AWS Intensive

WELCOME!

Develop Intelligence

John Kidd


Cloudwatch

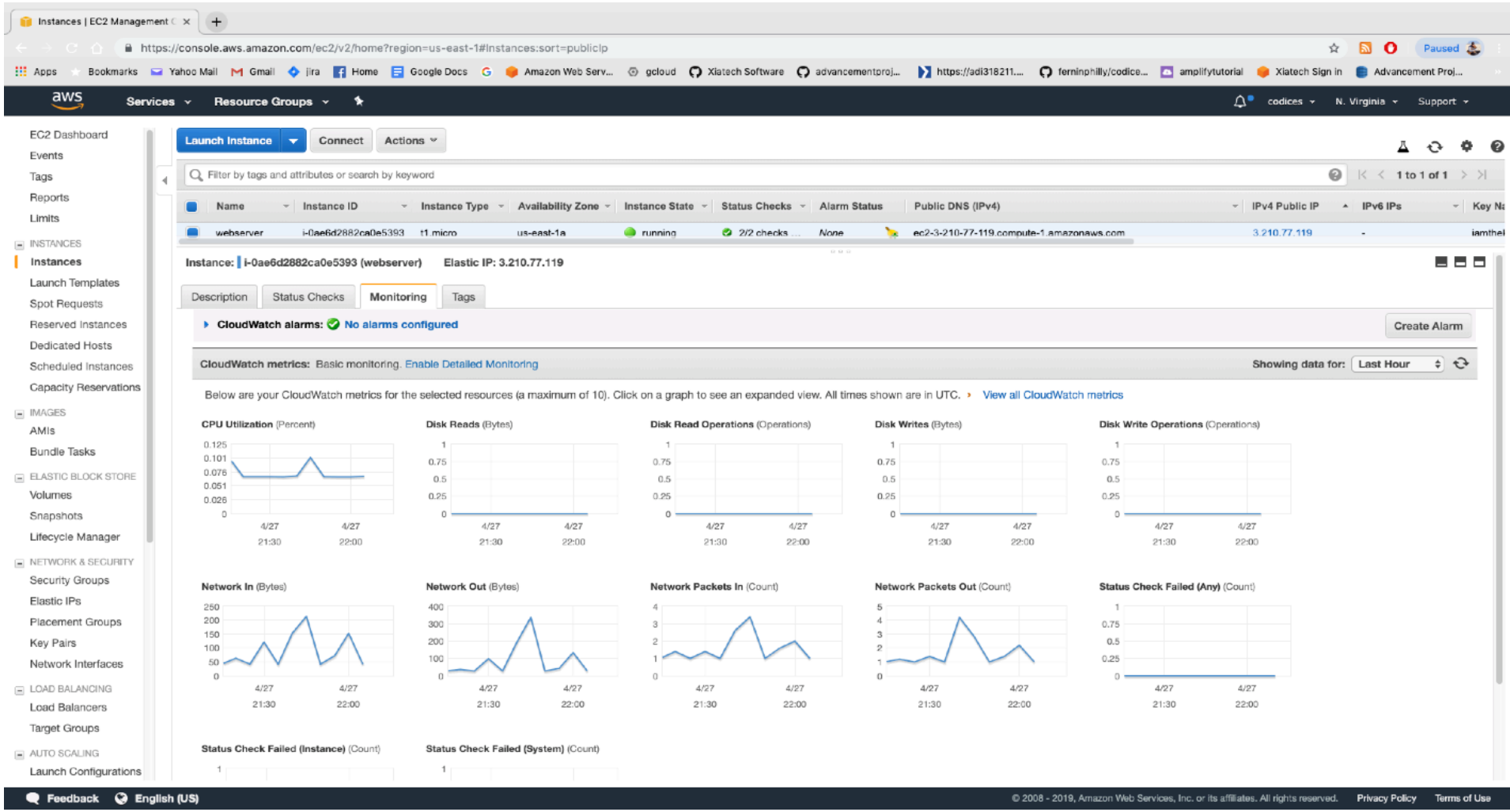
Infrastructure philosophy

- Logging and bug hunting is essential to any stack. One of the biggest advantages of using AWS for deploying and maintaining infrastructure is that logging is **built in** to the AWS infrastructure.
- This built in logging is known as **cloudwatch** and can be provisioned and utilized and managed via **terraform**.

Cloudwatch in terraform

- Go into your aws console and take a look at some of our previously deployed resources- specifically the EC2 webserver instance that we deployed
- Go down to the bottom of the screen and look at the **Monitoring** tab. You should see a bunch of charts there that demonstrate everything from CPU Utilization to Network IN, Disk Reads and Disk Writes, and Packets.

