**1. Kinesis Data Streams (KDS)**

**Overview**

* **Purpose**: Kinesis Data Streams is a real-time data streaming service that allows applications to process and analyze data as it arrives.
* **Use Cases**: Real-time analytics, streaming log files, website clickstreams, social media feeds.

**Key Properties**

* **Shards**: The unit of scalability. Each shard can handle:
  + **Write**: Up to 1 MB/s or 1,000 records per second.
  + **Read**: 2 MB/s.
* **Retention Period**: Data can be retained up to 7 days (24 hours by default).
* **Partition Key**: Used by producers to send records to specific shards, maintaining order within a shard.

**Capacity Modes**

1. **On-Demand Mode**
   * Automatically scales to match demand.
   * Ideal for unpredictable or spiky workloads.
   * **Example**: A news app with varying daily traffic might use on-demand to handle unexpected spikes.
2. **Provisioned Mode**
   * Manually set shard count based on expected throughput.
   * Lower cost than on-demand if traffic is predictable.
   * **Example**: A fixed-capacity IoT app streaming data from thousands of sensors can set shards for consistent usage.

**Security**

* **Encryption**: Uses AWS KMS for encryption at rest.
* **Access Control**: IAM roles and policies for fine-grained access.

**Producers**

* **Kinesis Producer Library (KPL)**, **Kinesis Agent**, or **AWS SDK** to send data to KDS.
* **Example**: A social media application uses the KPL to send posts in real-time to KDS.

**Consumers**

1. **Enhanced Fan-Out** (EFO): Dedicated throughput per consumer, 2 MB/s per shard.
   * **Example**: Ideal for applications where multiple consumers need the same data at high speed, like analytics and monitoring systems.
2. **Standard**: Shared 2 MB/s per shard for all consumers.
   * **Example**: Suitable for single-consumer applications that prioritize cost savings.

**Key Operations**

* **Splitting Shards**: Increases capacity by dividing a shard into two.
* **Adding Shards**: Increases overall capacity by adding new shards.
* **Merging Shards**: Reduces costs by combining shards.
* **Example**: A company might split or merge shards based on demand patterns, scaling up for the holiday season.

**Handling Duplicates**

* **Producers**: Implement retries with unique IDs to avoid duplicate data.
* **Consumers**: Use sequence numbers for deduplication.
  + **Example**: A logging application using sequence numbers to ensure no log entry is duplicated in the database.

**2. Kinesis Data Firehose**

**Overview**

* **Purpose**: Data streaming service designed to load data into destinations like S3, Redshift, or Elasticsearch with optional transformations.
* **Buffer Settings**: Controls how data is batched.
  + **Buffer Interval**: Default 60 seconds.
  + **Buffer Size**: Default 1 MB.

**Transformation Example**

* Use **AWS Lambda** to format data.
  + **Example**: A financial app uses Firehose with Lambda to sanitize transaction data before storing it in S3.

**Kinesis Data Streams vs. Firehose**

* **KDS**: Real-time data stream for analytics or immediate processing.
* **Firehose**: Ingests data to storage services with optional transformation.
  + **Example**: Use KDS for real-time dashboards and Firehose for archiving data to S3.

**3. Amazon CloudWatch Logs Subscription Filters**

* **Purpose**: Enable real-time processing of log data.
* **Example**: Filter out ERROR messages from application logs and route them to an SNS topic for alerting.

**4. Amazon Simple Queue Service (SQS)**

**Standard Queue**

* **Attributes**: Unlimited throughput, at-least-once delivery, potential duplicates.
* **Example**: E-commerce application sends order messages to a Standard Queue to ensure high throughput.

**FIFO Queue**

* **Attributes**: Ordered processing, exactly-once delivery.
* **Example**: Financial application processing transactions in a specific order.

**5. AWS IoT Core**

**Components**

* **IoT Thing**: Represents the physical device.
* **Device Gateway**: Securely connects IoT devices to AWS.
* **Rules Engine**: Routes messages based on conditions to AWS services.
* **Device Shadow**: Keeps device state synchronized with the cloud.

