AWS Data Analytics Specialty

- AWS Partner practice exams
- Official AWS exam guide
- Official AWS exam sample questions
- Tutorialsdojo AWS Analytics cheatsheets
- Tutorialsdojo exam study path #TODO

- AWS Data Analytics Specialty
  - Data migration to AWS
  - Kinesis
    - Data stream
    - Firehose lab
      - Direct put
      - Kinesis Agent
      - Data stream
  - SQS
  - MSK
  - Data pipeline
  - IoT
  - Dynamo DB
    - query vs. scan

Characteristics	Data warehouse	Data lake
Data	Relational from transactional systems, operational databases, and line of business applications	Non-relational and relational from IoT devices, websites, mobile apps, social media, and corporate applications
Schema	Designed prior to implementation (schema-on-write)	Written at the time of analysis (schema-on-read)
Price/ performance	Fastest query results using higher cost storage	Query results getting faster using low-cost storage
Data quality	Highly curated data that serves as the central version of the truth	Any data, which may or may not be curated (e.g., raw data)
Users	Business analysts	Data scientists, data developers, and business analysts (using curated data)
Analytics	Batch reporting, BI, and visualizations	Machine learning, predictive analytics, data discovery, and profiling.

Data migration to AWS

- Storage gateway: hybrid storage that connects on-prem to AWS. ideal for backup, bursting, tiering, migration

- Kinesis firehose: capture, transform and load streaming data into s3 for use with business intelligence and analytics tools
- Data sync: 5x faster file transfers than open source tools, good for migration data into EFS or moving between cloud file systems
- S3 transfer acceleration: fast transfers in and out of s3, ideal when working with long geographic distances
- Direct connect: private connection between on-prem and aws with dedicated fiber optic, no ISP provider required. increased bandwidth
    - dedicated connection: 1Gbps, until 100Gbps
    - hosted connection: 50Mbps, until 10GBps if you order from approved aws direct connect partners
- Snow family: physical pre-packaged hardware box that gets delivered to your on-prem system
    - Snowcone: 2GB box, lightweight device used for edge computing, storage and data transfer. up to 8TB of usage store, use it offline or connect via internet with aws datasync to send data (agent is preinstalled in snowcone)
    - Snowball
    - Snowball edge: storage optimized with ec2 compute functionality. also available: compute optimized and compute optimized with GPU. Gets send from AWS to your edge device
    - Snowmobile: for huge data transfers,they send a truck instead of a box

Kinesis

Streaming service:

The source (mobile, metering, click streams, sensors, logs...) send to data stream producers, which can be SDK, kinesis producer library, kinesis agent, 3rd part libraries like spark, log4j, kafka etc.

Kinesis types:

