partitioning: divide large dataset into smaller subsets called partitions  - Simplifies process of finding items as it reduces amt of data scanned. – Enables parallel processing | | |
| 3) Compression: reduce space to store data. Encode data in more compact representation and decompressed when needed. – Consumes CPU resources to compress and decompress  - E.g. GZIP, ZIP, GZ, RAR or Redshift compression algos (LZOP, BZIP2, GZIP) | | |
| Schema Evolution and Updating Data Catalogs | | | To update data catalogs: 1) Manually through - AWS Glue Console. – Manual re-run crawler  2) Programmatically through – Glue APIs, - AWS SDK, - ETL scripts  3) Automated through – schedule Glue crawlers to run periodically, - AWS CloudFormation | | | | |
| Schema Evolution: 1) Schemas (modifying schema don’t change underlying data)  2) Partitions (add/delete partition by adding/removing key value pairs OR update partition metadata)  3) Tables (create/delete tables, modify table properties, change table ownership)  4) Indexes (modify indexes to improve query performance) | | | | |
| Deleting cols = cannot query col even though underlying data is not deleted  Adding cols = can query col. For existing data, can put null/default value, populate using existing values  Changing data type = interpret col according to its new data type. May result in data type conversion errors | | | | |
| Transformation via ETL scripts:  1) Custom transformations/user-defined functions for format conversion / complex transformations. - Package script in library. – Upload package to S3. – Create Glue job referencing the S3 path  2) Bundled Transformations/predefined functions. – AWS Glue Console or API. – Simple transformations like map, join, filter, aggregate, drop column, ResolveChoice  3) ML Transformations: deduplication, records linkage (link records from diff data sources that refer to same entity), data quality enhancement  E.g. FindMatches = deduplicate by linking similar records across datasets (even with diff in attribute) | | | | |
| ResolveChoice = resolve ambiguities in a DynamicFrame, i.e. data inconsistencies  - Identify inconsistencies and resolves it. – Specify rules to handle | | | | |
| DataFrame = data structure composed of records. - Batch processing. - Defined schema (need to manually update schema if schema changes). - Accessed through APIs by frameworks like Spark  DynamicFrame = data structure composed of DynamicRecords. – Handle semi-structured data. – Support nested data formats. – Support auto schema evolution. – Introduced by AWS Glue | | | | |

DynamoDB: NoSQL DB ideal for high-throughput, random-access workloads. Scales effortlessly

