**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 |