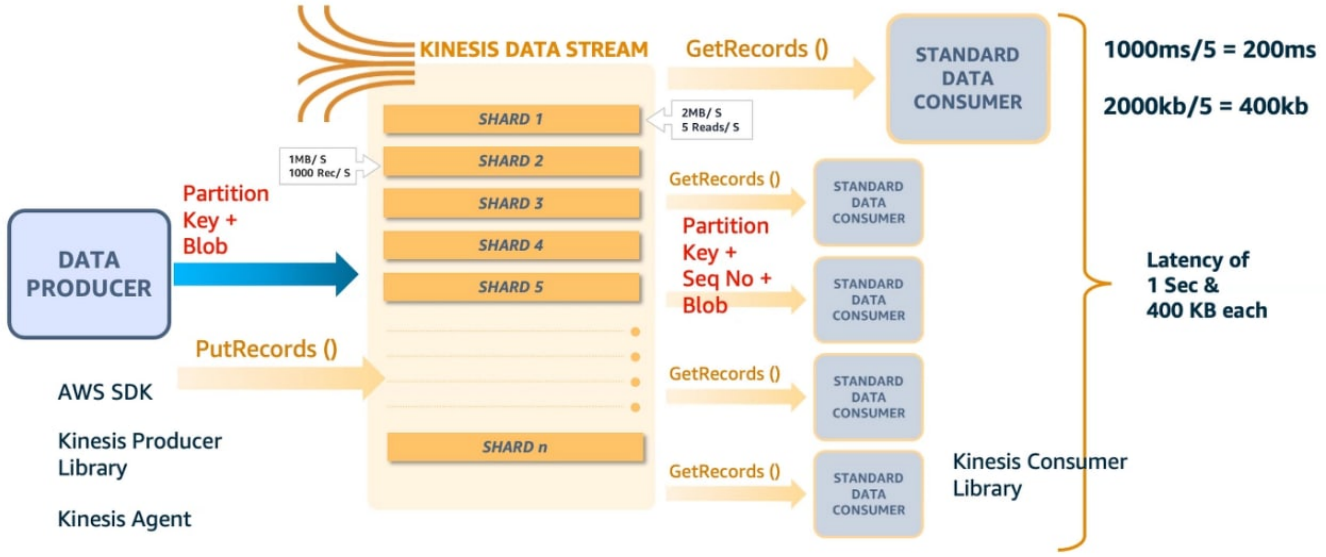
- **Kinesis Stream**: is real-time, 200ms latency. provides storage. good for low-latency reqs
- **Kinesis Firehose**: near real-time, good to just *ingest*/collect data for instance: iot, clickstream analytics, log analytics, security monitoring
    - Easiest way to dump streaming data into aws
    - Provides no storage
    - Minimum latency 60s, max size 1KB (#TODO check this)
    - built-in integration with s3 and other services
    - built-in lambda capability for data transform
- **Kinesis data analytics**: real-time analytics on the streaming data. not ingesting, but analytics
    - SQL: query and analyze streaming data
    - Apache Flink: stateful stream processing
- **Video streams**: stream videos inside aws

Stream workflow:

1. Ingest stream
2. Store stream
3. Process stream
4. Send to final storage

- S3: buffer size range is from 1MB to 128MB. buffer interval is from 60s to 90s. firehose can raise buffer size dynamically. data delivery failures, auto retries up to 24h
- Elasticsearch: same as for s3. custom retry duration up to 7200s, moves skipped files to s3 and provides a manifest file for manual entry
- Redshift, depends on redshift to finish the COPY command. firehose issues a new copy command automatically. for data delivery failures, same as elasticsearch

Data stream



A stream is an ordered (fifo) stream of data, composed of shards, in which each record is uniquely identifiable.

The user has to design a partition key and throughput, according to that shards get allocated. A bad partition key would mean that certain shards are overloaded or hot.

The stream has limits about how many MB/s, and how many records/s per shard. Example: the stream can generate 2MB/s, or 5reads/s per shard. If we have 5 consumers, then each consumers has 500kb/s read capacity.

For standard consumer, average latency is 200ms. With **enhanced fanout**, you get a dedicated throughput of up to 2MB of data/s/shard and average latency of 70ms.

> Errors and their meanings

- ProvisionedThroughputExceedException errors, partition key is well designed but increase shard (scaling), retried with backoff
- RecordMaxBufferTime error, increase batch efficiency by delay
- ExpiredIteratorException KCL error, increase WCU of Dynamodb

Firehose lab

Direct put

Create Firehose delivery stream. The source can be data stream or direct put. Create a name for the stream.

Direct PUT is a method to send data directly from the clients to Kinesis Data Firehose.

In destination, put S3 and select a bucket. In the bucket prefix box, write:

`data/webaccess/year=!{timestamp:yyyy}/month=!{timestamp:MM}/day=!{timestamp:dd}/`

In S3 bucket error output prefix, write:

`error/webaccess/!{firehose:error-output-type}/year=!{timestamp:yyyy}/month=!{timestamp:MM}/day=!{timestamp:dd}/`

In buffer interval, write 60s. This is how often data is sent. the higher interval allows more time to collect data and the size of data is bigger. lower interval, sends data more frequently.

Compression for data records can be Disabled, gzip, snappy, zip or hadoop-compatible snappy.

```
firehose = boto3.client('firehose')
firehose.put_record(
    DeliveryStreamName="my-stream",
    Record={ 'Data': msg }
)
```

Kinesis Agent

Another data source:

Amazon Kinesis Agent is a standalone Java software application. The agent continuously monitors a set of files and sends new data to your Kinesis Data Firehose delivery stream. The agent handles file rotation, checkpointing, and retry upon failures. It delivers all of your data in a reliable, timely, and simple manner. It also emits Amazon CloudWatch metrics to help you better monitor and troubleshoot the streaming process.

