**AWS CSA-Pro Notes ACloudGuru Section 6 Architecting to Scale**

**Exam Tips:**

**Auto Scaling Groups:**

**Know the different scaling options and policies**

* 4 Scaling Options:
  + Maintain: Keep a specific or min number of instances running
  + Manual: Use max, min, or specific number of instances defined manually
  + Scheduled: Set a time for new instances to spin off maybe based on your pattern of traffic
  + Dynamic: Scale based on real-time metrics of the systems
* Scaling Policies:
  + Target Tracking: Scale based on a predefined or custom metric in relation to a target
  + Simple Scaling: Waits until health check and cool down period expires before evaluating new needs
  + Step Scaling: Responds to scaling in a step format for more granular scaling options

**Understand the difference and limitations between horizontal and vertical scaling**

* Horizontal: Adding or subtracting the same instance specs to your fleet
  + No down time, built in capabilities with Auto Scaling, Virtually unlimited
* Vertical: Adding more CPU or RAM to an existing Instances increasing its size individually
  + Down time required, limited in size, no built-in option to scale vertically

**Know what a cool down period is and how it might impact your responsiveness to demand**

* A cool down period is the amount of time you set before another scaling action can be triggered.
* If you did not set this you could have a loop where Auto Scaling would scale up super-fast without monitoring your new instances leaving you with way more than you need. This would then cause Auto Scaling to scale down your fleet.

**Kinesis:**

**Exam is likely to be restricted to the Data Stream use cases for Kinesis such as Data Streams and Firehose**

**Understand shard concepts and how partition keys and sequences enabled shards to manage data flow**

* Shards are able to ingest 1000 records per second
* To scale Kinesis up you would add more Shards. Shards are like lanes in a highway, the more lanes, the more traffic can go through.
* Each record put into a shard comes with a partition key and a sequence number
* Sequence numbers the order in which records go into a shard. 1st record would have this number be a 1 and so on.
* A partition key is unique to each shard so you can match the number 1 and use the partition key to know which shard that number 1 belongs too.

**DynamoDB Auto Scaling:**

**Know the new and old terminology and concept of a partition, partition and sort key in the context of DynamoDB**

* Partition key = Hash key
* Sort key = Range key

**Understand how DynamoDB calculates total partitions and allocates RCU and WCU across available partitions**

* Partitions are split into 10GB chunks. Once you get to 11GB you need 2 partitions

**Conceptually know how data is stored across partitions**

**CloudFront Part 2:**

**Know that both static and dynamic content is supported**

**Understand possible origins and how multiple origins can be used together with Behaviors**

* Origins:
  + S3, EC2, ELB, or another web server
* Using Behaviors you can split up traffic similar to path based routing for ALB.
  + You can define a web URL path for your dynamic content to go to your fleet of EC2 instances and have your static content be routed to an S3 bucket

**Know invalidation methods, zone apex, and geo-restrictions**

* Invalidation methods:
  + Delete the file from the origin and wait for the TTL to expire
  + Use AWS console to request an invalidation
  + Use the CloudFront API to submit an invalidation request
  + 3rd party tools
* Geo-Restrictions:
  + The CloudFront Geo-Restrictions are a basic whitelist or blacklist of countries
  + If you need more granular Geo-Restrictions for your website you should use a 3rd party tool. (Learned in Security Exam)

**SNS:**

**Understand a loosely coupled architecture and benefits it brings**

* A loosely coupled architecture bring a few benefits
  + Allows for independent scaling of components
  + IF one component fails it does not break your whole application

**Know the different types of subscription endpoints supported**

**SQS:**

**Know the difference between Standard and FIFO queues**

* Standards is just a normal queue
* FIFO is first in first out and should be used when you need to keep order.
* Disadvantage of using FIFO would be if one message gets stuck then it jams up the whole queue

**Know the difference between a Pub/Sub and Message Queueing architecture**

* Pub/Sub involves a publisher and different endpoints subscribing to the publisher. This is pushed based where anyone subscribed will get that message no matter what
* Message Queueing is more of a Poll based systems where workers take from the queue. Better for intermediate work where the queue is just a stopping point waiting to be processed.