**AWS IoT Greengrass Example**

* **Example**: Deploy AWS Lambda to run on IoT devices for local processing, reducing latency and internet dependency in a factory.

**6. AWS Database Migration Service (DMS)**

**Migration Types**

* **Homogeneous**: Migrate within the same database engine.
  + **Example**: MySQL to RDS MySQL.
* **Heterogeneous**: Migrate between different engines, often using **Schema Conversion Tool (SCT)**.
  + **Example**: Oracle to Amazon Aurora PostgreSQL using SCT to translate schema.

**Direct Connect**

* **Private Connection** to AWS.
* **Example**: DX for secure and high-speed transfer for a bank migrating its on-premise database to AWS.

**7. AWS Snow Family**

**Devices**

1. **Snowcone**: Small (8 TB), edge computing capability.
2. **Snowball Edge**: Storage up to 80 TB, compute for real-time data processing.
3. **Snowmobile**: Petabyte-scale data migration by truck.

**Use Cases**

* **Data Migration**: Use Snowball to transfer large datasets physically to avoid internet bandwidth constraints.
* **Edge Processing**: Perform pre-processing on Snowball Edge before uploading data to AWS.
  + **Example**: Use Snowball Edge for real-time video analysis on-site, reducing data sent to AWS.

**8. AWS OpsHub**

* **Purpose**: GUI for managing Snow Family devices.
  + **Example**: Set up and manage Snowball Edge for data migration without needing command-line tools.

**9. Amazon Managed Streaming for Apache Kafka (MSK)**

**Components**

* **Producers**: Send messages to Kafka topics.
  + **Example**: A weather station sends temperature data to a Kafka topic.
* **Consumers**: Subscribe to topics and process messages.
* **MSK Connect**: Integrates MSK with other data sources.
  + **Example**: Integrate MSK with S3 for real-time backup of Kafka topics.

**Security**

* **TLS Encryption**: For in-transit data protection.
* **IAM**: Enforces secure access policies.
  + **Example**: Use TLS encryption to secure data exchanged between Kafka producers and consumers.

**Monitoring**

* **CloudWatch**: Monitors Kafka metrics, such as throughput, latency, and consumer lag.
  + **Example**: Track consumer lag in a real-time monitoring application to ensure timely processing.

### 1. ****Kinesis Data Streams (KDS) vs. Kinesis Data Firehose****

| **Feature** | **Kinesis Data Streams (KDS)** | **Kinesis Data Firehose** |
| --- | --- | --- |
| **Purpose** | Real-time data streaming for analytics and processing | Real-time data delivery to destinations with optional transformation |
| **Data Processing** | Requires custom consumers for processing data | Built-in integration with Lambda for transformations |
| **Latency** | Sub-second latency for real-time use cases | Near real-time (delivers data in batches) |
| **Data Retention** | Configurable up to 7 days | Not designed for long-term data retention |
| **Use Case** | Real-time dashboards, anomaly detection | Data archiving, ETL (Extract, Transform, Load) |
| **Example** | Streaming stock market data for real-time analysis | Delivering IoT sensor data to S3 for storage |

### 2. ****Kinesis Data Streams vs. Amazon SQS****

| **Feature** | **Kinesis Data Streams** | **Amazon SQS** |
| --- | --- | --- |
| **Message Ordering** | Guarantees ordering within a shard | FIFO queues offer ordering (standard queues do not) |
| **Throughput** | Highly scalable with multiple shards | Limited throughput per queue, though scalable |
| **Data Retention** | Up to 7 days | Up to 14 days for FIFO queues |
| **Primary Use Case** | Real-time data streaming with analytics or processing | Asynchronous task queues, decoupling of components |
| **Consumer Type** | Multiple consumers, including Enhanced Fan-Out for high-throughput | Single consumer or limited multiple consumer support |
| **Example** | Real-time monitoring for IoT devices | Queueing up tasks for batch processing |