With a python script, you can generate app logs, the Agent will constantly monitor the log and transmit new log entries to delivery stream. incoming log entries will be stored into s3.

Install kinesis agent:

```
sudo yum install –y aws-kinesis-agent
```

Create a config file for the Agent, get the role ARN created to allow kinesis agent access to Firehose

`aws iam get-role --role-name FH-KinesisAgentFirehoseRole | grep Arn`

Write into agent.json

```
{
    "assumeRoleARN": "arn:aws:iam::xxxxxxxxxx:role/FH-KinesisAgentFirehoseRole",
    "flows": [{
        "filePattern": "/tmp/api.log*",
        "deliveryStream": "my-FH-Stream-Agent"
```

```
        }]
}
```

This json should be in folder: `/etc/aws-kinesis/agent.json`. Copy it there if necessary. Then run the agent:

```
sudo service aws-kinesis-agent start
```

In a python script in cloud9, write logs into /tmp/api.log. Run the log-generating script, which prints logs which are saved using the logging library.

In the Agent Firehose role, update trust relationship and update the ARN to: AWS": "arn:aws:iam::xxxxxxxxxxxx:role/service-role/AWSCloud9SSMAccessRole" so that we can run it from cloud9

Data stream

KDS is a massively scalable and durable real-time data streaming service. KDS can continuously capture gigabytes of data per second from hundreds of thousands of sources such as website clickstreams, database event streams, financial transactions, social media feeds, IT logs, and location-tracking events.

A typical use case for Firehose is to capture incoming streams of data from Kinesis Data Stream. Kinesis Data Generator can send data to Kinesis Data Stream and capture them on S3 with Firehose.

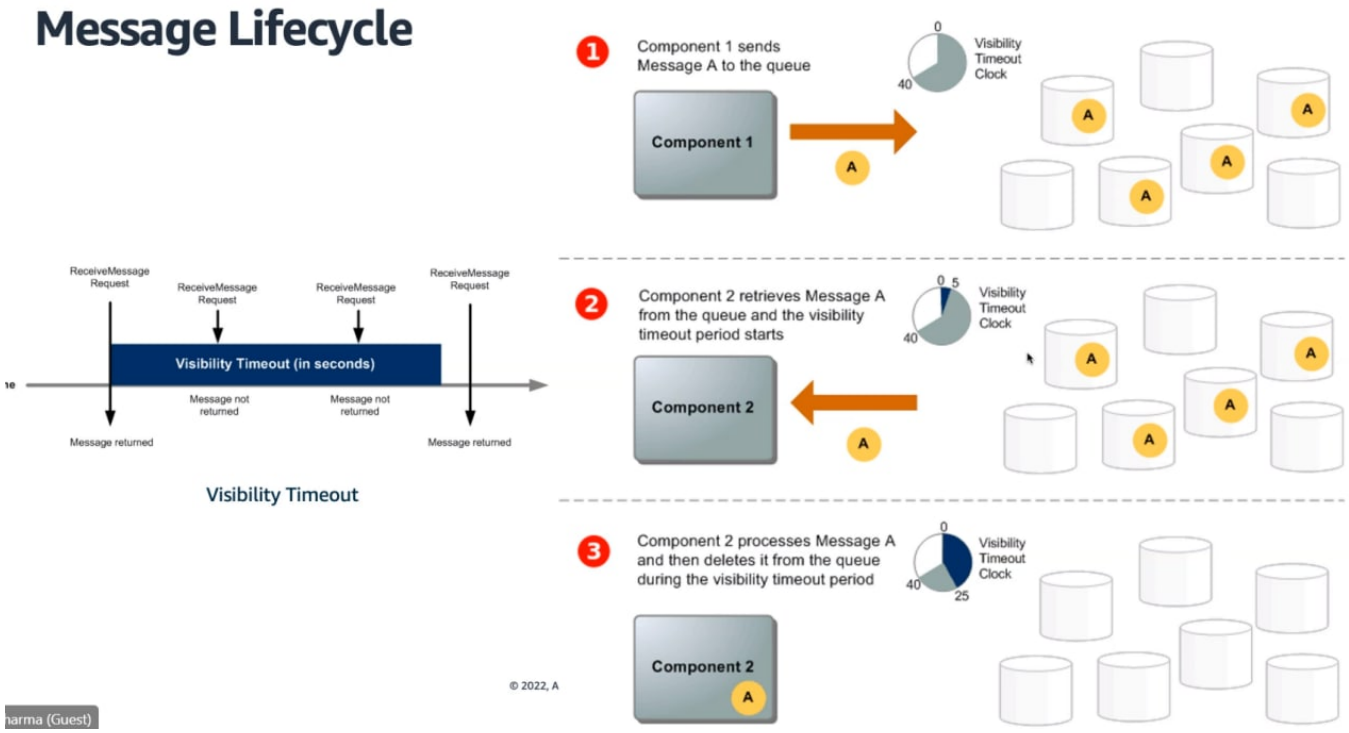
Create data stream, you can choose on-demand or provisioned capacity, and in provisioned you can choose how many shards. Then it shows you the total data stream capacity, for 1 provisioned shard, write capacity is 1MB/s and 1krecords/s and 2MB/s read capacity.