**Lambda:**

**Know what Serverless is in concept and how Lambda can facilitate such an architecture**

* Servers are not managed by you. Lambda is a FaaS application where you define functions in code and Lambda carries out those tasks when triggered.

**Know the languages supported by Lambda**

**SWF:**

**Understand the difference and functions of a Worker and a Decider**

* Worker: A program that interacts with the AWS SWF service to get tasks, process tasks, and return results
* Decider: A program that controls coordination of tasks such as their ordering, concurrency, and scheduling

**Best suited for human-enabled workflows like order fulfillment or procedural requests**

**Elastic MapReduce:**

**Understand the parts of a Hadoop landscape at a high level**

* Hadoop HDFS- The file system that the data gets stored into
* Hadoop MR- The framework to do the data processing

**Know what a Cluster is and what Steps are**

* Cluster- A collection of EC2 instances provisioned by EMR to run your steps
* Steps- A programmatic task that is performed on the data

**Understand the roles of a Master Node, Core Nodes, and Task Nodes**

* Master Node- Orchestrates the Core and Task nodes
* Core Node- Where the data is stored and has persistent storage
* Task Node- These are the nodes that work on the steps defined and they have ephemeral data storage

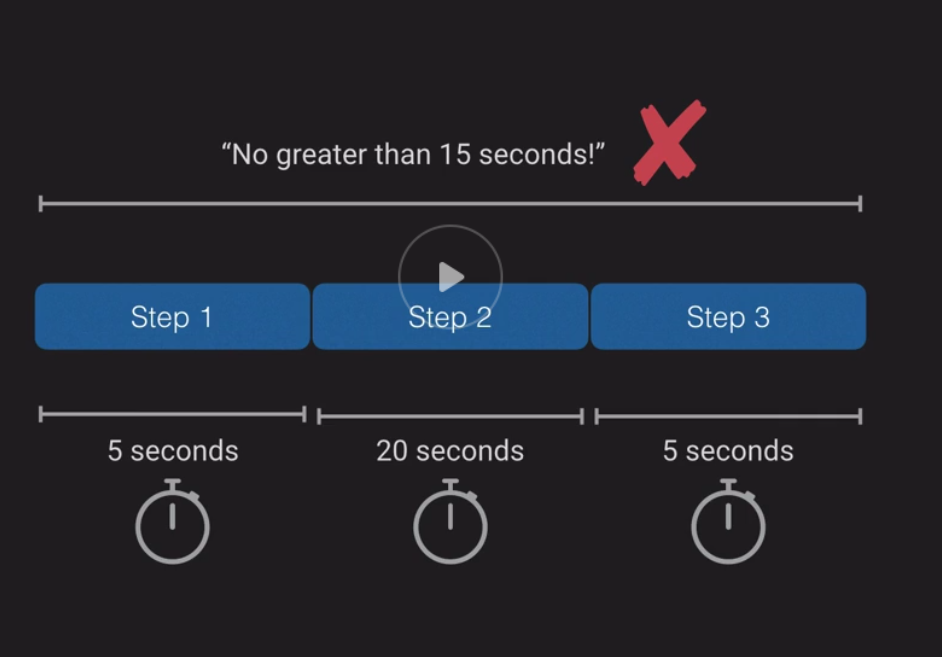
**Concepts:**

**Architectural Patterns:**

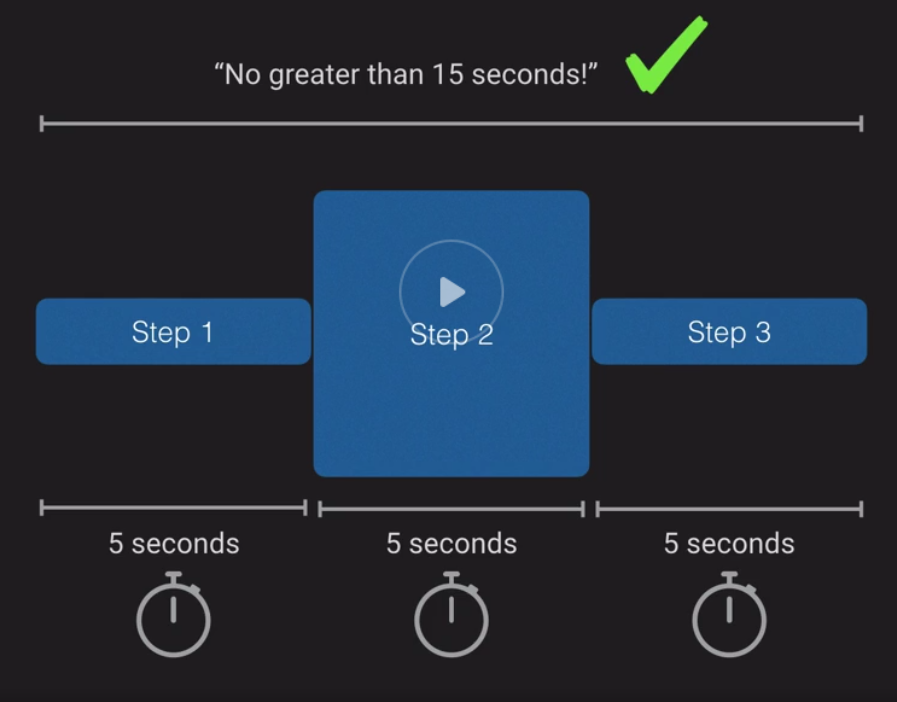
**Loosely Coupled Architectures:**

* Components can stand independently and require little or no knowledge of the inner workings of the other components
* Allows for better scalability by providing:
* Layers of abstraction
* Permits more flexibility
* Interchangeable components
* More atomic functional units
* Can scale components independently

**Tightly Coupled Architecture:**

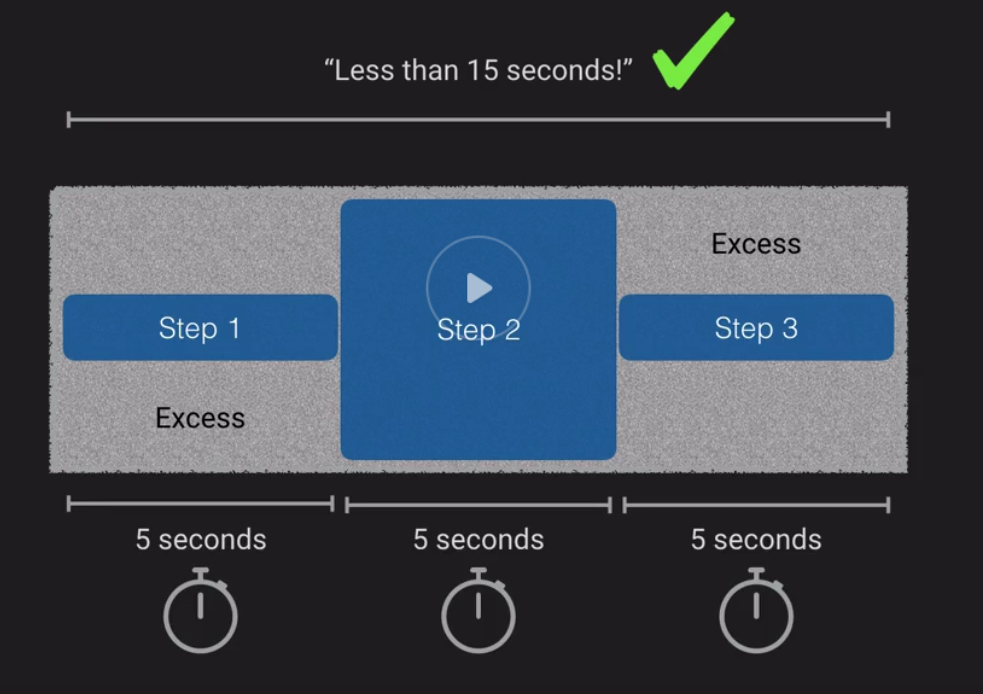
****

We want this entire process to be less than 15 seconds, but unfortunately it takes 25 seconds.