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Operations and Support | | | | | |
| Data APIs | For DBs: can use JDBC or ODBC. For Data Providers (like salesforce, SAP, slack): use Amazon AppFlow | | | | |
| Amazon AppFlow: fully managed integration service to transfer data from SaaS apps to AWS  - can filter, enrich and validate incoming data to ensure data quality  - Built-in integrations for Google Analytics, Facebook Ads, ServiceNow, and many others | | | | |
| AWS Data Exchange: 3rd party data API marketplace  Data providers can host their data APIs on Data Exchange, and consumers can subscribe and use the data APIs | | | | |
| EMR | - Map: splitting huge amt of data into smaller chunks for parallel processing and each chunk is transformed using its own source of compute power. - Reduce: reassemble the separate chunk.  EMR uses Apache Hadoop: Hive or Spark for data prep (ETL), Spark ML library or Presto for data analysis  But instead of managing the infra yourself, EMR help manage the cluster for running these big data ops  - Only choose size of ERM cluster to use, simplifying ETL for large amts of data in and out of AWS data stores | | | | |
| For performance, EMR clusters always exist in a single AZ  To have some sort of “high availability”: can choose to have 2 extra Primary Node on standby  Core Nodes run tasks and coordinate distributed file system  Optional Task Nodes for parallel processing. Can be used by Core Nodes to faster process data | | | | |
| Default file system is HDFS. (best for intermediate job-flow steps). Data on cluster is not persistent  2nd option is EMRFS: HDFS implementation integrated with S3 (best for persistent data)  3rd option is local file system: ideal for storing temp data like caches or buffers | | | | |
| EMR Serverless: eliminate cluster config and scaling management  - Just define runtime and jobs, AWS will provision workers when jobs execute | | | | |
| Ephemeral Clusters: more cost effective for periodic or infrequent, scheduled jobs  Tears down when job is complete | | | Long-Standing Clusters: for jobs that involve indefinite streams of data or close-to-constant use or don’t want to wait for cluster to start up | |
| EMR Hive Metastore: stores info about data, like table schemas, partitions and data types  - Built in feature to use Glue Data Catalog OR use external Hive Metastore (stored on Amazon Aurora or RDS)  - Choose Glue Data Catalog to integrate well w other AWS services like Athena or Redshift Spectrum  - Use external Hive Metastore if need expose to 3rd party apps like Apache Ranger or Apache Atlas | | | | |
| EMR use Spark to define jobs that transform data  Can leverage other open-source tools (Hadoop, Hive)  Superior price-performance (especially for huge data) | | | Glue use Spark to define jobs that transform data  More built-in features for data discovery, connectors, job monitoring and orchestration  Superior operational efficiency (not worry about infra) | |
| Glue DataBrew | | Visual data prep tool in AWS console designed to enable data analysts to filter, validate & transform data. – Low code or no code solution for data validation.  – Over 250 transformations to clean and normalize data. – Can visually map data lineage  Data Lake/Warehouse -> Glue DataBrew -> S3 (for staging).  – Data analyst define data quality rule to validate incoming data. If fail check, will alert analyst, analyst can then correct the data  - Analyst can define and reuse transformations | | | |
| Common Transformations: - remove/replace missing values. – combine datasets. – create cols. – filter data. – label mapping (string map to numerical value). – aggregate data | | | |
| Apache Offerings | | | Apache Flink: data analytics for streaming data, streaming ETL, cts metric generation, real-time analytics, interactive analysis of data streams  AWS has Amazon Managed Service for Apache Flink (formerly Kinesis Data analytics) | | |
| Apache Airflow: orchestrate and monitor workflows  AWS has Amazon Managed Workflows for Apache Airflow (Amazon MWAA): if already using airflow  Use AWS Step Functions: more robust support of other AWS services and less operational overhead  AWS Glue Workflows: specifically for defining ETL data pipelines  Amazon EventBridge: for event processing, receive and send events, schedule events | | |
| Apache Kafka: distributed event streaming platform optimized for real-time data streams  Amazon Managed Streaming Service for Apache Kafka (Amazon MSK): if already using Kafka  Amazon Kinesis: if streaming data and building on AWS from scratch | | |
| Apache Hive: built on top of Hadoop for warehousing. Interact w petabytes of data using SQL  Amazon EMR: to leverage Hive | | |
| Apache Spark: large scale data processing in parallel  Amazon EMR: can use Spark OR AWS Glue: Apache Spark used to define ETL jobs | | |