Create firehose delivery stream, source KDS, choose which KDS, destination S3, write the bucket prefix and bucket error output prefix.

In data generator we can create data. In the stack used by the workshop #TODO the cloudformation stack outputs the data generator url, user and password. There we can log in and generate data.

You can then monitor the stream in data stream and also in firehose

SQS



Secure, durable and available hosted queue to integrate and decouple distributed components. Can be another ingestion service

- Standard:
  - unlimited throughput
  - at-least once delivery
  - best effort ordering
- fifo:
  - high throughput upto 3k messages/s, per API method (with matching) or up to 3k API calls/s per API method (without batching)
  - Exactly once processing, no duplicates
  - First in first out messages delivery

Data retention is 3 days default, max 14 days. Low latency, <10ms on publich and receive. message size is 256kb max.

> SQS vs. kinesis

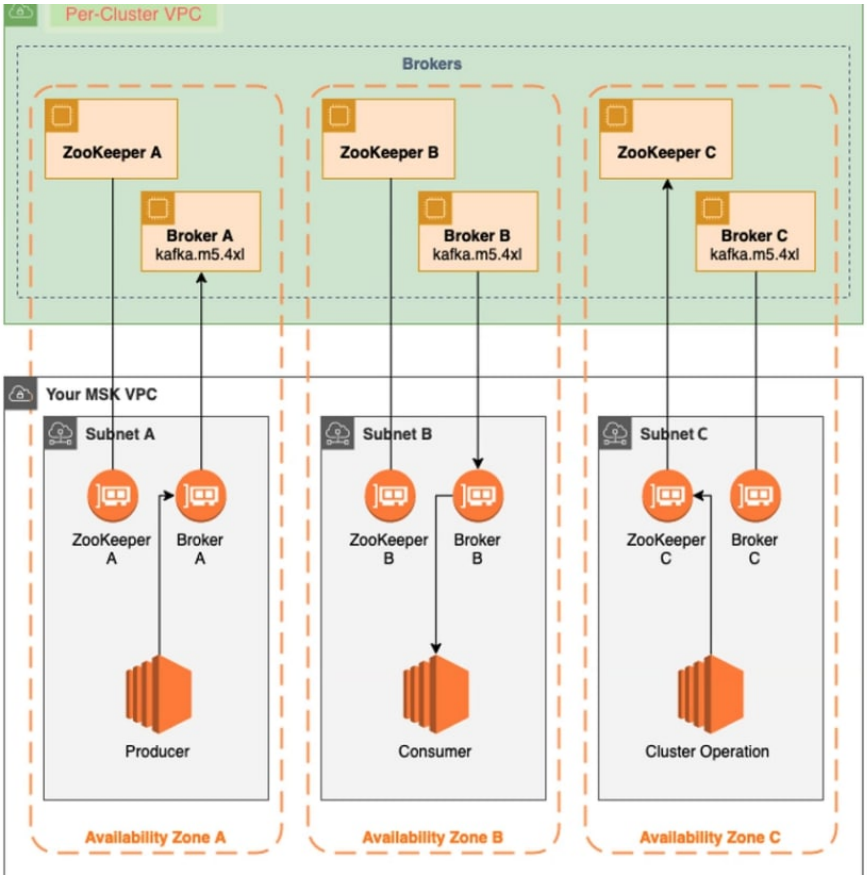
- SQS for order, image processing. good for decoupling front and backend, and it can only received by one consumer. highly scalable, data deleted after read, capable of delaying messages
- Kinesis streams for fast log and data intake and processing, real-time metrics and reporting, real-time data analytics, complex stream processing. read and replays records in same order, many apps can read from the same kinesis data stream, 1MB payload, data deleted after retention period, ordering of records is preserved at shard level, provisioned vs. on-demand mode with replay capability

