AWS Certified Machine Learning – Speciality Examination (MLS-C01)

Curriculam

- Data Engineering (20%)
- Exploratory Data Analysis (24%)
- Modeling (36%)
- Implementation and Operations (20%)

Data Engineering

- Storage Solutions
  - S3 Data Lakes
  - DynamoDB
- Transformation
  - Glue
  - Glue ETL

Data Engineering

- Streaming
  - Kinesis
  - Kinesis Video Streams
- Workflow Management Tools
  - Data Piplelines
  - AWS Batch
  - Step Functions

Exploratory Data Analysis

- Data Science
  - scikit-learn
  - Data Distributions
  - Trends and Seasonality
- Analysis Tools
  - Athena
  - Quicksight
  - Elastic Map Reduce (EMR)
  - Apache Spark

Exploratory Data Analysis

- Feature Engineering
  - Imputation methods
  - Outliers
  - Binning/Categorizing Data
  - Log transforms
  - One-hot encoding
  - Scaling and Normalization

Modeling

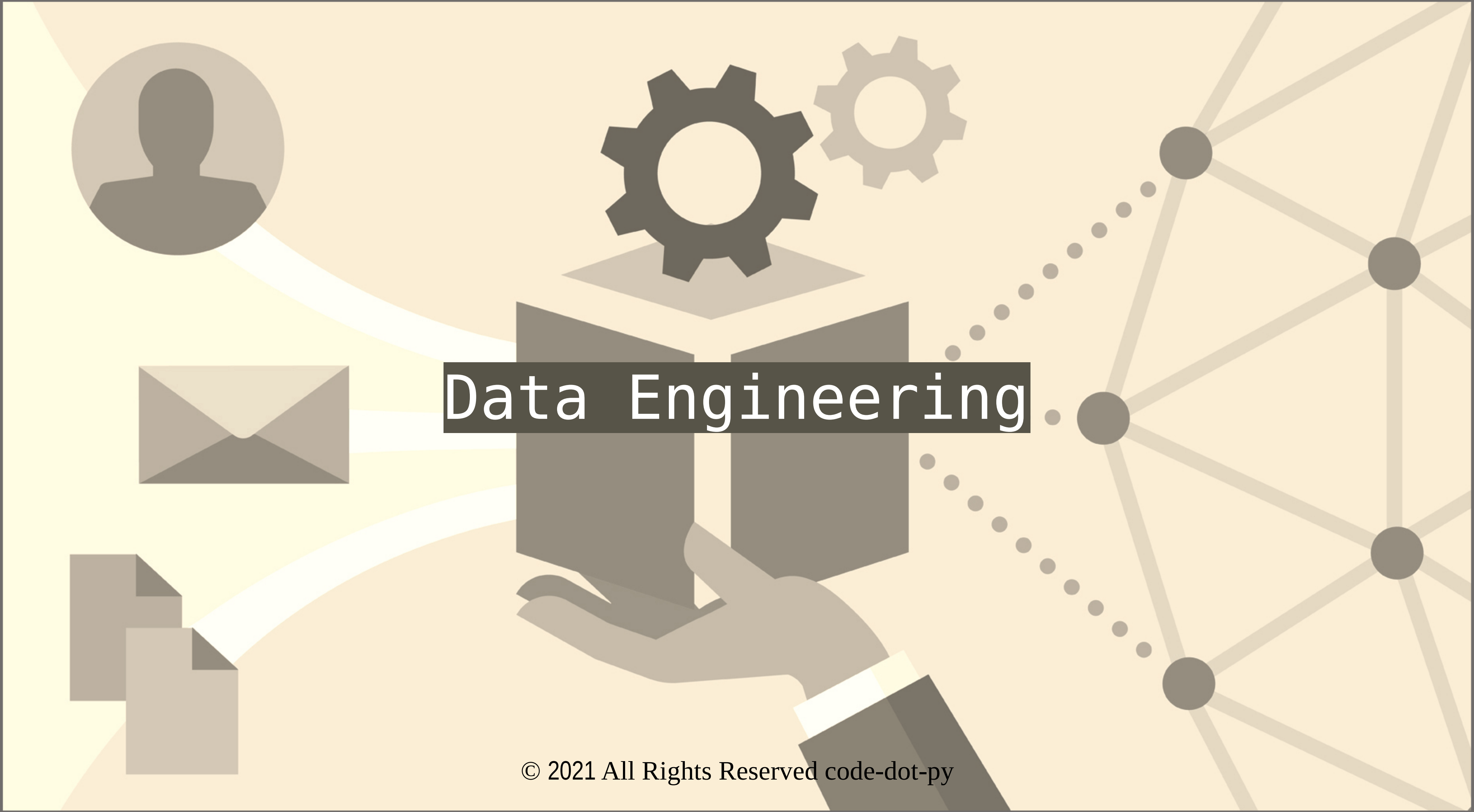
- Deep Learning
  - Multi-layer Perceptrons (MLPs)
  - Convolutional Neural Networks (CNNs)
  - Recurrent Neural Networks (RNNs)
  - ANN – Tuning and Regularization Techniques
- SageMaker
  - Architecture
  - Built-in Algorithms
  - Automatic Model Tuning
  - SageMaker Integration with other services - Spark

Modeling

- High-level AI Services
  - Comprehend
  - Translate
  - Polly
  - Transcribe
  - Lex
  - Rekognition
  - Additional Services – Personalize, Forecast, Textract etc
  - DeepLens
- Evaluating and Tuning
  - Confusion Matrix
  - RMSE
  - Precision and Recall
  - F1 Score
  - ROC / AUC

Implementation and Operations

- Sagemaker Operations
  - Using containers
  - Security with SageMaker
  - Choosing instance types
  - A/B testing
  - Tensorflow integration
  - SageMaker Neo and GreenGrass
  - SageMaker Pipes
  - Elastic Inference
  - Inference Pipelines



Data Engineering

AWS S3 Overview

- S3 allows for storing objects (files) in buckets (directories)
- Buckets must have a globally unique name
- The full path of the objects is called ‘Key’. Example:
  - <bucketname>/<filename>.txt
  - <bucketname>/<foldername>/<filename>.txt
- The maximum object size that can be stored: 5TB

AWS S3 for Machine Learning

- Backbone for many AWS ML services (Ex: SageMaker)
- Core service for Data Lake
  - Infinite size, no provisioning
  - 99.999999999% durability
  - S3 allows for decoupling (segregating) storage for all the compute based services. Examples:
    - EC2, Athena, Redshift, Rekognition, Glue
- Centralized Architecture – all the data at the same place
- Object Storage – supports any file format
- Common formats for ML – CSV, JSON, Parquet, ORC, Avro, Protobuf

AWS S3 Data Partitioning

- Pattern for speeding up range queries (Eg: AWS Athena)
- Partitioning Examples:
  - By Date: s3://<bucketname>/<dataset>/year/month/day/hour/<datafile>.csv
  - By Product: s3://<bucketname>/<dataset>/product-id/<datafile>.csv
- We should choose the partitioning type based on use case
- Some tools like Kinesis and Glue can help with partitioning