| EventBridge | | | Process & schedule asynchronous events in AWS. Ingest & deliver events, schedule automated actions  Events can be published from AWS services or API calls. Need to define an Event Bus, rules and Targets  Use EventBridge only if need handle events from many sources. Else, simple S3 event trigger for Lambda | | |
| Athena | | | Directly query data in S3 buckets using SQL interface  Fully managed, serverless, interactive query service. Run ad-hoc SQL queries on petabytes of data  - Underlying uses open-source Trino query engine.  – Can create tables and schemas using Apache Hive Data Definition Language (DDL)  - Optionally run ad-hoc Spark apps | | |
| Supports JSON, CSV, Apache ORC, Apache Avro, Apache Parquet | | |
| For query optimization: 1) partition data: partitions act as virtual cols and group related data tgt  - Data WON’T populate to partitions by default. For Hive Style partitions, run MSCK REPAIR TABLE cmd. For non-Hive partitions, run ALTER TABLE ADD PARTITION cmd  Hive style partitioning has a key-value format w equal sign btw. E.g. year=1999/month=03/day=26/…  2) compress data: Athena supports compression formats like gzip, snappy, zesty, ….  - Compressed JSON and CSV files are not splitable, and cannot be processed in parallel  - Compressed Parquet and ORC files are splitable due to segmentation and metadata  3) Use columnar file format: Parquet or ORC | | |
| Amazon QuickSight | | Create and share visualization from many diff sources  On provisioning QuickSight, you create a serverless app where you can add users and share visualizations  In AWS acct, have IAM users or IAM roles. In QuickSight, permissions are granted separately  QuickSight itself must be granted IAM permission to your data source  - To connect QuickSight to Athena, QuickSight also need IAM permissions to any associated S3 buckets | | | |
| Dashboards = collection of visualizations. Give diff users diff permissions to view diff dashboards | | | |
| Data sources can be from S3, QuickSight, Aurora, Redshift, OpenSearch, RDS, 3rd party | | | |
| QuickSight for dashboards w highly interactive viz  Optimized for BI data viz  Huge selection of viz  Users are separate from AWS users | | | CloudWatch for dashboards w interactive viz  Optimized for cloud resource monitoring & automation  Limited selection of viz  Manage AWS permissions for users |
| Visualizing Data | | Tools: Glue Studio, Athena, Redshift Query Editor, QuickSight (Left to Right: data prep -> data analysis) | | | |
| Glue Studio to filter and transform data directly from AWS console  Using visual editor in console, can define Glue pipeline | | | |
| Athena has a viz tool called Athena Visual Query Analysis to see query performance (optimized or not) | | | |
| Redshift Query Editor: create simple viz w/o creating an entire QuickSight | | | |
| - Line charts: track trends over time. – Bar charts: comparisons btw categories  - Pie charts: compare % of a whole. – Scatter plots/bubble charts compare 2/3 dims  - Funnel charts: stages through a process. – Histograms: dist of values for a given dim  - Gauges display a single metric (like KPI) | | | |
| Amazon Macie | | Simplify monitoring process of all S3 buckets involved in data pipeline  Continuously scans S3 buckets for sensitive data such as PII/specific IP  Configure to send alert or trigger automated action | | | |
| Multi-Account Amazon Macie: activate Macie in each acct and then centrally managed from a single acct | | | |
| CloudWatch logs | | | Collect logs from diff sources like EC2, CloudTrail, apps logs and each placed in a log group  Most AWS service can have logging enabled.  – Can also send CloudWatch logs via CloudWatch agent (installed on EC2 or on-prem)  - Can send CloudWatch logs via AWS CLI or programmatically w AWS APIs | | |
| - Can enable Anomaly Detection on a log group. – Use CloudWatch Logs Insights to query log data  - Export logs to S3 for analysis. – Stream logs to Kinesis or OpenSearch | | |
| Optionally create 1 log anomaly detector for each log group. Uses ML and pattern recognition  - Dynamically assigns priority to each detected anomaly based on deviation from baseline | | |
| To query CloudWatch Logs Insights: - CloudWatch Logs Insights Query OR  – Natural Language Query: use AI to translate query into correct format  Queries can be executed on ≤ 50 diff log groups. Results can be downloaded and further analyzed | | |
| To export logs to S3, either manual or programmatically export on periodic basis  From S3, can use Athena or QuickSight to visualize logs. – S3 also cheaper to store logs long term | | |
| OpenSearch: best option for near real time monitoring and search ability of CloudWatch logs  To react to certain patterns in real time: use subscription filter to trigger an event -> automated action | | |