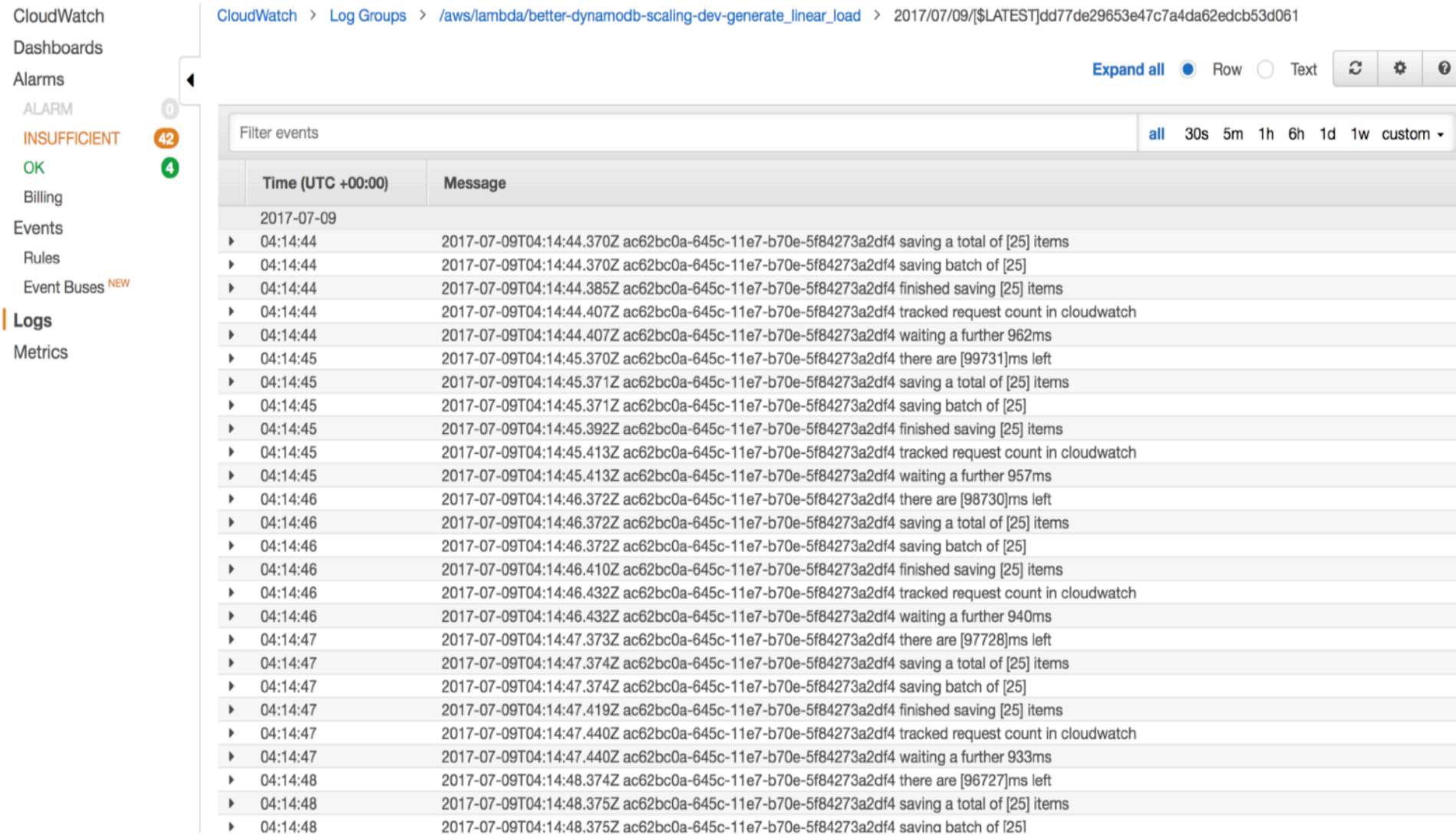
Cloudwatch (continued)



Cloudwatch (continued)

- Note that you can look at different time ranges.
- BUT- cloudwatch is MUCH more than just charts! You can also use cloudwatch to log issues sent to **STDOUT** and **STDERR** from an application.
- So we can read items sent from **console.log, print, Printf, etc…** to the log. We can also filter these items. In the lab we'll be deploying an application that does nothing but print out items.

Cloudwatch logs (continued)

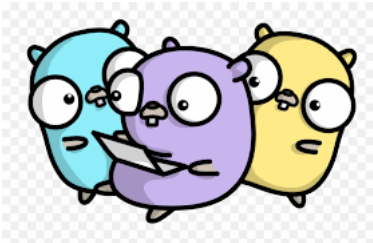


Cloudwatch Logs

- Cloudwatch logging should be the first place your developers go when they are looking for issues.
- Speaking from personal experience- I'd recommend having your dev team log out to stdout frequently to keep track of the flow of the application.
- All STDOUT errors and stack traces will also appear in cloudwatch logs- which you can have devs look at directly or forward directly to them.
- Finally- SYSTEM logs (what DEVOPS are interested in) and APPLICATION logs are different.

Cloudtrail agent

- So- along with cloudwatch at the application level there is also an add-on known as **cloudtrail** which is basically **cloudwatch** but on the AWS account level.
- Every event that occurs on your AWS account (provision an EC2 instance, log in to the console, etc…) can be sent as a log to an S3 bucket where you can then access it through an api interface.
- This is a good thing for the senior sys-ops personnel to use to monitor use (and costs) of AWS resources.

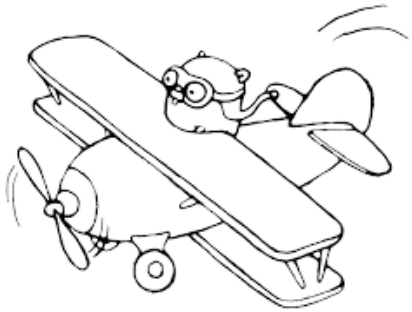


Cloudwatch trail (completion)

- Although **cloudtrail** is outside of the scope of deploying and managing applications it's worth studying up on if you are the senior sys-ops.
- AWS costs get out of control pretty quickly and from personal experience I'd suggest setting up cloudwatch alarms for certain activities (like creating Redshift instances at $150 per month a pop) and DynamoDB tables ($5k a pop).

Cloudwatch Alarms

- Another advantage of cloudwatch is that we can set up alarms in AWS that will notify the team when certain activities occur in an application.
- In the example of a web application we can set up alarms that notify us if there are more than three 404 errors in 5 minutes or a system overload or anything like that.



Cloudwatch Alarms (continued)

- You can set cloudwatch alarms with text recognition to notify a person or group depending on the type of text that comes in on the logs (we'll be doing this in the upcoming lab).
- This allows us the flexibility to customize the alarm depending on the error handling being used by your coders.