AWS S3 Storage Tiers

- Amazon S3 Standard – General Purpose (GP)
- Amazon S3 Standard – Infrequent Access (IA)
- Amazon S3 One Zone-Infrequent Access
  - Cheaper IA with diluted availability
- Amazon S3 Intelligent Tiering
  - New – Amazon determines where to put data to save cost
- Amazon Glacier
  - Archival

AWS S3 Storage Tiers

	Standard	Standard - Infrequent Access	One - Infrequent Access	S3 Intelligent-Tiering	Glacier
Durability	99.999999999%	99.999999999%	99.999999999%	99.999999999%	99.999999999%
Availability	99.99%	99.9%	99.5%	99.90%	NA
AZ	≥3	≥3	1	≥3	≥3
Concurrent facility fault tolerance	2	2	0	1	1
	Frequently accessed	Infrequently accessed		Intelligent (new!)	Archives

S3 Lifecycle Rules

- In order to save on cost, the lifecycle rules help in moving data between different tiers
- Example:
  - General Purpose (GP) -> Infrequent Access (IA) -> Glacier
- Transition actions – Objects are transitioned to another storage class
  - Move objects from:
    - GP to IA, 60 days post creation
    - IA to Glacier 6 months post creation
- Expiration actions – S3 deletes expired objects on our behalf
  - Log files can be set to delete after a specific period of time

S3 Security - Encryption for Objects

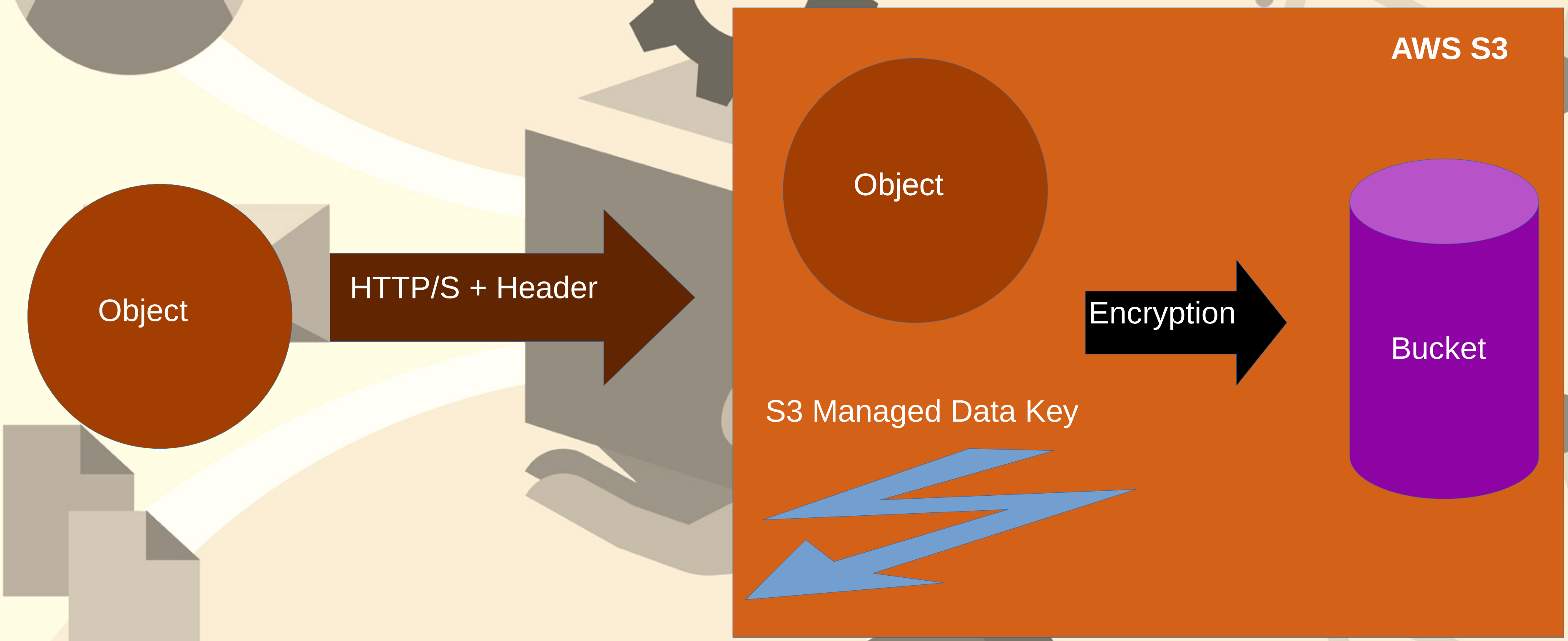
- There are four methods of encrypting objects in S3:

- SSE-S3: Encrypts S3 objects using keys handled and managed by AWS
- SSE-KMS: Use AWS key Management Service to manage encryption keys
  - Additional Security
  - Audit trail for KMS key usage

S3 Security - Encryption for Objects

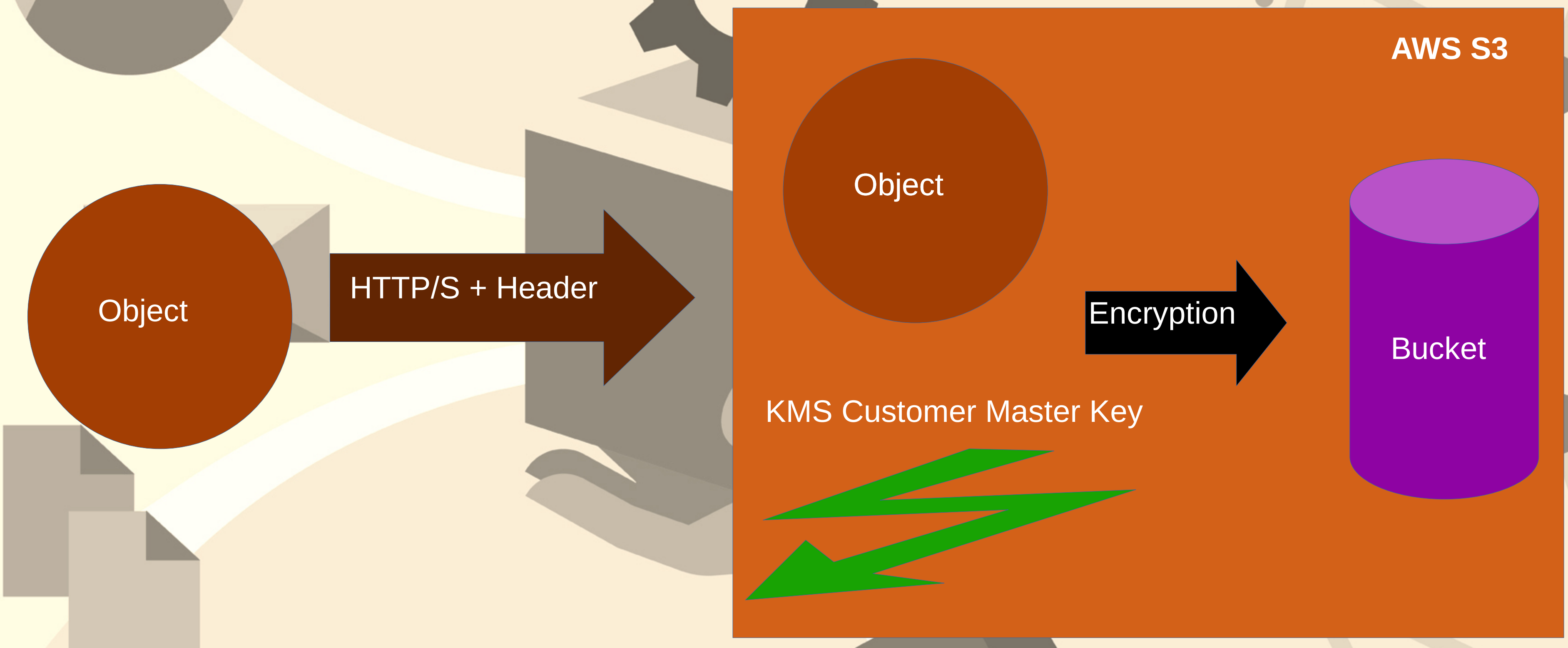
- SSE-C: We need to use our own encryption keys
- Client Side Encryption
- From an ML perspective, SSE-S3 and SSE-KMS will be the most likely used scenarios

