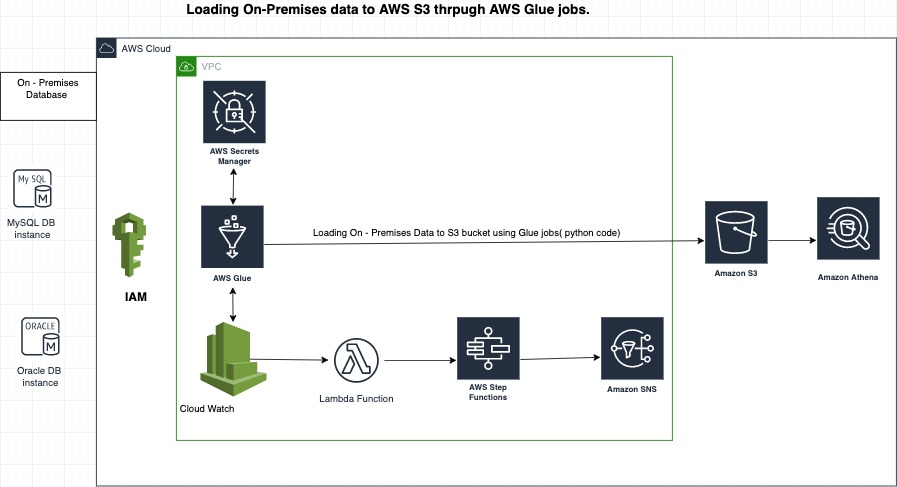
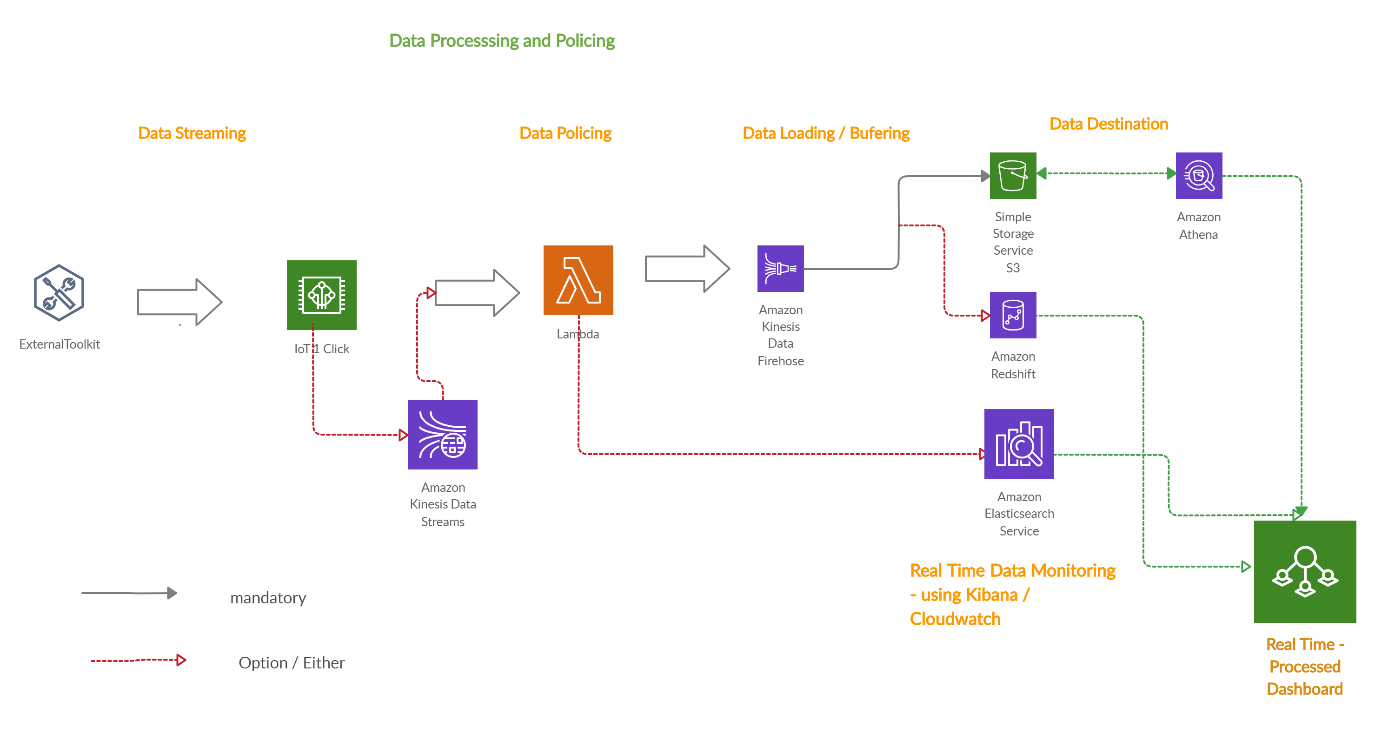
**Architecture diagram to load on - Premises data to AWS Cloud using AWS Glue.**