### 3. ****SQS Standard Queue vs. FIFO Queue****

| **Feature** | **SQS Standard Queue** | **SQS FIFO Queue** |
| --- | --- | --- |
| **Message Ordering** | Not guaranteed (best-effort ordering) | Guaranteed ordering |
| **Deduplication** | No deduplication (duplicates possible) | Supports deduplication (exactly-once processing) |
| **Throughput** | Nearly unlimited, best for high-throughput, non-ordered tasks | Limited throughput (300 messages per second with batching) |
| **Primary Use Case** | High-throughput applications where order is not critical | Transaction processing, event-driven workflows |
| **Example** | Distributing jobs to multiple servers in parallel | Processing bank transactions in order |

### 5. ****AWS Snow Family (Snowcone, Snowball Edge, Snowmobile)****

| **Feature** | **Snowcone** | **Snowball Edge** | **Snowmobile** |
| --- | --- | --- | --- |
| **Capacity** | Up to 8 TB | Up to 80 TB | Exabyte-scale |
| **Primary Use** | Small data collection and transport | Data migration with some edge computing capability | Large-scale data migration |
| **Edge Computing** | Limited compute | Robust computing (supports EC2 instances) | No edge computing |
| **Use Cases** | Remote locations, fieldwork | Data processing and transfer from enterprise sites | Data center migration |
| **Example** | Collecting research data from remote sites | Video rendering at a remote production site | Moving an entire data center to AWS |

### 8. ****MSK (Amazon Managed Streaming for Apache Kafka) vs. Kinesis Data Streams (KDS)****

| **Feature** | **Amazon MSK** | **Kinesis Data Streams** |
| --- | --- | --- |
| **Protocol** | Apache Kafka protocol (Kafka client libraries) | Proprietary AWS API |
| **Data Retention** | Configurable by Kafka (can be indefinite) | Up to 7 days |
| **Latency** | Low latency, tuned for streaming analytics | Low latency, suited for real-time processing |
| **Ecosystem Compatibility** | Wide Kafka ecosystem compatibility | Primarily AWS-compatible services |
| **Primary Use Case** | Complex data processing pipelines needing open-source compatibility | Real-time data ingestion and analytics within AWS |
| **Example** | Using Kafka for cross-cloud event streaming | Real-time log processing for analytics dashboard |

**AWS Storage & Amazon S3 Cheat Sheet**

**1. S3 Overview**

* **Buckets**: Containers for storing objects; globally unique within AWS.
* **Keys**: Unique identifier for objects within a bucket.
* **Size Limit**: Individual objects can be up to 5 TB.
* **Multipart Upload**: For files larger than 5 GB, enables uploading in parts.
* **Metadata**: Store additional info about objects (e.g., tags, expiration).

**2. S3 Consistency Model**

* **Read-after-Write**: Strong consistency for new objects.
* **Eventual Consistency**: For overwrites and deletes (read may reflect stale data briefly).

**3. S3 Storage Classes**

* **S3 Standard**: High durability, low latency, frequent access.
  + **Use Case**: Websites, mobile applications.
* **S3 Standard-IA (Infrequent Access)**: Lower cost, retrieval fee, suitable for less-frequent access.
  + **Use Case**: Backup and disaster recovery.
* **S3 One Zone-IA**: Cheaper, infrequent access, stored in one AZ.
  + **Use Case**: Storing secondary backup copies.
* **S3 Glacier Flexible Retrieval**: Archival with flexible retrieval time (minutes to hours).
  + **Use Case**: Archived media assets.
* **S3 Glacier Deep Archive**: Lowest cost for long-term archiving, retrieval in hours.
  + **Use Case**: Regulatory and compliance data.
* **S3 Intelligent-Tiering**: Automatically moves objects to lower-cost storage tiers based on access.
  + **Use Case**: Unpredictable data access patterns.
* **S3 Glacier Instant Retrieval**: Low-cost archive, instant retrieval.
  + **Use Case**: Data requiring infrequent but immediate access.

**4. Lifecycle Rules**

* Automates transitions between storage classes or deletes data based on specified conditions.

**5. S3 Versioning**

* Retains multiple versions of objects to prevent accidental deletion.

**6. S3 Replication**

* **Cross-Region Replication (CRR)**: Replicate objects to a different region.
* **Same-Region Replication (SRR)**: Replicate objects within the same region.
* **Use Case**: Disaster recovery, compliance.