SSE-S3



Object
HTTP/S + Header
AWS S3
Object
Encryption
Bucket
S3 Managed Data Key

SSE-KMS



Object
HTTP/S + Header
AWS S3
Object
Encryption
Bucket
KMS Customer Master Key

S3 Security

- User Based
  - IAM Policies – which API calls the user should be allowed
- Resource Based
  - Bucket Policy – allowing cross account access
  - Object Access Control List (ACL) – more precise control
  - Bucket Access Control List (ACL) – less commonly used

S3 Bucket Policies

- JSON based policies
  - Resources: buckets and objects
  - Actions: Set of API to Allow or Deny
  - Effect: Allow / Deny
  - Principal: The account or user to apply the policy to

S3 Bucket Policies

- Use S3 bucket policies for:
  - Granting public access to the bucket
  - For objects to be encrypted at upload
  - Grant access to another account (Cross Account)

S3 Security – Points to Remember

- Networking – VPC Endpoint Gateway
  - Allow traffic to stay within your VPC
  - Make sure the private services (Eg: SageMaker) can access S3
- Logging and Audit:
  - S3 access logs can be stored in other S3 bucket
  - API calls can be logged in AWS CloudTrail
- Tagged Based (combined with IAM and bucket policies)

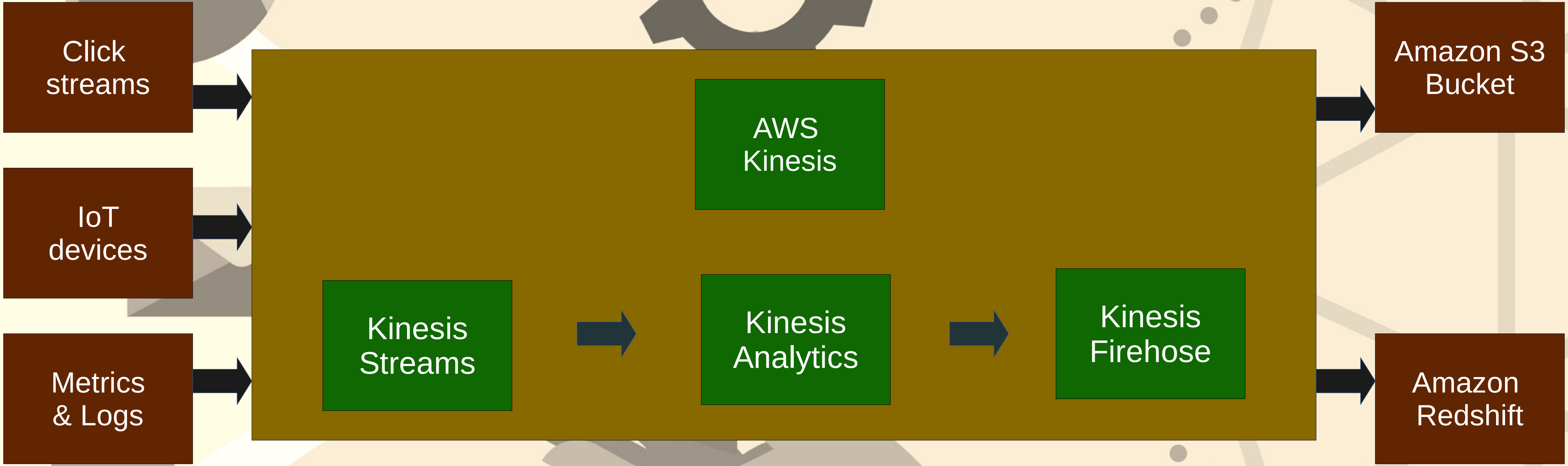
AWS Kinesis - Overview

- Kinesis is a managed alternative to Apache Kafka
- Great for application logs, metrics, IoT, click streams etc
- Any reference to ‘real-time’ in the exam is an indication of relation to Kinesis
- Great for streaming processing frameworks (Spark, NiFi etc)
- Data is automatically replicated synchronously to 3 AZs

AWS Kinesis – Key Services

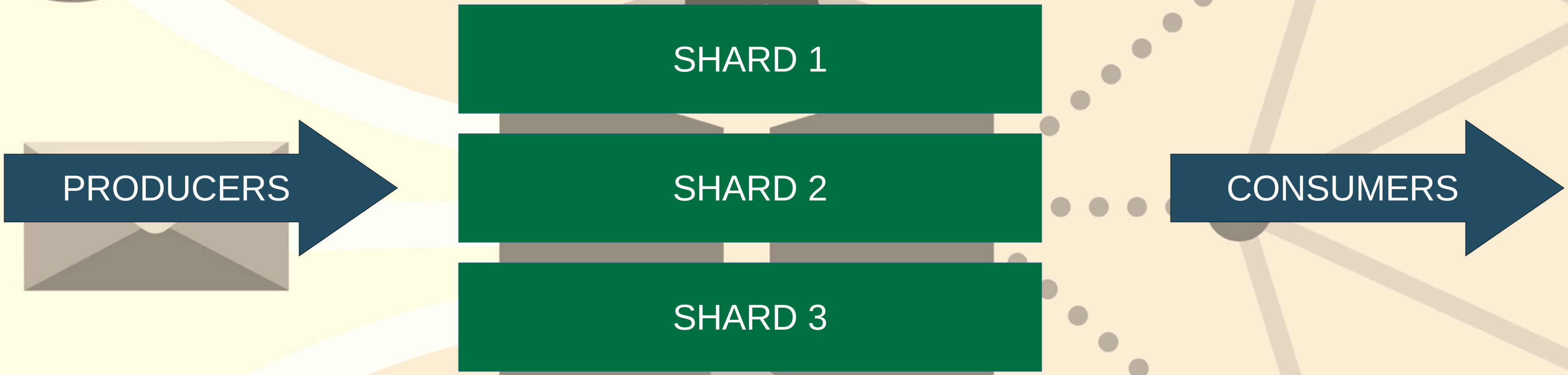
- Kinesis Streams – low latency streaming ingest at scale
- Kinesis Analytics – perform real-time analytics on streams using SQL
- Kinesis Firehose – load streams into S3, Redshift, ElastiSearch and Splunk
- Kinesis Video Streams – meant for streaming video in real-time