Cloudwatch alarms and SS

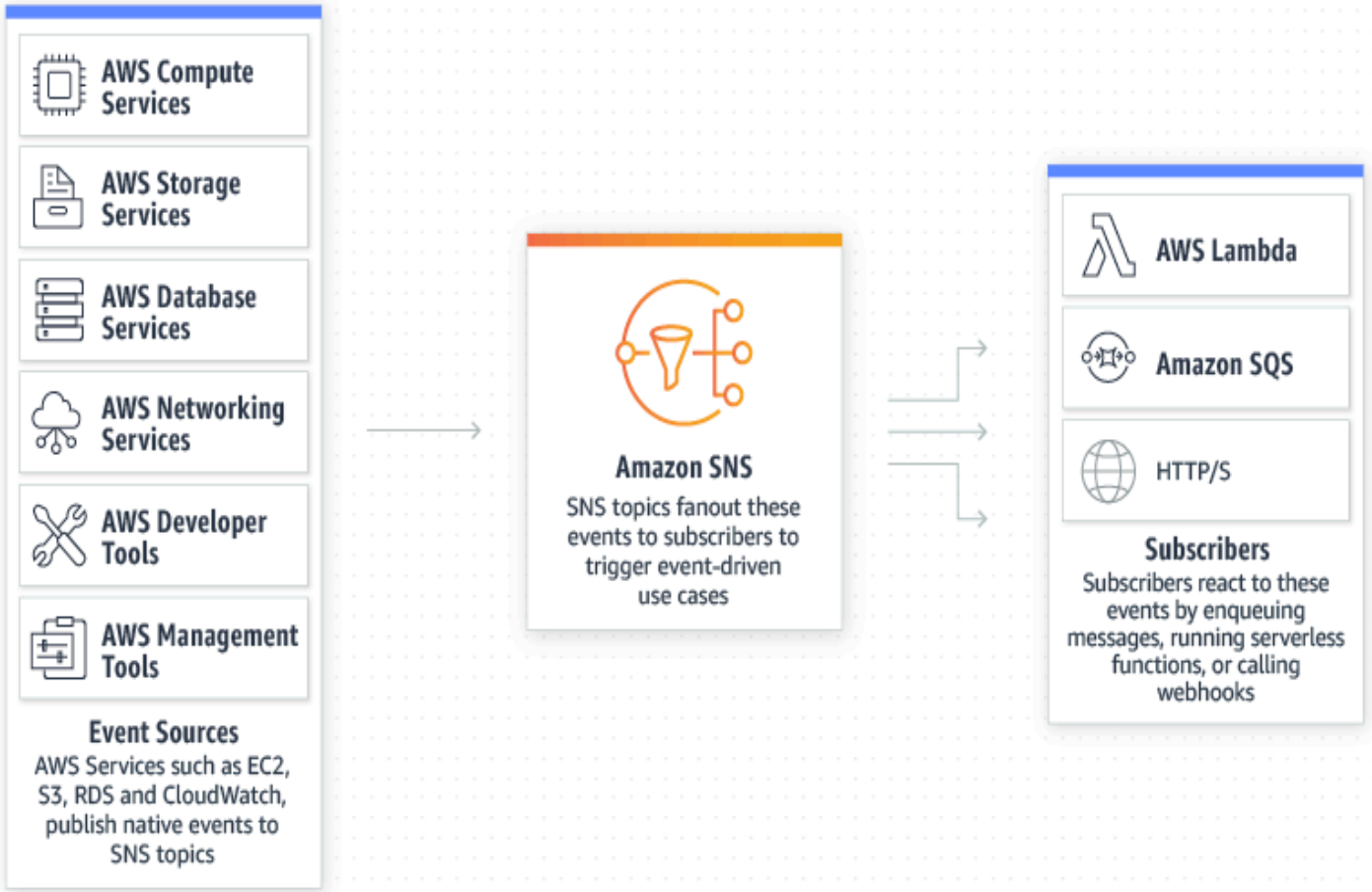
- So- while cloudwatch alarms will notify you as to what is happening…the next logical question is “notify WHO?...and HOW?”.
- The answer to this is another AWS service that we can use for multiple resources: the AWS **Simple Notification Service**.
- SNS can be used for notifications from most services but in this case we’ll be hooking it up to cloudformation.

Simple Notification Service

PUB/SUB

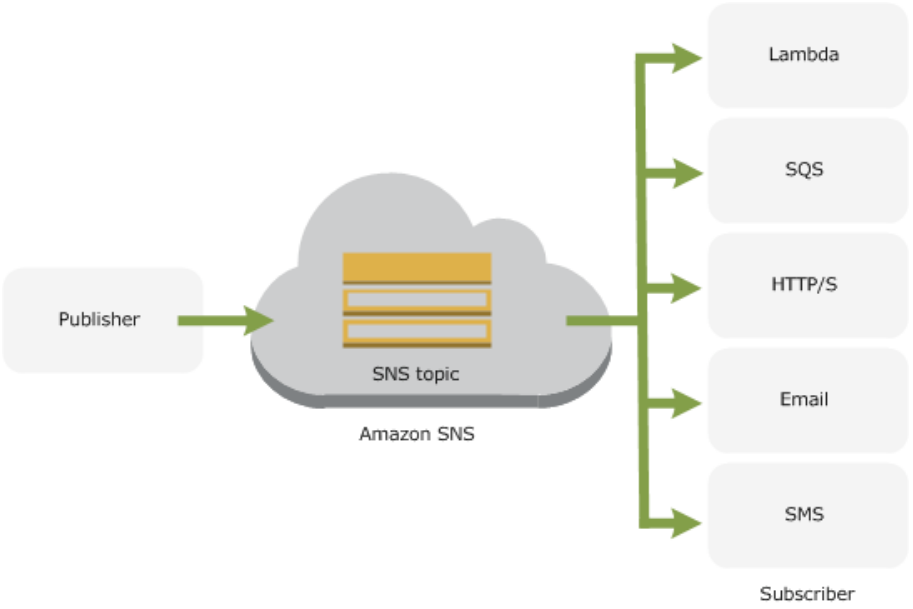
- So the AWS Simple Notification Service is basically a publish/subscribe (pub/sub) service in AWS.
- The service can send messages between various AWS resources based on the existence of an event. This makes it a key component of **event driven architecture.**
- Although we'll be utilizing SNS to send messages based on **cloudformation events…** this is just ONE of the events we can use to trigger messages.