**Section A (Main Proto Type):**



**Section A ( Proto Type - Optional):**

****

**Section A ( Proto Type - Optional):**

A screenshot of a cell phone

Description automatically generated

**Section B :**

On -Premises : Loading Source data from database like Oracle,Mysql...etc to AWS Cloud (S3 Bucket).

**IAM** : AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.

**AWS Secrets Manager :**

AWS Secrets Manager is a secrets management service that helps you protect access to your applications, services, and IT resources. This service enables you to easily rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle. Using Secrets Manager, you can secure, audit, and manage secrets used to access resources in the AWS Cloud, on third-party services, and on-premises.

**AWS Glue:**

AWS Glue is a serverless data integration service that makes it easy to discover, prepare, and combine data for analytics, machine learning, and application development. In the present flow we are creating scripts using python & pyspark and trying to load data into S3 bucket . **AWS Glue** uses other **AWS** services to orchestrate your ETL (extract, transform, and load) **jobs** to build data warehouses and data lakes and generate output streams. **AWS Glue** calls API operations to transform your data, create runtime logs, store your **job** logic, and create notifications to help you monitor your **job** runs.

**AWS Cloud Watch :**

Amazon **CloudWatch** is a monitoring and management service that provides data and actionable insights for **AWS**, hybrid, and on-premises applications and infrastructure resources. With **CloudWatch**, you can collect and access all your performance and operational data in form of logs and metrics from a single platform.

**AWS Lambda:**

AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you. You can use AWS Lambda to extend other AWS services with custom logic, or create your own back-end services that operate at AWS scale, performance, and **security**.

**AWS Step functions :**

AWS Step functions are used to create dependency on AWS Glue jobs .it is a serverless orchestration service that lets you combine [AWS Lambda](https://aws.amazon.com/lambda/) functions and other AWS services to build business-critical applications. Through Step Functions' graphical console, you see your application’s workflow as a series of event-driven steps.

**Amazon SNS :**

It is a Notification service where you can send messages to many people in a single go and notify the update about the job .

**Amazon S3:**

Amazon Simple Storage Service (Amazon S3) is an object storage service , where we can store data in file format and load files data into the Redshift database directly .

**AWS Athena:**

Amazon **Athena** is a service that enables a data analyst to perform interactive queries in the **Amazon Web Services** public cloud on data stored in Amazon Simple Storage Service (S3). ... An **Athena** user can query encrypted data with keys managed by **AWS** Key Management Service, and can also encrypt query results.

**Section C:**

import boto3

import os

import datetime

import time

import sys

import datetime

from pyspark.sql import SQLContext

from pyspark.context import SparkContext

from pyspark.sql import functions as F

from pyspark.sql.functions import \*

from pyspark.sql.types import \*

from pyspark.sql import Row

from pyspark.sql.functions import udf

from operator import add

from functools import reduce

from pyspark.sql.types import StructField, StringType, IntegerType, StructType

import datetime

from pyspark.context import SparkContext

from pyspark.sql import SQLContext

from pyspark.sql import functions as F

from pyspark.sql.functions import \*

from pyspark.sql.functions import format\_string

from pyspark.sql.types import \*

from pyspark.sql.functions import explode

from pyspark.sql.functions import split

from pyspark.sql.functions import regexp\_replace

from pyspark.sql.functions import concat

from pyspark.sql.functions import col

from pyspark.sql.functions import lit

from pyspark.sql import Row

from pyspark.sql.functions import udf

from operator import add

from functools import reduce

from pyspark import SparkContext, SparkConf, SQLContext

from pyspark.sql.functions import unix\_timestamp

from pyspark.sql.functions import from\_unixtime

import boto3

import os

from datetime import tzinfo, timedelta, datetime

from awsglue.utils import getResolvedOptions

from awsglue.utils import getResolvedOptions

import sys

from awsglue.transforms import \*

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

from pyspark.sql import SparkSession

from pyspark.sql import SQLContext

#Initiate Spark Session

spark = SparkSession.builder.getOrCreate()

## Steps to initiate spark context, glue context and connecting SC to Glue

sc = spark.sparkContext

glueContext = GlueContext(sc)

job = Job(glueContext)

sqlContext = SQLContext(sc)