AWS Kinesis – Architecture



Kinesis Streams - Overview

- Streams are divided into ordered Shards / Partitions



- Shards have to be provisioned in advance (capacity planning)

Kinesis Streams - Overview

- Data retention is 24 hours by default and can go upto 7 days
- Ability to reprocess/replay data
- Multiple applications can consume the same stream
- Once data is inserted in Kinesis, it cannot be deleted (data immutability)
- Records can be upto 1 MB in size – fine for streaming use cases but not for large data analysis

Kinesis Data Streams - Limits

- Producers:
  - 1MB/s or 1000 messages/s write speed per shard
  - If exceeded – ‘ProvisionedThroughputException’
- Consumer (Classic):
  - 2MB/s read speed per shard across all consumers
  - Max 5 API calls/s/shard across all consumers
- Data Retention:
  - By default – 24 hours
  - Can be extended to upto 7 days

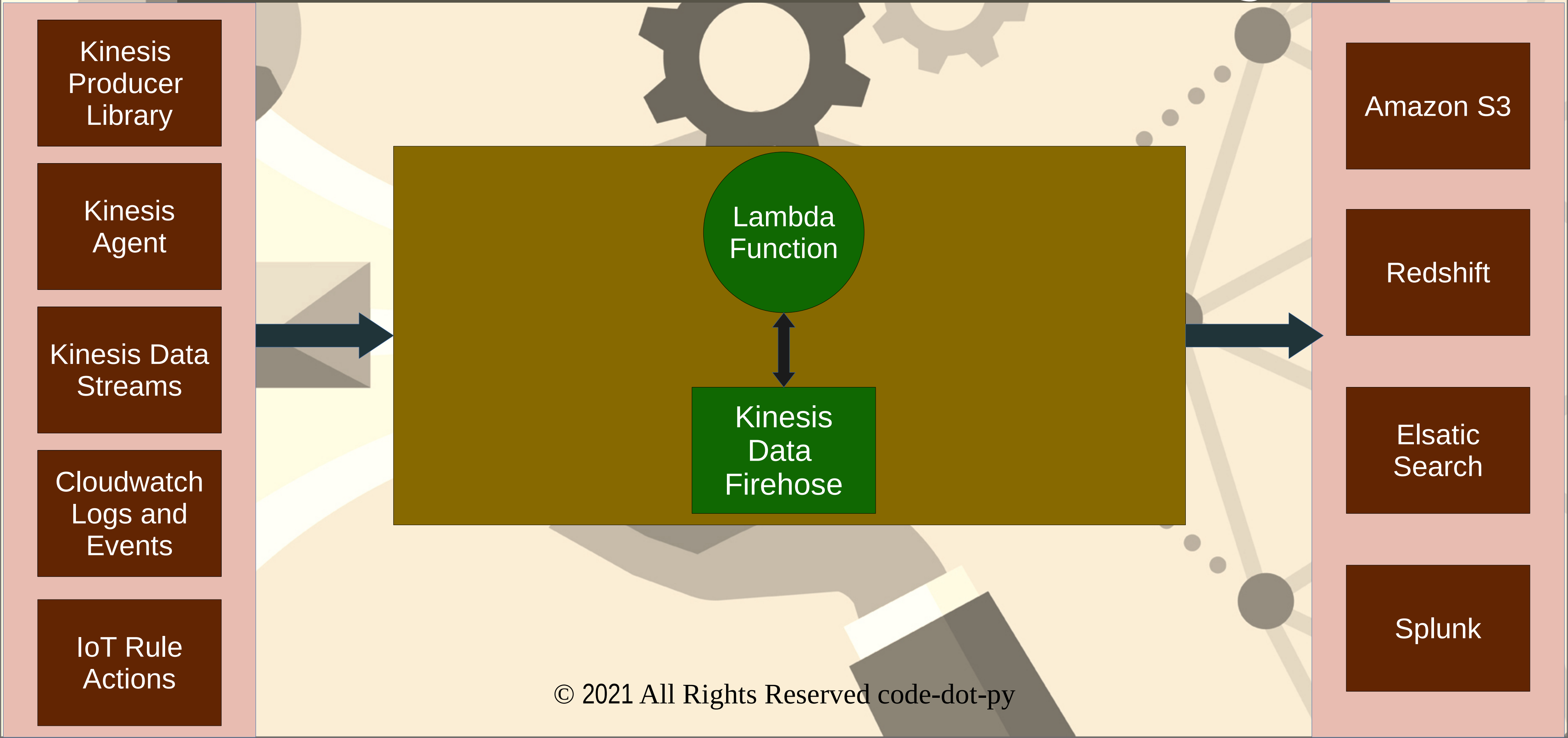
Kinesis Data Firehose

- Fully managed service, no administration
- Near real time (60 seconds latency minm for non full batches)
- Can perform data injestion into the following four services:
  - Redshift
  - Amazon S3
  - ElasticSearch
  - Splunk
- Automatic scaling

Kinesis Data Firehose

- Supports many data formats
- Data conversion from CSV/JSON to Parquet/ORC (only for S3)
- Data transformation through AWS Lambda (CSV => JSON)
- Supports compression when target is Amazon S3 (GZIP, ZIP and SNAPPY)
- Pay for the amount of data going through Firehose