Simple Notification Service



SNS Topics

- SNS publishes messages to TOPICS- which are what the client **subscribes** to.
- So we'll set up a TOPIC based on each type of **cloudwatch alarm** we want to create (for example "404 errors" or something.



SNS Broader context

- So TOPICS can be used for multiple triggers in your AWS infrastructure. Think of SNS as the glue that holds all of the microservices in your infrastructure together: it's a convenient way to pass messages between resources and ensure that you retain your **event driven architecture**.
- In the lab we're going to create a topic to send an email and text message based on a string pattern in cloudwatch.


Lambdas

Serverless applications

- So if SNS is the pub/sub glue that holds our serverless application together then LAMBDAS are the functional portion of our application.
- Think of Lambdas as small code repositories that do one very specific job for you without you having to worry about server management or deployment.
- Lambdas are an incredibly powerful tool that can do anything from create APIs to transform data mid-stream.

Lambdas

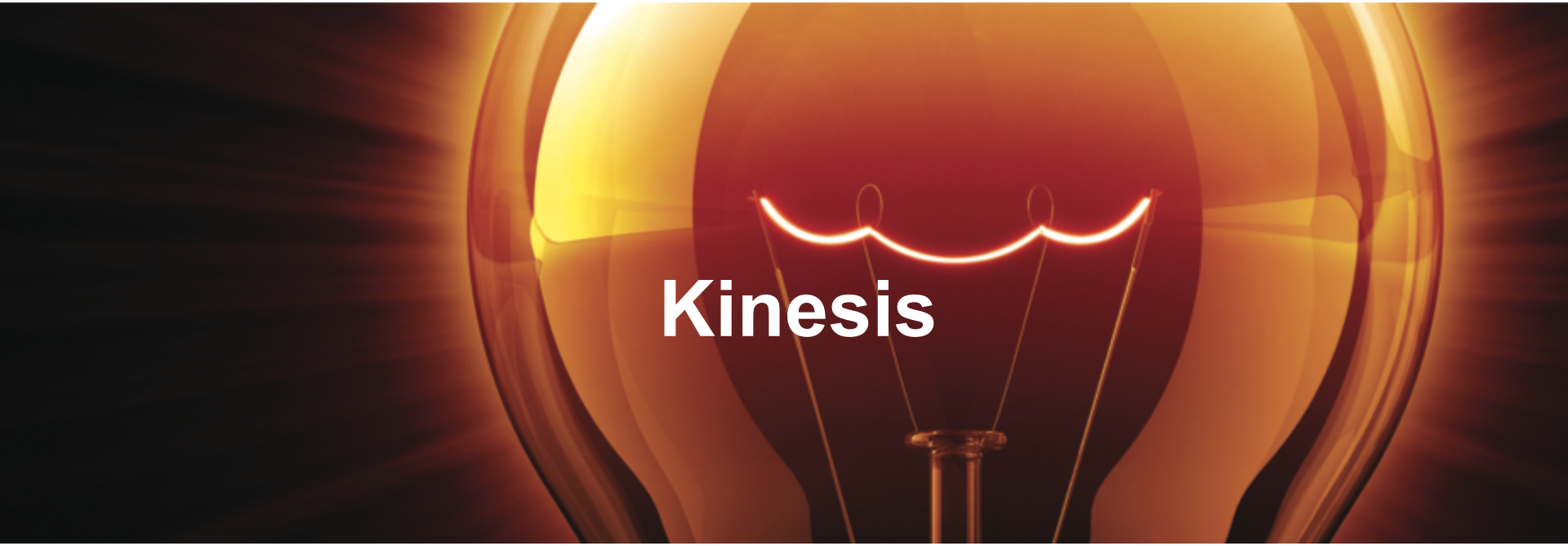
- So a lambda function is essentially a **container** (a lot like the Docker container we built in lab one) that is built and runs on any spare resources by AWS.
- AWS manages finding the resources, provisioning the container, and running it.
- The other thing to know about LAMBDAS is that, due to the nature of containers, they are designed to be *ephemeral*. This means that we want them to DO A THING then commit seppuku.

Lambdas and triggers

- Lambdas are a key component of any event driven architecture. You can have them run based on **lambda triggers** which are basically an event that occurs that sets off your lambda.
- This event can be anything from **adding data to an s3 bucket** to **receiving a message that we are subscribed to on an sns topic** to **consuming a queue from a Kinesis stream.**