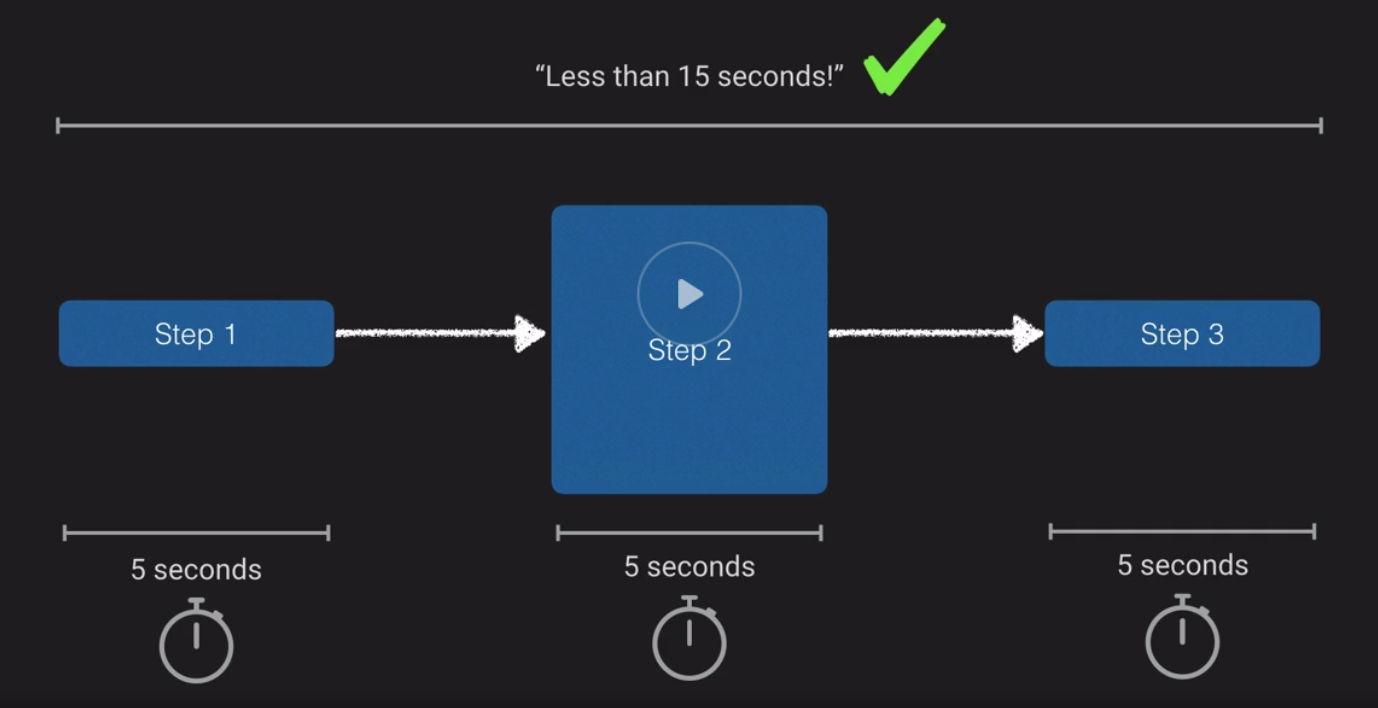
We can help achieve are goal if we provide step 2 with more computing power.

Because these are tightly coupled when we add more computing power to complete step 2 faster all of our other parts also get that computing power even when they do not need it.

With tightly coupled architecture you can’t just scale one step you have to scale the entire process.



If you have a loosely coupled process where you separate out all of the steps you can then scale steps independently allowing for the most efficient use of all your resources.