Kinesis Data Firehose - Diagram



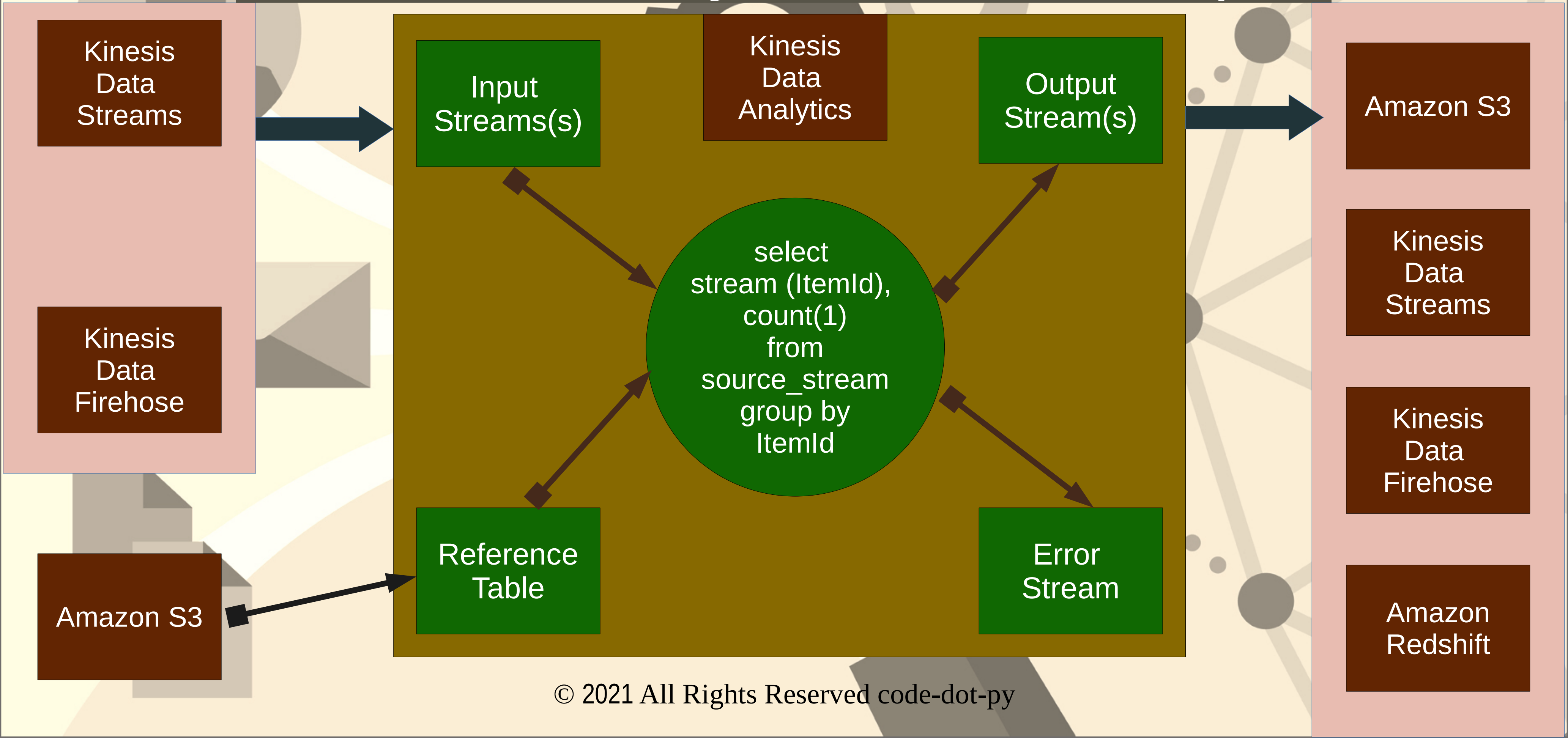
Kinesis Data Streams vs Firehose

- Streams
  - Supports custom code writing for producer/consumer
  - Real-time applications (latency ~200ms for classic and ~70ms for enhanced fan-out)
  - Must manage scaling ourselves (shard splitting/merging)
  - Data Storage for 1 to 7 days, replay capability, multi consumers

Kinesis Data Streams vs Firehose

- Firehose
  - Deliver or ingestion service
  - Fully managed. Can send to S3, Splunk, Redshift, ElasticSearch
  - Serverless data transformations with Lambda
  - Near real time (lowest buffer time is 60 seconds)
  - Automated scaling
  - No data storage – no replay capability

Kinesis Analytics – In Depth



Kinesis Data Analytics – Use Cases

- Streaming ETL
  - select columns, make simple transformation on streaming data
- Real-time metric generation
  - live leaderboard for a mobile game
- Responsive analytics
  - look for certain criteria and build alerting

Kinesis Data Analytics - Features

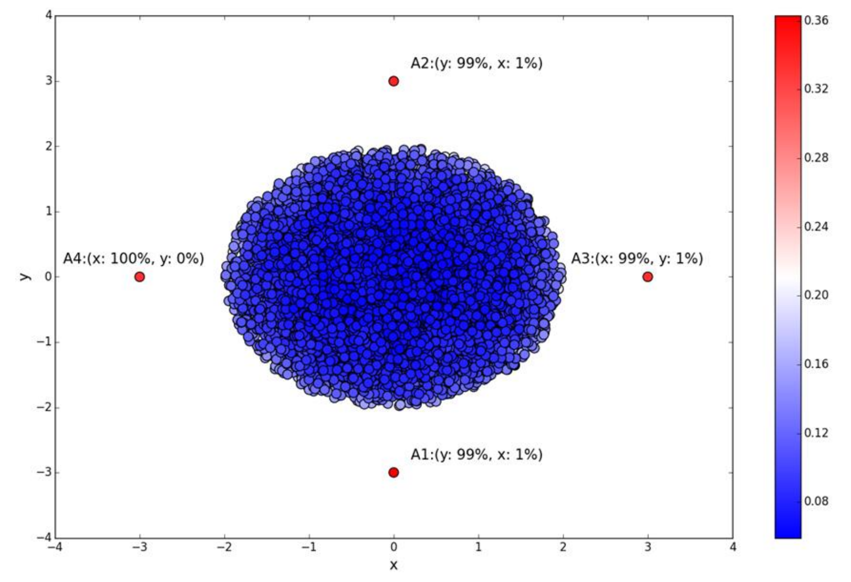
- Pay only for the resources consumed – not cheap
- Serverless and scales automatically
- Use IAM permissions to access streaming source and destination(s)
- SQL or Flink to write the computations
- Schema discovery
- Lambda can be used for preprocessing