MSK

Managed Streaming for Apache Kafka is a service that uses fully managed Apache Kafka to ingest and process streaming data in real time. It's an alternative to kinesis, default message size is 1MB but it can be configured to send larger messages, like 10MB.

The MSK cluster is composed of keeper and broker, broker is where you keep partitions. in kinesis stream we have streams and shards, in kafka we have topics and partitions.

Amazon Managed Streaming for Apache Kafka (Amazon MSK)



Kinesis Streams vs Amazon MSK

Amazon Kinesis Streams	Amazon MSK
1 MB message size hard limit	1 MB default message size, can configure for higher (ex. 10 MB)
Streams and Shards	Topics and Partitions
Easy to merge or split shards	You can only add partitions to topics
By default mTLS not there	Supports mTLS
Security using IAM for authentication and authorization	3 options for authentication and authorization • Mutual TLS for Authentication and Kafka ACLs for Authorization • User id password for Authentication and Kafka ACLs for Authorization • IAM access control for both Authentication and Authorization

MSK serverless: for intermittent non-constant workloads, no need to manage or scale cluster capacity. For constant workloads, use provisioned

security:

- Optional in flight encryption using TLS between brokers, and between clients and brokers
- KMS for EBS for at encryption rest
- Security groups for clients to ensure network security

Authentication and authorization:

- Mutual TLS for authentication and kafka ACL for authorization
- User id password for authe and kafka acl for autho
- iam access control for both authe and autho

Monitoring:

- Cloudwatch basic monitoring for cluster and broker
- Cludwatch enhanced monit. for broker metrics
- cloudw topic level monitoring with enhanced metrics
- open source monitoring, like prometheus
- broker log delivery to cloudwatch, s3 or kinesis data streams

Pricing:

You are charged for the following:

- Every Apache Kafka broker instance
- The amount of storage you provide in your cluster

MSK Serverless charges you for cluster, partition, and storage

kafka connect works with other 3rd party services, we can use plugin. you can deploy any kafka connect connectors to MSK connect as plugin like S3, redshift, openserach.

Source connectors can be used to import data from external systems into your topics. with sink connectors, you can eport data from your topics to external systems.

amazon managed streaming for apache kafka (cluster) can only span 1 region, in several AZs.

more info: https://docs.aws.amazon.com/msk/latest/developerguide/what-is-msk.html

Data pipeline

Managed ETL service for scheduling regular data movement and processing. Integrated with on-prem and cloud.

dependency of tasks, waits for the previous tasks.

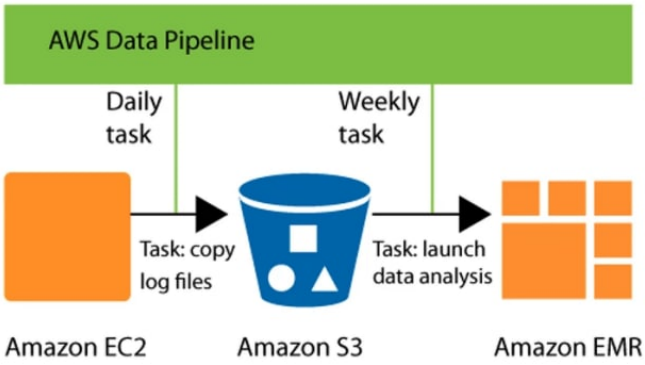
A pipeline definition specifies the business logic of your data management. From the definition, DP determines the tasks, schedules them and assigns them to task runners.