**Horizontal vs. Vertical Scaling:**

**Horizontal Scaling:**

* Add more instances as demand increases
* No downtime required to scale up or down
* Automatic using Auto-Scaling Groups
* Unlimited

**Vertical Scaling:**

* Add more CPU and/or RAM to existing instances as demand increases
* Requires restart to scale up or down
* Would require a custom script to automate
* Limited by instance sizes

Scaling out is adding instances.

Scaling up is adding resources to an existing instance.

Scaling in is to take away instances.

Scaling down is decreasing resources on an existing instance.

**Autoscaling Groups:**

* Automatically provides horizontal scaling for your landscape
* Triggered by an event or scaling action to either launch or terminate instances
* Availability, cost, and system metrics can all factor into scaling
* Four scaling options:
  + Maintain- Keep a specific or minimum number of instances running
  + Manual- Use max, min, or specific number of instances
  + Schedule- Increase or decrease instances based on schedule
  + Dynamic- Scale based on real-time metrics of the systems

**Launch Configs:**

* Defines what instances are launched when ASG are triggered

**Auto-Scaling Groups:**

* Specify VPC and subnets for scaled instances
* Attach to a ELB
* Define a Health Check Grace Period
* Define size of group to stay at initial size
* Or use scaling policy which can be based from metrics

**Scaling Policies:**

**Target Tracking Policy:**

* Scale based on predefined or custom metric in relation to a target value
* You have a target goal and if you go above that goal for the whole of your instances then you scale
* When CPU utilization gets to 70% on current instances, scale up

**Simple Scaling Policy:**

* Waits until health check and cool down period expires before evaluating new need

**Step Scaling Policy:**

* Responds to scaling needs with more sophistication and logic

**Scaling Cooldowns:**

* Configurable duration that gives your scaling a chance to “come up to speed” and absorb load
* Default cooldown period is 300 seconds
* Automatically applies to dynamic scaling and is optionally to manual scaling, but not supported for scheduled scaling
* Can override default cooldown via scaling-specific cool down

**Kinesis:**

* Collection of services for processing streams of various data
* Data is processed in “shards” – with each shard able to ingest 1000 records per second
* The default limit of 500 shards can be requested to be raised by contacting amazon support
* Record consists of Partition Key, Sequence Number and Data Blob (up to 1MB)
* Kinesis is a Transient data store
  + Default retention of 24 hours for your data
  + Up to 7 days

**Kinesis Video Streams:**

* Allows you to process video streams in real time.
  + Most likely not going to be on the exam

**Kinesis Data Streams:**

* Allows you to ingest a high volume of data and process it through a variety of ways

**Kinesis Firehose:**

* Prepares and loads the data continuously to the destinations you choose

**Kinesis Data Analytics:**

* Run standard SQL queries against data streams as the data comes in

**Kinesis Data Stream Key Concepts:**

* Kinesis processes data in shards
* Shards are like lanes on a highway. The more lanes the more traffic can go through at once
* Max of 1000 writes/sec per shard
* Each shard is given a partition key (unique to each shard)
* Each piece of data gets assigned a sequence number
* Sequence numbers can be duplicated across shards