Machine Learning on Kinesis Data Analytics

- There are two algorithms that can be used:
  - Random Cut Forest
  - Hotspots

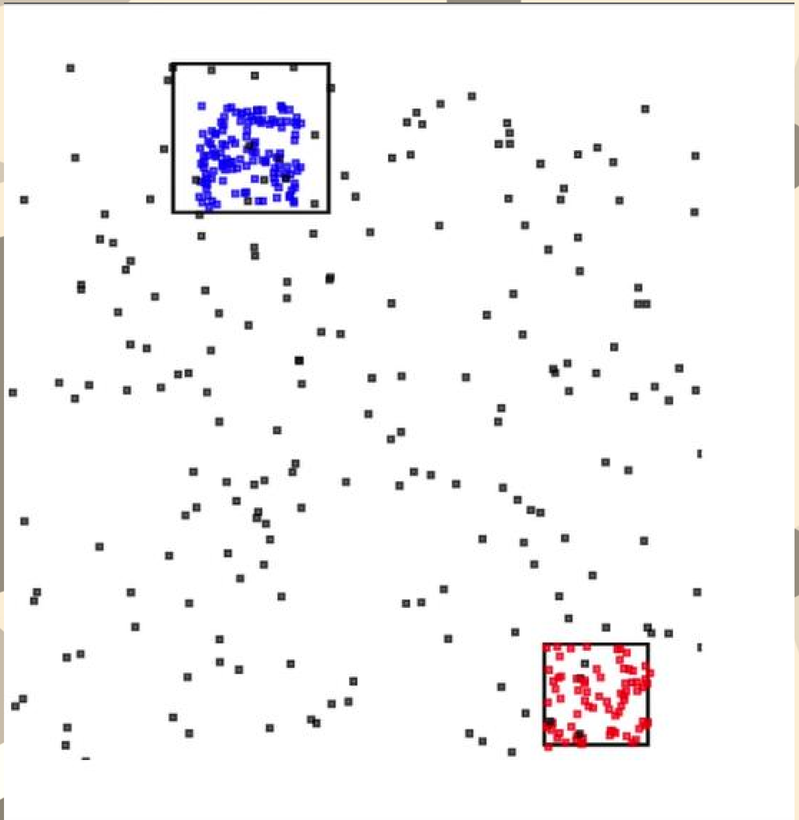
ML on Kinesis Data Analytics

- Random Cut Forest
  - Used for anamoly detection in a numerical column data in a stream
  - Provided as a SQL like function to use
  - Uses recent history to compute the model



ML on Kinesis Data Analytics

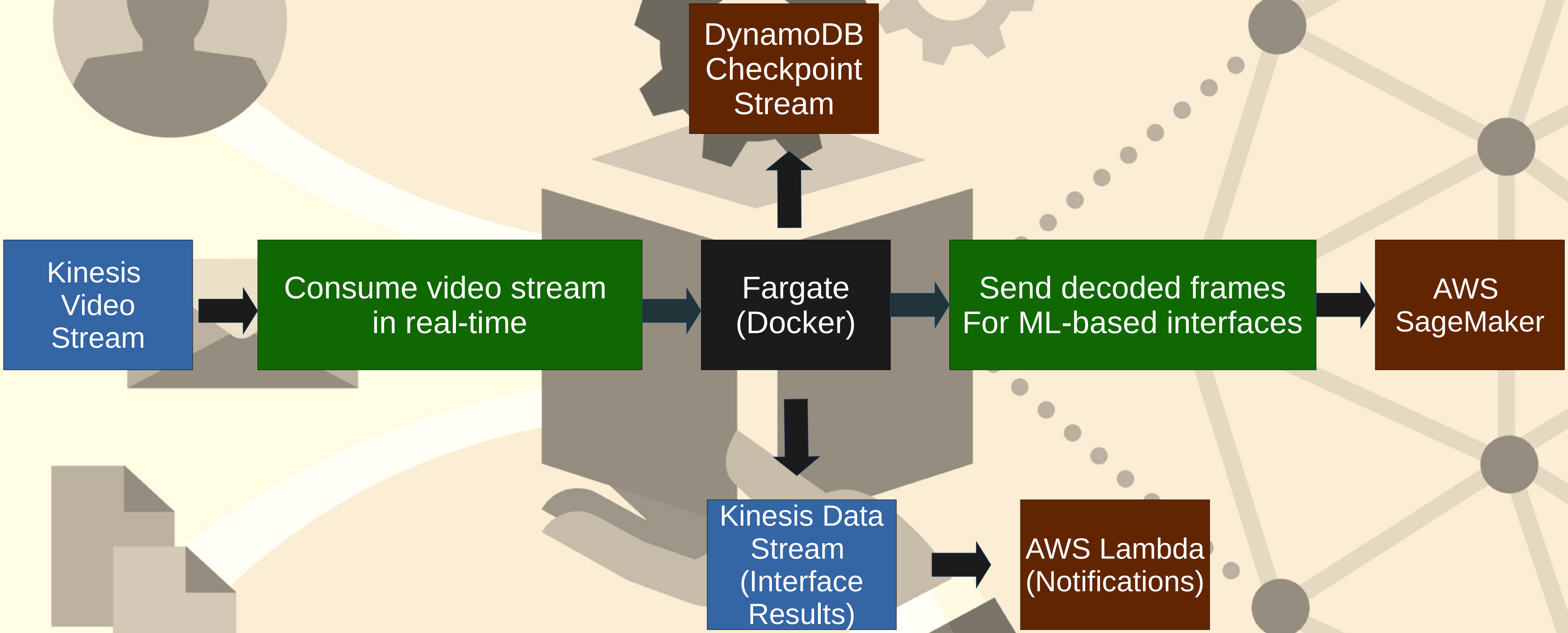
- Hotspot
  - Locate and return information about relatively dense regions in the data
  - Provided as a SQL like function to use



Kinesis Video Stream

- Producers
  - Cameras – security, body-worn, action, smartphone
  - AWS DeepLens
  - Audio and Image feeds
  - RADAR data
  - One producer per video stream
- Consumers
  - Custom – MXNet, Tensorflow
  - AWS SageMaker
  - Amazon Rekognition Video
- Data can be retained for 1 hour to upto 10 years
- Video playback capability

Kinesis Video Stream – Use Case



Kinesis Summary

- Kinesis Data Stream – read real-time data and create real-time ML applications on top
- Kinesis Data Firehose – ingest massive data with near-real-time scenario
- Kinesis Data Analytics – real-time ETL/ML algorithms on stream
- Kinesis Video Stream – real-time video stream to create ML applications

AWS Glue

- Glue Data Catalog
  - Metadata repository for all the tables within the account
    - Automated Schema Inferences
    - Schema Versioning
  - Integrates with AWS Athena or redshift Spectrum (schema and data discovery)
  - ‘Glue Crawlers’ can help building the Glue Data Catelog

AWS Glue

- Glue Data Catelog – Crawlers
  - Crawlers go through the data to infer schemas and partitions
  - They work with – JSON, Parquet, CSV, Relational data etc
  - They work with services like – AWS S3, Redshift, RDS
  - Crawlers can be run on a schedule or on-demand
  - Need IAM role/credentials to access the data stores

AWS Glue and S3 Partitions

- Glue crawler will extract partitions based on how the S3 data is organized
- It has to be thought-of upfront as to how the data will be queried
- Example:
  - A device is sending sensor data every hour
  - Query primarily by time-range -> Organize the bucket like:
    - s3://my-bucket/dataset/yyyy/mm/dd/**device**
  - Query primarily by device -> Organize the bucket like:
    - s3://my-bucket/dataset/**device**/yyyy/mm/dd/

mysamplemltestbucket

AWS Glue ETL

- Extract, Transform and Load
- Transform Data, Clean Data, Enrich Data before performing any analysis
- Glue ETL Features:
  - Generate ETL code in Python or Scala, the code can also be updated/modified
  - Can also work with custom Spark or PySpark scripts
  - Targets can be S3, JDBC (RDS, Redshift etc) or in Glue Data Catalog
- Fully managed, cost effective – pay only for the resources consumed
- Jobs are executed on a serverless Spark platform
- Glue Scheduler to schedule the jobs
- Glue Triggers to automate job runs based on ‘events’