By default, Step Functions fails the execution upon encountering an error in a state

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data Security and Governance | | | | | | |
| IAM Foundations | | IAM (Identity and Access Management) -> Network -> Authentication -> Encryption -> Monitor & Log  Principal = users/group. Resource = S3, RDS. IAM grant principal access to resources  Human users can access resource via AWS Management Console or AWS CLI  Apps and other AWS services use AWS API | | | | |
| Either Identity-Based policy or Resource-Based must grant access, and both must not deny access  1) Check any Policy have Deny rule (yes, reject). 2) Check any Policy have Allow rule (yes, allow)  IAM policy stored in JSON format | | | | |
| IAM Users = map permission sets to human users OR Apply permission sets to IAM groups & assign users to those groups. Both have issue of leaking access & secret key | | | | |
| IAM Identity Center: don’t map permission sets directly to users or user groups but map to IAM roles  Users then must use SSO -> IAM Identity Center -> Grant IAM role to user for this session  IAM roles also used by AWS services and apps to temporarily assume privileges to access resources | | | | |
| Accessing Private Data Stores | | Networking scenarios: VPC w multiple private subnets. Not in VPC: S3, IAM, Lambda, …  E.g. VPC contains private subnet 1 (RDS w its security group) + private subnet 2 (EC2 w its security group). At each subnet level, have its own NACL | | | | |
| Security Group (SG) applied at instance level  Define inbound and outbound rules  Only allow rules. Traffic implicitly denied if not allowed  Order of rules don’t matter  Stateful: traffic allowed in/out always allowed out/in | | | | NACL applied at subnet level  Define inbound and outbound rules  Define allow and deny rules separately  Rules evaluated in order  Stateless: traffic allowed in might not allow out |
| Troubleshooting Connectivity btw EC2 and RDS:  1) Ensure route table is correctly routing traffic within your VPC  2) Ensure DB traffic is allowed at NACL level within you VPC  3) Ensure there is an inbound rule on RDS instance SG allowing request traffic from EC2 instance  4) Ensure there is an outbound rule on EC2 instance SG allowing request traffic to RDS instance | | | | |
| For the private EC2 to access S3: use Gateway Endpoints = private endpoints for S3 or DynamoDB that permit private traffic withing the same region (don’t traverse public internet; more secure + cheaper) | | | | |
| For connecting Lambda to private RDS: need configure lambda to have access to 1 of your private VPC  - Lambda functions w/o access to your VPC cannot connect to private RDS instances  - Lambda function still sit within AWS Managed Subnets/VPC but given an Elastic Network Interface (ENI) at VPC level if configure lambda to have access. ENI also provisioned on RDS VPC  - By allowing Lambda to have access to your private VPC, the Lambda function also become private and cannot access public internet | | | | |
| For connecting Lambda to S3: just need IAM role since both are in public VPC | | | | |
| RDS Authentication Options | | | 1) IAM Authentication. 2) Kerberos Authentication (AD: Authenticated Directory, from Microsoft)  3) Password Authentication | | | |
| 1) IAM Authentication only available for MariaDB, MySQL, PostgreSQL on RDS  Using IAM Authentication may limit your max connections per second | | | |
| 3) Pw authentication, storing in plain text / env variables not safe, use secrets manager | | | |
| Intro to Secrets Manager and Parameter Store | Parameter Store: store, optionally encrypt and programmatically access key-value params  Secrets Manager: managed encryption and rotation of stored secret key-value pairs | | | | | |
| Param store stores key-value pairs  Optionally enable encryption  Manually change secrets  1st 10,000 params are free  Best for centralizing variables | | | | Secrets Manager stores key-value pairs  Encryption always enabled  Auto rotates secrets  $0.40 per secret per month  Provides highest security | |
| 1) Define secret’s name & region in initialization. 2) Initialize session & client. 3) Retrieve secret at runtime | | | | | |
| Securing Data using Lake Formation | Lake Formation creates a single entry point for users to access all resources in your data lake (otherwise, have to grant access to the diff service individually in your data lake like S3, Glue Data Catalog, Athena) | | | | | |
| IAM to define permissible actions -> Lake Formation to determine what data those actions can be taken on  Lake Formation achieves its fine-grained permissions using RDBMS E.g. IAM (AthenaFullAccess + Inline Policy for Glue & Lake Formation read) -> Lake Formation (fine-grained permissions: restrict access to specific table) | | | | | |
| Cross-Account Access: grant data lake permissions to specific cross-acct principals  Lake Formation Event Logging: integrates w CloudTrail to log data actions & permissions granting API event | | | | | |
| Lake Formation Granular Permissions | | On setting up data lake w Lake Formation, you create a data catalog (pull data from diff sources and break it into DBs and tables)  For more fine-grained control (besides restrict to particular table), can use filters (defined at table level) | | | | |
| Filter 1) Column Masking (Exclude Columns: col\_name). 2) Row Masking (col\_name = value)  Can combine col and row filtering to filter cells and provide cell-level security | | | | |
| Protecting PII and Masking Cols | | S3 (data lake): use Amazon Macie to scan buckets in diff accts at the organizational level | | | | |
| Redshift (data warehouse): can mask whole cols and rows  1) To mask cols: Grant SELECT and UPDATE privileges by table and col  2) To mask rows: Create an RLS (Row Level Security) Policy.  E.g. CREATE RLS\_POLICY policy\_name WITH (…) USING (col = value)  3) Dynamic Data Masking (DDM): partial masking or full masking. - Masking occurs at query time | | | | |
| Data Encryption Options | | S3 Encryption | | | | |
| Redshift Encryption: can use KMS AWS-Managed Key OR KMS Customer Managed Key  - Need to encrypt at launch or migrate to an encrypted cluster by changing the Encrypt database option | | | | |
| Glue Encryption: 1) Ensure encrypted in-transit from data source to Glue  2) Glue Data Catalog and S3 need to be encrypted using KMS or S3-managed keys  To access encrypted Glue Data: need read/write permissions + encrypt/decrypt permissions | | | | |
| Tracking API Calls via CloudTrail | Preparing Logs for Audit: Track API calls via CloudTrail  - When any principal access your AWS acct or resources, an API call is made and recorded in CloudTrail  From CloudTrail: - can store logs in S3, then Athena to query OR  - store logs in a CloudTrail Lake (easier to store and save queries, and can filter by certain types of events) | | | | | |
| To analyze logs from multiple accts: - can have CloudTrail in each individual acct  - Best practice to have a central account to store logs from diff accts | | | | | |
| CloudWatch Alarms and Logs | | - Near real-time analysis. – Can subscribe services to log events at acct or log group level | | | | |
| Can attach metric filters to log groups & define alarms based on the metrics -> SNS or lambda  E.g. receive email notification when Data Definition Language (DDL) cmd issued in Redshift | | | | |
| CloudTrail to monitor API activity on resources  Track and audit actions  Store logs in S3 & monitor events w CloudWatch | | CloudWatch to monitor resource & apps  Monitor performance and app logs  Store logs w CloudWatch or S3 | | |
| By default, CloudWatch logs stored indefinitely. So might be cheaper to store in S3 | | | | |
| Sharing Data across Redshift Clusters | | To share data across redshift clusters: use Datashare  Datashare can consist of ≥ 1 DB, table, udf, materialized view or schema  - Can share across cluster types, AZs or regions, AWS accounts  - Datashares provide live, up-to-date data across Amazon Redshift clusters for read-only access  *CREATE DATASHARE datashare\_name;*  *ALTER DATASHARE datashare\_name ADD SCHEMA schema\_name;*  *ALTER DATASHARE datashare\_name ADD TABLE table\_name;*  *GRANT USAGE ON DATASHARE datashare\_name TO NAMESPACE redshift\_namespace\_name;* | | | | |