**7. S3 Performance**

* **Baseline Performance**: 3,500 PUT and 5,500 GET requests per second per prefix.
* **S3 Transfer Acceleration**: Faster global uploads using CloudFront edge locations.

**8. S3 Security**

* **Encryption Options**:
  + **S3 Managed Keys (SSE-S3)**.
  + **KMS Managed Keys (SSE-KMS)** - subject to KMS limits.
  + **Customer-Provided Keys (SSE-C)**.
  + **Client-Side Encryption**.
* **Encryption in Transit**: SSL/TLS for data transfers.
* **Security Policies**:
  + **User-Based** (IAM Policies).
  + **Resource-Based** (Bucket Policies, ACLs).
  + **Public Access Block**: Prevents public access at the bucket/account level.
* **Networking**: VPC endpoint for secure access to S3 from within a VPC.
* **Logging & Auditing**: Access logs to monitor requests and activity.

**9. S3 Advanced Features**

* **S3 Select & Glacier Select**: Query and retrieve specific data from objects.
* **S3 Event Notifications**: Trigger events on actions (e.g., Lambda trigger on upload).

**AWS DynamoDB Cheat Sheet**

**1. Overview and Use Cases**

* **Purpose**: Fully managed NoSQL database.
* **Use Cases**: E-commerce, gaming, IoT applications.
* **Advantages**: Scalability, low-latency, highly available.
* **Limitations**: Limited aggregation and complex query support.

**2. Keys in DynamoDB**

* **Primary Key**: Uniquely identifies each item.
* **Partition Key**: Distributes items across partitions.
* **Sort Key**: Allows sorting within a partition.
* **Anti-Patterns**: Avoid large items, frequent updates to a single partition key, and complex joins.

**3. Capacity Units**

* **Write Capacity Units (WCU)**: 1 WCU = 1 KB per second write.
* **Read Capacity Units (RCU)**:
  + **Strongly Consistent Read**: 1 RCU = 4 KB read per second.
  + **Eventually Consistent Read**: Doubles read throughput at reduced consistency.
* **Capacity Calculation Example**:
  + For 10 reads of 8 KB with strong consistency: 10 \* 2 RCUs = 20 RCUs.

**4. Data Operations**

* **Writing Data**: Single-item writes, batch writes.
* **Reading Data**: Single-item, multi-item reads.
* **Deleting Data**: Single or batch item deletions.
* **Throttling**: Requests exceeding capacity limits are throttled; managed with retries.

**5. Indexes**

* **Local Secondary Index (LSI)**: Supports different sort key on the same partition key.
* **Global Secondary Index (GSI)**: Supports queries on different partition and sort keys.
* **Use Case**: Flexible querying based on different attributes.

**6. PartiQL for DynamoDB**

* SQL-compatible querying language, simplifies querying and data manipulation.

**7. DynamoDB Accelerator (DAX)**

* **Purpose**: In-memory caching to reduce read latency.
* **DAX vs. Elasticache**: DAX is specific to DynamoDB, while Elasticache is general-purpose caching.
* **Use Case**: High-read, low-latency applications.

**8. DynamoDB Streams**

* **Purpose**: Tracks changes (insert, update, delete) in a DynamoDB table.
* **Use Case**: Event-driven applications and triggering AWS Lambda on data changes.

**9. DynamoDB Triggers**

* **Purpose**: Automatically triggers Lambda functions based on stream events.
* **Use Case**: Real-time data processing, audit logging.

**10. Time-to-Live (TTL)**

* Automatically deletes expired items to save on storage costs.
* **Use Case**: Session expiration data, temporary logs.

**11. Large Objects Pattern**

* Store large objects in **S3**; use DynamoDB for metadata and indexing.
* **Example**: Store media files in S3, metadata (e.g., file path, type) in DynamoDB.

**12. Security and Backup**

* **IAM Policies**: Grant user and application access.
* **Encryption**: Server-side encryption using AWS KMS.
* **Backups**: Point-in-time recovery (PITR) for data integrity.