AWS Glue ETL - Transformations

- Bundled Transformations:
  - DropFields, DropNullFields – remove (null) fields
  - Filter -specify a function to filter records
  - Join – to enrich data
  - Map – add fields, delete fields, perform external lookups
- Machine Learning Transformations:
  - FindMatches ML: Identify duplicate or matching records in your dataset, even when the records do not have a common unique identifier and no fields match exactly. Useful for data de-duplication
- Format conversions: CSV, JSON, Avro, Parquet, ORC, XML
- Apache Spark Transformations (Eg: k-means)

AWS Data Store for Machine Learning

- Redshift
  - Data Warehousing, SQL Analytics, OLAP
  - Load data from S3 to Redshift
  - Use Redshift Spectrum to query data directly in S3
- RDS, Aurora
  - Relational Store, SQL, OLTP
  - Must provision servers in advance

AWS Data Store for Machine Learning

- DynamoDB
  - NoSQL data store
  - Serverless – provision read/write capacity
  - Useful to store a machine learning model served by your application
- S3
  - Object storage
  - Serverless, infinite storage
  - Integration with most AWS Services

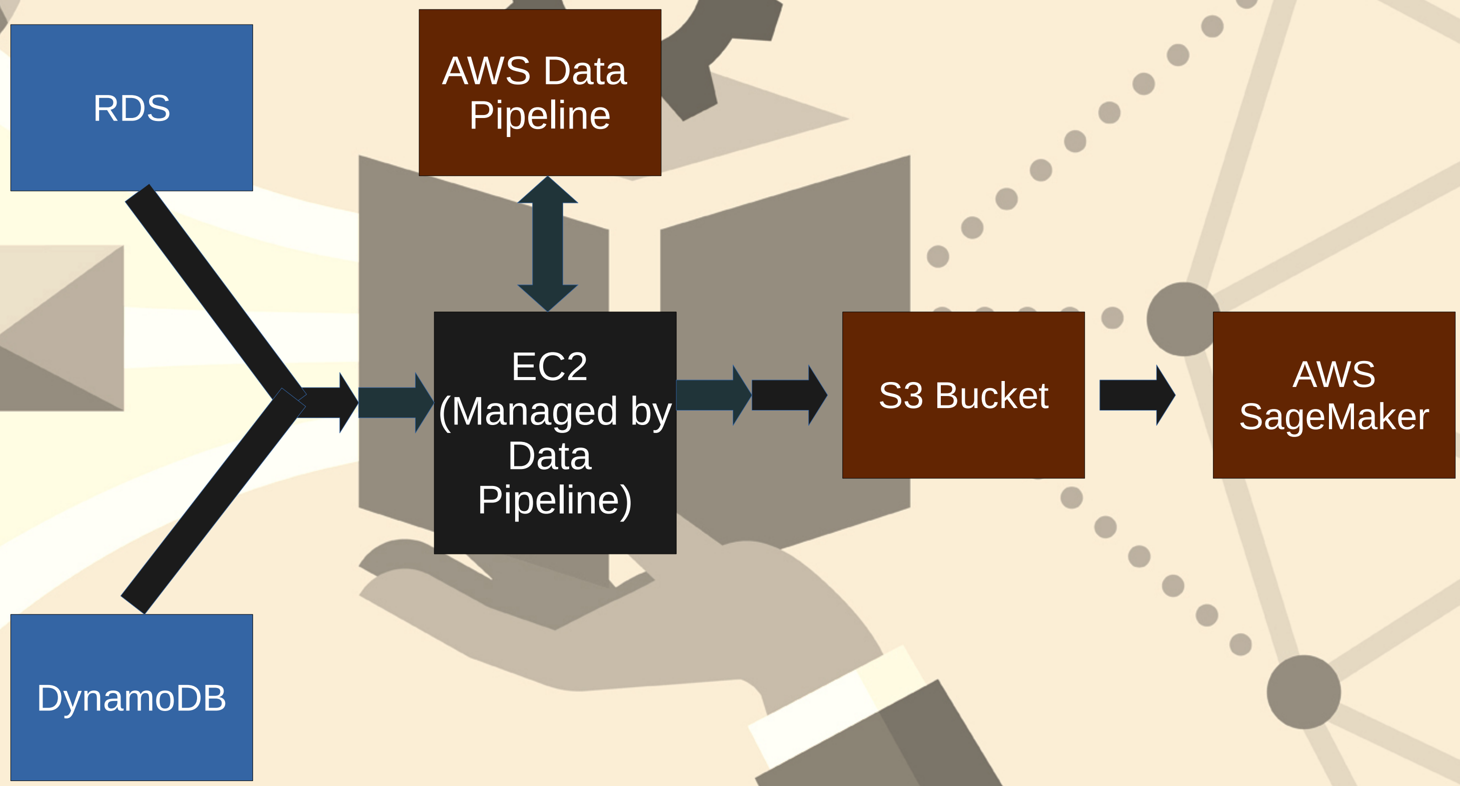
AWS Data Store for Machine Learning

- ElasticSearch
  - Indexing of data
  - Search amongst data points
  - Clickstream Analytics
- ElastiCache
  - Caching mechanism
  - Not really used for Machine Learning

AWS Data Pipeline Features

- Destinations include S3, RDS, DynamoDB, Redshift, EMR
- Manages task dependencies
- Retries and notifies on failures
- Data source may be on-premise
- Highly available

AWS Data Pipeline Example



AWS Data Pipeline vs Glue

- Glue:
  - Glue ETL: Runs Apache Spark Code
  - Scala or Python based
  - Focused on ETL
  - Not focused on configuring or managing resources
  - Catalogs data for Athena or Redshift Spectrum

- Data Pipeline:
  - Orchestration service
  - More control over:
    - The environment
    - Compute resources that run the code
    - Code itself
  - Allows access to EC2 or EMR instances
  - Creates resources in same account

AWS Batch

- Run batch jobs as Docker images
- Dynamic provisioning of the instances (EC2 and Spot Instances)
- Optimal quantity and type based on the volume and requirements
- No need to manage clusters, fully serverless
- Just pay for the underlying EC2 instances
- Schedule batch jobs using CloudWatch Events
- Orchestrate Batch Jobs using AWS Step Functions

AWS Batch vs Glue

- Glue:
  - Glue ETL: Runs Apache Spark Code
  - Scala or Python based
  - Focused on ETL
  - Not focused on configuring or managing resources
  - Catalogs data for Athena or Redshift Spectrum
- Batch:
  - For any computing job regardless of the job
  - Must be provided with a Docker image
  - The resources are created within the account and managed by Batch
  - For a non-ETL requirement, Batch is a better option

AWS DMS – Database Migration Service

- Quickly and securely migrate databases to AWS
- Resilient and self-healing
- The source database remains available during the migration
- Supports:
  - Homogeneous migrations – Oracle to Oracle
  - Hetrogeneous migrations – SQL Server to Aurora
- Continuous Data Replication using CDC
- An EC2 instance is needed to perform the replication task