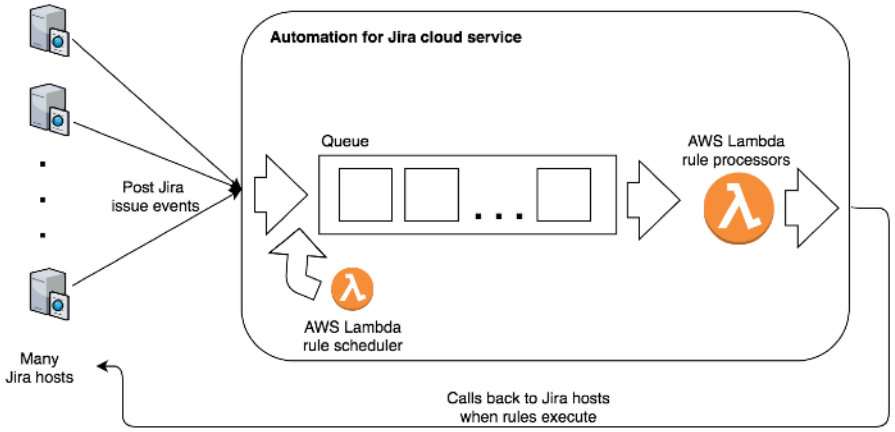
Event driven architecture and Lambdas

- SO- like everything else in AWS- Lambdas need to exist within a VPC and have cloudwatch logs attached.
- The ideal scenario for your architecture is that you create lambdas to do transformations on your data (or put data in to or take data out of a database) that run on **triggers** from other events passed through SNS.
- This allows you to save managing large machines to make up complex architectures.


Kinesis

Streaming Data in AWS

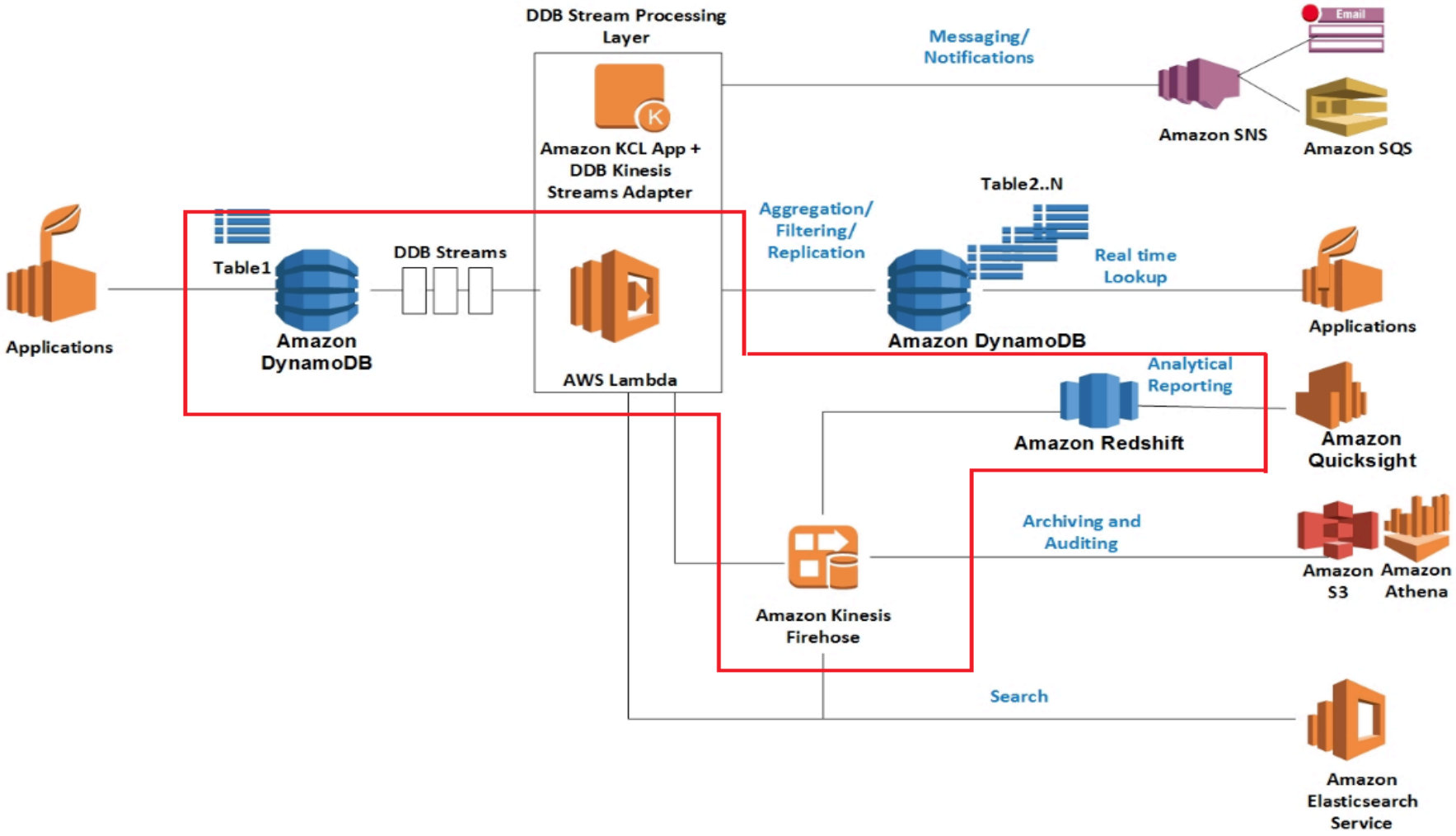
- So as a hypothetical- let's say we have a situation where data is coming in at an insanely fast rate… something like a twitter stream or sports odds stream.
- We want to create a system to consume this data, make some changes (to standardize the inputs) and then put the data into our previously created RDS instance.



Stream Data

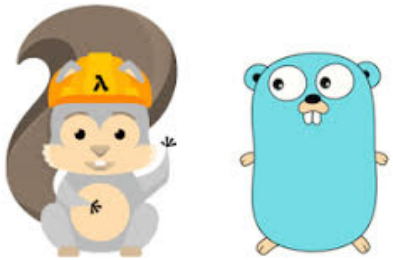
- The advantages of using queues in this situation is that we can keep the data in order and, in the event that our consumers slow down or fail, we can keep a copy of the data in a pre-database system.
- This will allow us time to spin up more instances if a consumer or two get overloaded by high velocity data going through the stream. This is a solid safety measure for data integrity.
- Additionally streams can provide endpoints that can **trigger events**.