A pipeline schedules and runs tasks by creating EC2 instances to perform the defined work activities. Task Runner polls for tasks and then performs those tasks. For example, Task Runner could copy log files to S3 and launch EMR clusters. Task Runner is installed and runs automatically on resources created by your pipeline definitions. You can write a custom task runner application, or you can use the Task Runner application that is provided by Data Pipeline.

If a task is not completed successfully, Data Pipeline retries the task according to your instructions and, if necessary, reassigns it to another task runner. If the task fails repeatedly, you can configure the pipeline to notify you.

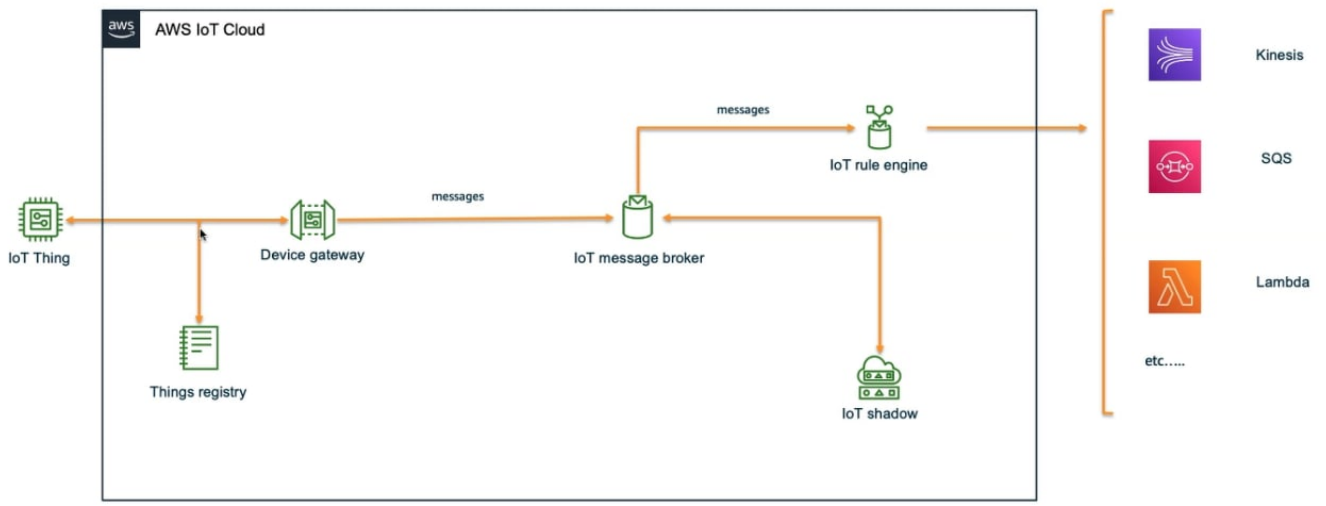
Key points

- ❑ Cross region replication
- ❑ Pre-built templates
- ❑ Pipeline is a container that contains
  - ❑ Data nodes: destination of data
  - ❑ Activities e.g. copy
  - ❑ Pre-conditions: readiness check
  - ❑ Schedules
- ❑ Runs on EC2 or EMR clusters
  - ❑ Provisioned and terminated automatically
- ❑ Works with Amazon DynamoDB, Amazon RDS, redshift and Amazon S3 for storage purpose



IoT

- Device gateway: entry point for IoT devices to AWS
- Message broker: pub/sub, messages published to topics, forwards messages to all the clients connected to the topic
- Thing registry: represents all connected devices represented by their ID: supports metada for each device, can group devices together
- Device shadow: JSON document to represent state of a connected thing
- Rules engine: when a rule is triggered, an action is taken e.g. write data to Kinesis, SQS or so. IAM role needed
- Greengrass: bring the compute layer to the device directly



#TODO add more info

Dynamo DB

- 1 read request unit = 1 strongly consistent read request, or 2 eventually consistent read requests, for an item up to 4KB in size. The total number of read request units required depends on the item size, and whether you want an eventually consistent or strongly consistent read
- One write request unit represents one write for an item up to 1KB in size.