AWS DMS vs Glue

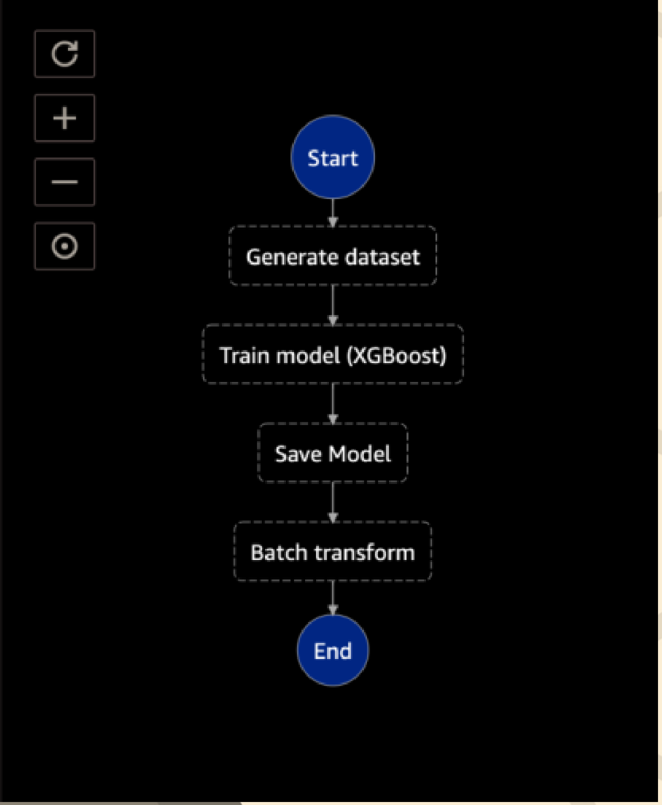
- Glue:
  - Glue ETL: Runs Apache Spark Code
  - Scala or Python based
  - Focused on ETL
  - Not focused on configuring or managing resources
  - Catalogs data for Athena or Redshift Spectrum
- DMS:
  - Continuous Data Replication
  - Service for Database migration
  - No data transformation
  - Post migration, ETL can be performed

AWS Step Functions

- Used to orchestrate and design workflows
- Easy visualizations
- Advanced Error Handling and Retry mechanism outside the code
- Audit of the history of workflows
- Ability to ‘Wait’ for an arbitary amount of time
- Maximum execution time of the state machine is 1 year

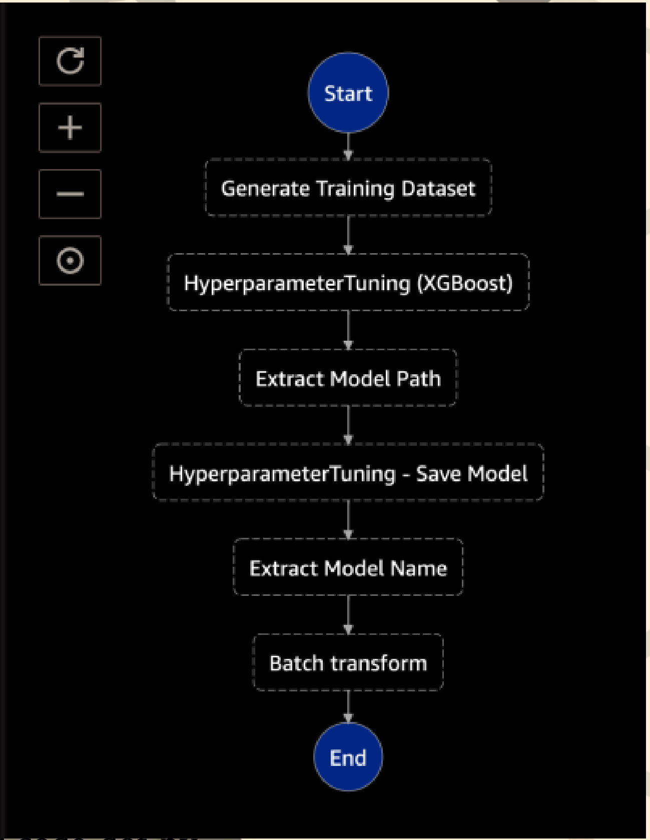
Step Functions – Examples
Train a Machine Learning Model

```
{
  "StartAt": "Generate dataset",
  "States": {
    "Generate dataset": {
      "Resource": "<GENERATE_LAMBDA_FUNCTION_ARN>",
      "Type": "Task",
      "Next": "Train model (XGBoost)"
    },
    "Train model (XGBoost)": {
      "Resource": "arn:
<PARTITION>:states:::sagemaker:createTrainingJob.sync",
      "Parameters": {
        "AlgorithmSpecification": {
          "TrainingImage": "<SAGEMAKER_TRAINING_IMAGE>",
          "TrainingInputMode": "File"
        },
        "OutputDataConfig": {
          "S3OutputPath": "s3://<S3_BUCKET>/models"
        },
        "StoppingCondition": {
          "MaxRuntimeInSeconds": 86400
        },
        "ResourceConfig": {
          "InstanceCount": 1,
          "InstanceType": "ml.m4.xlarge",
```



Step Functions – Examples
Tune a Machine Learning Model

```
{
    "StartAt": "Generate Training Dataset",
    "States": {
        "Generate Training Dataset": {
            "Resource": "<GENERATE_LAMBDA_FUNCTION_ARN>",
            "Type": "Task",
            "Next": "HyperparameterTuning (XGBoost)"
        },
        "HyperparameterTuning (XGBoost)": {
            "Resource": "arn:
<PARTITION>:states:::sagemaker:createHyperParameterTuningJob.sync",
            "Parameters": {
                "HyperParameterTuningJobName.$": "
<JOB_NAME_FROM_LAMBDA>",
                "HyperParameterTuningJobConfig": {
                    "Strategy": "Bayesian",
                    "HyperParameterTuningJobObjective": {
                        "Type": "Minimize",
                        "MetricName": "validation:rmse"
                    },
                    "ResourceLimits": {
                        "MaxNumberOfTrainingJobs": 2,
                        "MaxParallelTrainingJobs": 2
                    },
                    "ParameterRanges": {
```



Step Functions – Examples
Manage a Batch Job

```
{
  "Comment": "An example of the Amazon States Language for
notification on an AWS Batch job completion",
  "StartAt": "Submit Batch Job",
  "TimeoutSeconds": 3600,
  "States": {
    "Submit Batch Job": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::batch:submitJob.sync",
      "Parameters": {
        "JobName": "BatchJobNotification",
        "JobQueue": "<BATCH_QUEUE_ARN>",
        "JobDefinition": "<BATCH_JOB_DEFINITION_ARN>"
      },
      "Next": "Notify Success",
      "Catch": [
        {
          "ErrorEquals": [ "States.ALL" ],
          "Next": "Notify Failure"
        }
      ]
    },
    "Notify Success": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::sns:publish",
```