Kinesis stream architecture



Kinesis streams conceptually

- So a basic unit of scale when dealing with Kinesis streams is about 1,000 PUTS per second of streaming data and emitting data at a rate of 2MB per second. This is a single **shard**.
- Shards scale linearly (2 shards = 2,000 puts, etc)
- Data is saved, by default, up to 24 hours (though you can pay to have it saved for up to 7 days)
- You can monitor streams through cloudwatch.



Kinesis Firehose

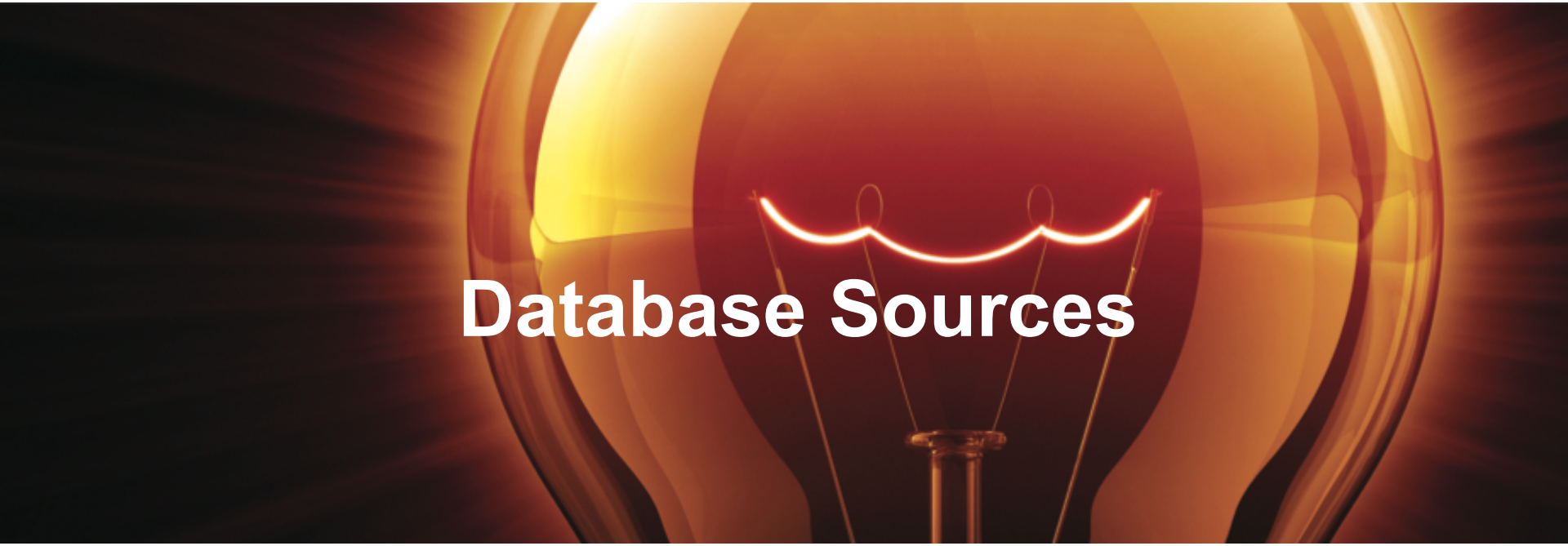
- Kinesis Firehose is the term used to refer to Kinesis services that stream between two AWS resources.
- An example of this might be a Kinesis stream that takes in data and sends it to a **redshift** database or **s3 bucket** (which is actually really nice- you can send a JSON object to REDSHIFT and, provided that the keys match column names, can PUT data into tables).

SQS vs SNS

- AWS offers a second option for message queuing: the **Simple Queue Service** which is a messaging broker (that looks a lot like the **Simple Notification Service**).
- The difference between **SQS** and **SNS** is simple:
  - **SNS** is a PUBSUB system- it *pushes* messages to *receivers* in order for them to do something. The receivers subscribe to a topic and when a message is published they receive it in near real time.
  - **SQS** is a *Message Broker:* consumers need to POLL it to get messages from it and you can't send the messages to multiple receivers. It's basically a mailbox- message in and message out.

SQS and Kinesis

- As Kinesis and SQS are both message brokers we should comment on the (subtle) differences between them:
  - The biggest difference is that SQS is a FIFO (First IN First OUT) queue whereas KINESIS is intended for **real time data processing…**so, as the name implies- it's really intended for a **stream** of data
  - Think of the difference you get from, say, an HTTP call versus a websocket connection stream: for an HTTP call it's probably better to use SQS but if you are opening up a websocket connection you want a kinesis stream.



Database Sources

Databases available in AWS

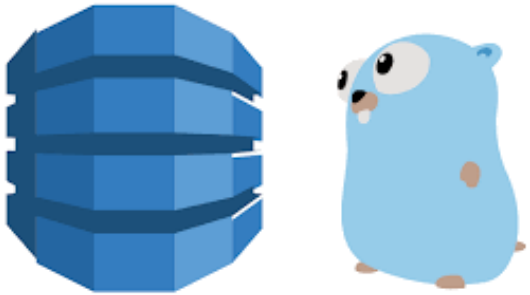
- So there are numerous options available for data storage in AWS. As pointed out earlier: fundamentally we'll want to deal with **s3 buckets** as a permanent data store.
- Try to, if you can, keep copies of all of the incoming data in gzipped files in s3 buckets based on a directory structure that you like
- **Kinesis** and **SQS** have libraries that will add unique IDs and store data in a directory structure with YEAR/MONTH/DAY/HOUR