**DynamoDB Scaling:**

* Scaling is divided into 2 dimensions
  + Throughput
  + Partition size

**DynamoDB Terms:**

**Partition:**

* A physical space where DynamoDB data is stored
* Divided into 10GB chunks

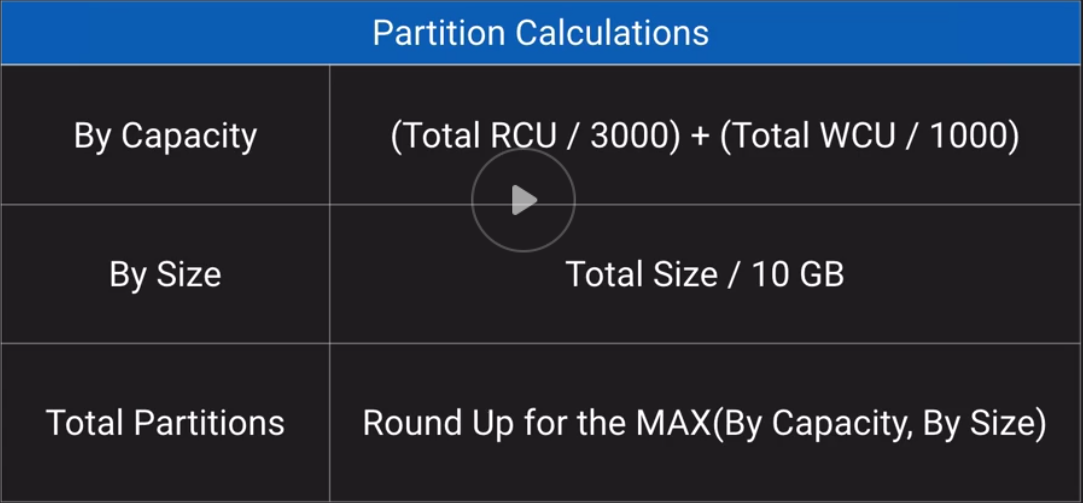
**Partition Key:**

* A unique identifier for each record; Could be called a hash key

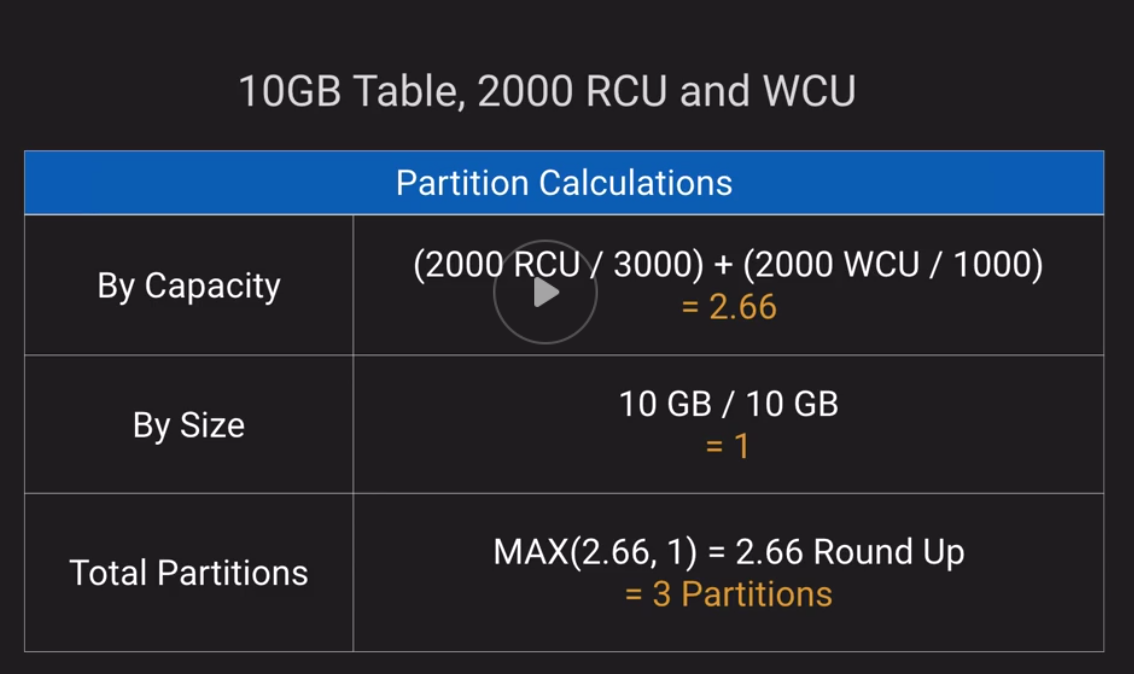
**Sort Key:**

* In combination with a partition key, optional second part of a composite key that defines storage order. Could be called a range key.