- Eventually consistent: response might not reflect the results of a recently completed write operation. response might include some stale data
- Strongly consistent: returns a response with the most up-to-date data
    - might not be available if there's a network delay or outage
    - may have higher latency
    - not supported on global secondary indexes
    - uses more throughput capacity

partitions

automatically stores data in partitions. a partition is an allocation of storage for a table backed by SSDs and automatically replicated across multiple AZ. dynamodb is optimized for uniform distribution of items across a table's partitions, no matter how many partitions there may be.

Limits: 10GB data (if lsi, otherwise no limit), 3000 RCU and 1000WCU.

number of partitions is a function of size and performance.

partition key for example is animalType, then it creates a hash and with it creates a partition key.

we can have primary key and sort key

secondary index is a data structure that contains a subset of attributes from a table, it allows efficient access to data with attributes other than the primary key. every secondary index is associated with exactly one table, from which it obtains its data. Secondary index is automatically maintained by dynamodb

two types:

- global secondary index
- local secondary index

main difference is that for lsi, the primary key of the lsi must be composite (partition key and sort key), and for gsi it can be simple or composite

sort key uses two attributes together to uniquely identify an item. within unordered hash index, data is arranged by the sort key. no limit on the number of items per partition key, except if you have local secondary index

TODO understand lsi vs. gsi

if we put animaltype as partition key and we have a lot of dogs, all dogs end up in one partition -> hot partition. data should be mapped into all partitions.

number of partitions is the maximum between size and performance.

if our size is 100GB and we want to have 10GB in each partition (which is the maximum we can have), we can have 10 partitions performance = ceiling (desired_RCU/3000 + desired_WCU/1000). for 10GB, we want an RCU of 10k and WCU of 3k. so performance = ceiling(10k/3k + 3k/1k) = ceiling(3.3 + 3) = ceiling(6.3) = 7 number of partitions = max(10, 7) = 10

allocated reads and writes are distributed to all partitions.

- key selection is critical
    - many distinct values (aitio guid, randomize)
    - uniform write pattern across partition keys
    - uniform temporal write pattern across keys
    - don't mix hot and cold keys within a table, create a new table
- Each partition key is limited to:
    - 10GB of data, 3k RCU and 1k WCU

performance by indices

- large number:
    - as compared to your wcu/rcu allocations
    - example: thousands of iot devices sending millions of requests
    - then you need autoscaling
- for reads:
    - gsi-s have their own rcu and wcu values
    - they use alternative keys
    - keep all the key selection best practices in mind
    - use caching with dax, elasticache, cloudfront etc
- for writes:
    - large number of unique keys
    - space out in time to avoid maxing out wcu
    - use SQS queue
- alternatives:
    - use burst capacity but this is 300s only, not a recommended approach
    - auto scaling, but should not be used to cover up bad key selection

try to not have a few random large requests or large scan operations. better is to have frequent requests in bursts, and even better is to have even distribution of requests and size.

query vs. scan

scan is for the whole table or secondary index, it filters out values to prpovide the results you want. A scan operation performs eventually consistent reads by default, and it can return up to 1MB (one page) of data

A single scan request can consume (1MB page size / 4KB item size) / 2 (eventuall consistent reads) = 128 read operations. If you want strongly consistent reads, the scan operation would consume twice as much provisioned throughput: 256 read operations.

techniques to reduce scanning is to use Limit, or create smaller tables.

use parallel scans, multiple threads scanning in parallel, but can quickly consume all of your table's provisioned read capacity.

query find sitems based on primary key values, you can query by partition key and the sort key

- you must specify the partition key name and value as an equality condition
- for sort key, you can use equality, less than, greater than, between and begins_with

a query operation can retrieve max 1MB data, and you can have non-unique data for your partition key.

query operations can consume read capacity units but they are the fastest and most efficient way to read data. dynamodb calculates the number of read capacity units consumed based on **item size**, not on the amount of data returned to an app. for that, the #capacity units consumed is the same whether you request all the attributes (default) or just some, using a projection expression.

---

adaptive capacity is enabled bad default, saves bad design from throttling and you should use this as a feedback to fix table design.

dynamodb streams, you can create a new record whenever a new item is created, and you can send it to lambda, or sns.

---

s3

add info from other notes