How to think about choosing databases

- The idea behind AWS data storage tools is to save you the trouble of having to manage servers that would normally run your instances.
- AWS offers a lot of awesome options for your back end: In the SQL end you can choose between
  - MySQL
  - Postgres
  - Redshift
  - Aurora (think Mysql)

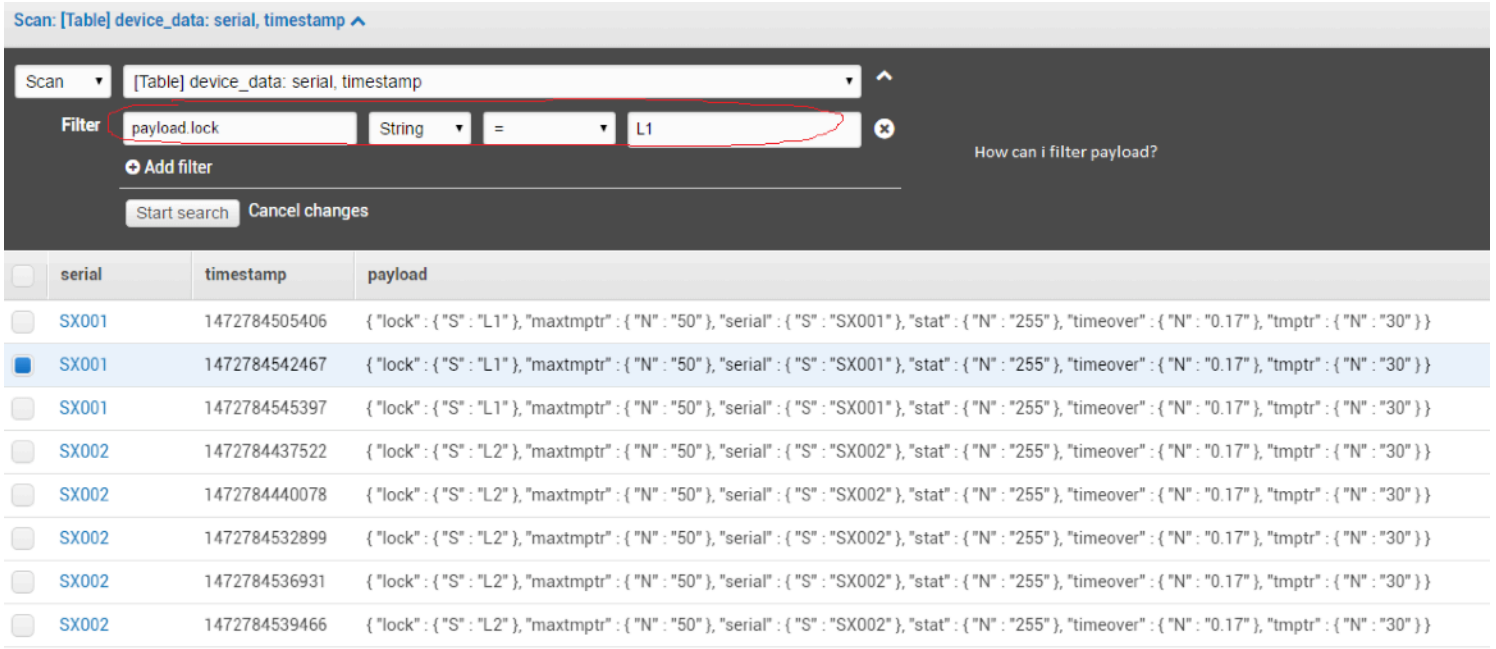
NoSQL solutions

- So for interim solutions for fast moving data (think websocket streams) it is frequently a good idea to have some way to do in-stream data transformations in near-real time. Think of things like **kafka** (which, interestingly enough, is also available in AWS now!)
- NOSQL solutions tend to be ideal for this and AWS offers a nice one known as **DynamoDB** to handle these streams.



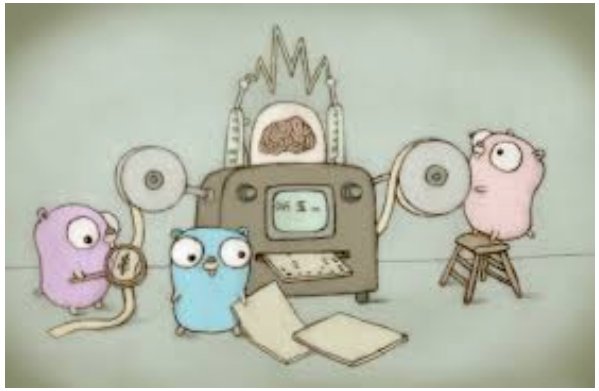
DynamoDB- managed NOSQL solution

- So Dynamodb saves your data as JSON objects- which is especially useful when dealing with things like node.js applications.
- You also get a nice interface:



DynamoDB and streaming

- Two more nice things to note about Dynamo:
  - It can handle streaming data nicely from Kinesis and you can make “in-stream” transformations (for instance changing a “raw event” to a structured bit of data in Lambda then storing it in Dynamo
  - It saves “before and after” images of any data transformed on a specific key…which is a nice way to log any changes on data going into your database (so if you need to “replay” an ingestion it’s a nice thing to have!



DynamoDB

- Think of DynamoDB as an excellent stop off and transformation opportunity for incoming data from a stream.
- You CAN use it as a permanent data store but it's not advisable as dynamo can be VERY expensive (if you DO use it use a ttl with the data).