#Initiate Spark Session

spark = SparkSession.builder.getOrCreate()

#Reading data from Oracle Database

src\_query = "select \* from emp where empid=123"

src\_df = spark.read \

.format("jdbc") \

.option("url", "jdbc:oracle:thin:username/pwd@//10.120.9.6:1531/EDWDEV") \

.option("dbtable", src\_query) \

.option("user", "username") \

.option("header","true")\

.option("password", "pwd") \

.option("driver", "oracle.jdbc.driver.OracleDriver") \

.load()

# Writing DF data to S3 file

src\_df.write.mode("overwrite").format('com.databricks.spark.csv').save("s3://qbucket/test.csv’,header = "true")

Seed.yaml:

IAM role :

resource "aws\_iam\_user" "lb" {

name = "loadbalancer"

path = "/system/"

tags = {

tag-key = "tag-value"

}

}

resource "aws\_iam\_access\_key" "lb" {

user = aws\_iam\_user.lb.name

}

resource "aws\_iam\_user\_policy" "lb\_ro" {

name = "test"

user = aws\_iam\_user.lb.name

policy = <<EOF

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"ec2:Describe\*"

],

"Effect": "Allow",

"Resource": "\*"

}

]

}

EOF

}

Create vpc.tf

cidr\_block: 10.0.0.0/16 allows you to use the IP address that start with “10.0.X.X”. There are 65,536 IP addresses are ready to use.

instance\_tenancy: if it is true, your ec2 will be the only instance in an AWS physical hardware. Sounds good but expensive.

resource “aws\_vpc” “prod-vpc” {

cidr\_block = “10.0.0.0/16”

enable\_dns\_support = “true” #gives you an internal domain name

enable\_dns\_hostnames = “true” #gives you an internal host name

enable\_classiclink = “false”

instance\_tenancy = “default”

tags {

Name = “prod-vpc”

}

}

**Create Public Subnet**

We are still in the vpc.tf

vpc\_id: this subnet will be the vpc just created before. We give the created VPC id to the subnet.

cidr\_block: 10.0.1.0/24. We have 254 IP addresses in this subnet

map\_public\_ip\_on\_launch: This is so important. The only difference between private and public subnet is this line. If it is true, it will be a public subnet, otherwise private.

resource “aws\_subnet” “prod-subnet-public-1” {

vpc\_id = “${aws\_vpc.prod-vpc.id}”

cidr\_block = “10.0.1.0/24”

map\_public\_ip\_on\_launch = “true” //it makes this a public subnet

availability\_zone = “eu-west-2a”

tags {

Name = “prod-subnet-public-1”

}

}

**Create Internet Gateway**

Create network.tf

It enables your vpc to connect to the internet

resource "aws\_internet\_gateway" "prod-igw" {

vpc\_id = "${aws\_vpc.prod-vpc.id}"

tags {

Name = "prod-igw"

}

}

**Create Custom Route Table**

We are still in the network.tf

Create a custom route table for public subnet. public subnet can reach to the internet by using this.

resource "aws\_route\_table" "prod-public-crt" {

vpc\_id = "${aws\_vpc.main-vpc.id}"

route {

//associated subnet can reach everywhere

cidr\_block = "0.0.0.0/0"

//CRT uses this IGW to reach internet

gateway\_id = "${aws\_internet\_gateway.prod-igw.id}"

}

tags {

Name = "prod-public-crt"

}

}

**Cloud watch :**

module "lambda-cloudwatch-trigger" {

source = "infrablocks/lambda-cloudwatch-events-trigger/aws"

region = "eu-west-2"

component = "my-lambda"

deployment\_identifier = "production"

lambda\_arn = "arn:eu-west-1:example-lambda"

lambda\_function\_name = "my-lambda"

lambda\_schedule\_expression = "rate(2 days)"

}

module "lambda-cloudwatch-trigger" {

source = "infrablocks/lambda-cloudwatch-events-trigger/aws"

region = "eu-west-2"

component = "my-lambda2"

deployment\_identifier = "production"

lambda\_arn = "arn:eu-west-1:example-lambda"

lambda\_function\_name = "my-lambda2"

lambda\_schedule\_expression = "cron(0 20 \* \* ? \*)"

}

**Terraforms for Glue job creation :**

resource "aws\_glue\_job" "example" {

name = "example"

role\_arn = aws\_iam\_role.example.arn

command {

script\_location = "s3://${aws\_s3\_bucket.example.bucket}/example.py"

}

}