concurrency = (average requests/second) \* (average request duration)

SageMaker Data Wrangler: data prep + feature engineering + data analysis for ML workflow

AWS Config monitors and records configuration of AWS resources continuously. Can identify changes and deviations from desired configs, helping to detect misconfigurations. Can create config snapshot and store on S3

To use AWS Config, need to set up: - S3 bucket to store config snapshots and history. – SNS topic for receiving notifications about config changes. – IAM role w permissions for Config to access resources

CloudTrail Lake provides a centralized repo for storing CloudTrail logs (capture API calls). Within CloudTrail Lake, can create “data stores” to organize and managed these events according to specific categories and needs. Logs are auto converted into Apache ORC (Optimized Row Columnar) format.

- Allows flexible retention of event data for up to 10 years

- Integrates w Glue Data Catalog and Athena for SQL querying

- To ensure only relevant data is stored, can use advanced event selectors to filter and capture most impt info

Use Redshift condition keys w specific tags and values to grant access to Redshift resources on a granular level of detail

- *aws:RequestTag* = require users to include specific tag key and value when creating resource

- *aws:ResourceTag* = restricts user access to resources based on existing tags w specific key-value pairs

- *aws:TagKeys* = compares tag keys in a request w those specified in the policy

AWS Outposts: separate environment from the shared AWS infra, offering stricter isolation for sensitive data

AWS European Sovereign Cloud: independent cloud designed for European customers.