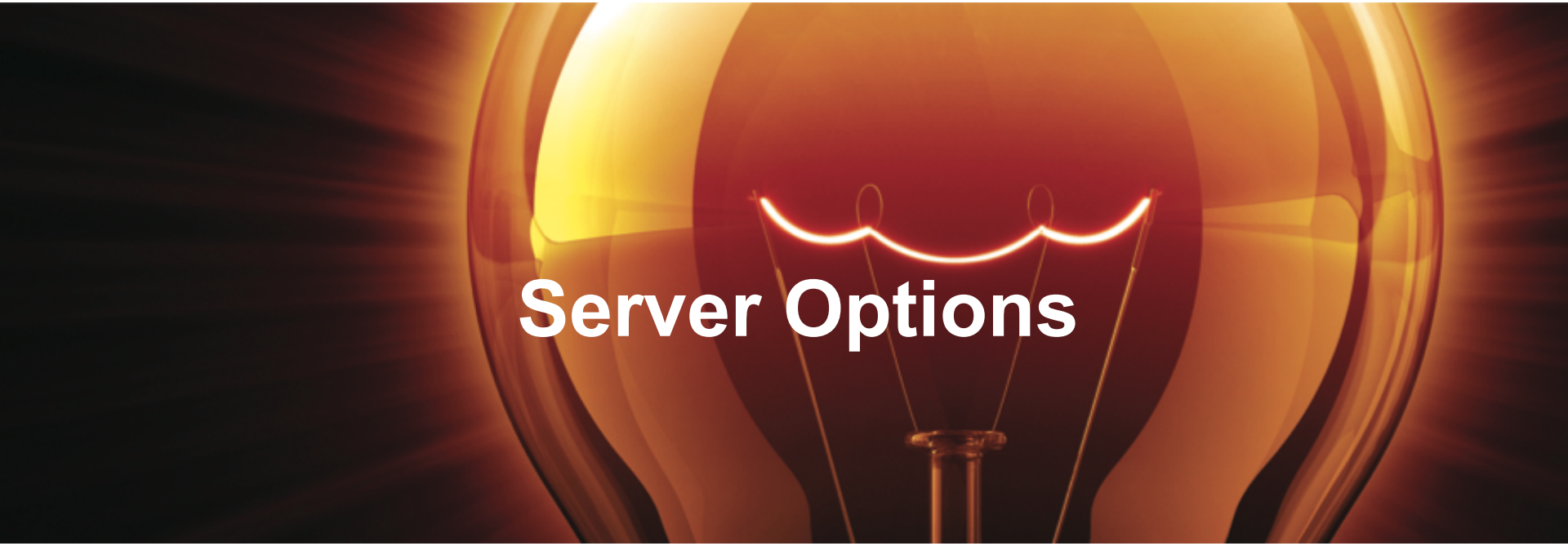
Redshift

- REDSHIFT is an excellent permanent analytic datastore. It's SQL based and works on POSTGRES syntax (which means that you can use windowed functions, etc)
- It is a columnar stored database that is capable of being scaled up to terrabytes and integrates with the AWS s3 buckets to form a nice endpoint to a datalake.

AWS RDS Service

- Outside of Redshift you have the options for AWS Relational Database Services that work with **any** of your favorite services:
  - MYSQL
  - Postgres
  - MariaDB (Mysql)
  - Oracle
  - SQL Server

The idea here is that you can choose whatever db you are most comfortable with and use that one for your app.



Server Options

EC2 instances

- The go to servers in AWS are known as EC2 instances. Just like everything else they come in **multiple** flavours depending on what you are comfortable with:
  - Amazon Linux
  - Red Hat Enterprise Linux (CentOS)
  - Suse Linux
  - Ubuntu
  - Windows

Elastic Block Store

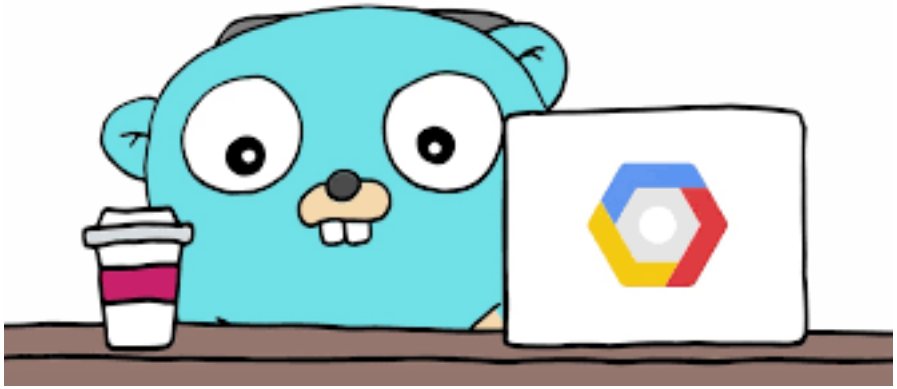
- One of the big advantages of utilizing EC2 in AWS is the availability of Elastic Block Store- which is basically a storage volume that you can attach to EC2 instances.
- You can take snapshots of EBS volumes into S3 thus providing reliability that you can't get anywhere else.
- This is useful if you're getting a data stream writing data to a local drive (so a local file system)

Elastic Block Store continued

- So Volumes in EBS are a fantastic way to persist data beyond the life of an ec2 instance.
- The idea is that you would mount the volume and use the directory for anything that requires high velocity throughput (read/writes, etc). The files in this mounted directory would be rapidly changing.
- The advantage of using a mounted volume here would be that you get snapshots sent to s3 therefore if there is an error you can roll back to a previous state.

Elastic Block Store

- If you've ever dealt with a situation where you needed a highly-available database structure with multiple nodes then hopefully you see how useful EBS blocks can be for storing data.
- Although they can only be attached to one instance at a time they make a powerful persistent store for data on EC2.



Confused????

- GOOD! Ask Questions!!