DynamoDB scales out by adding partitions:



Example of calculations:



DynamoDB also allows for Auto Scaling when you reach a certain Target utilization.

**Auto Scaling for Dynamo DB:**

* Uses Target Tracking method to try to stay close to target utilization
* Currently does not scale down if table consumption drops to 0
  + DynamoDB cannot tell if it is simply not being used or the demand is gone for good
  + Workaround: send requests to the table until it auto scales down
  + Workaround 2: Manually reduce the max capacity to be the same as the min capacity

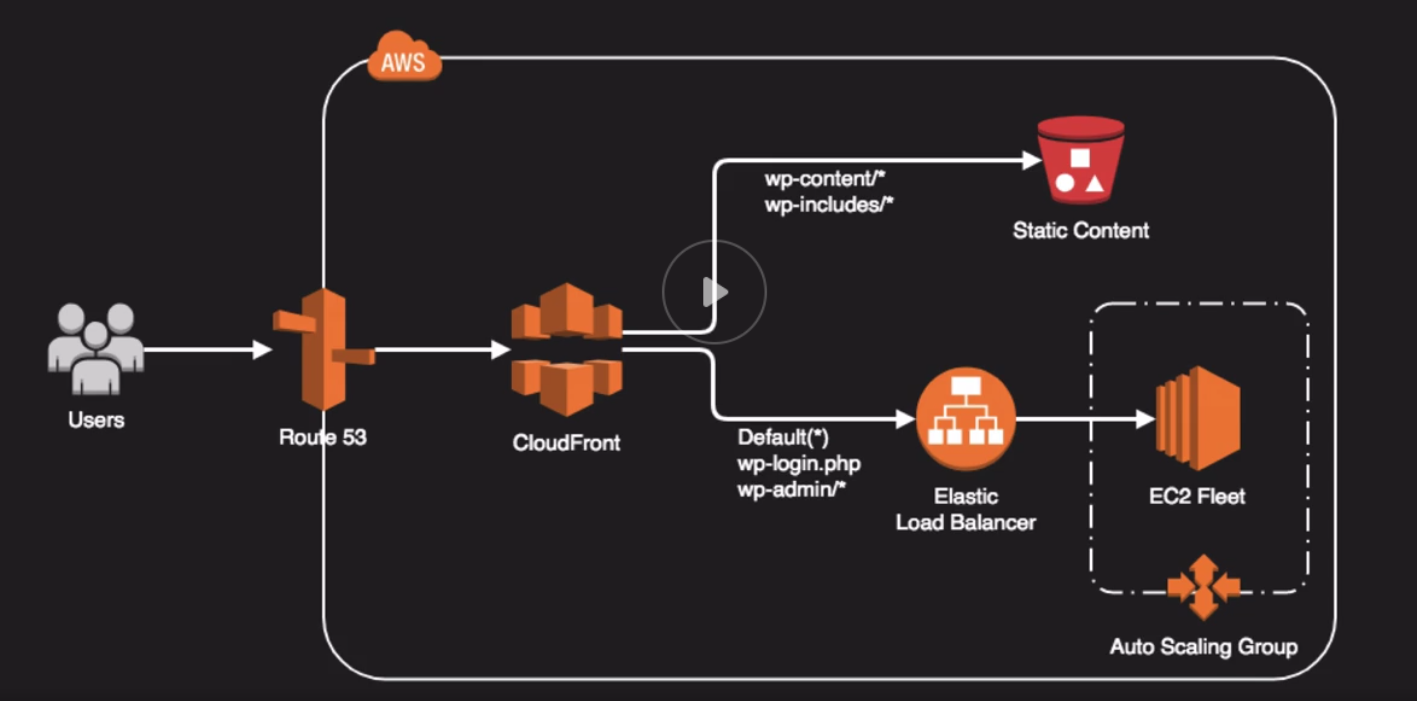
**CloudFront Part 2:**

* Can deliver content to your users faster by caching static and dynamic content at edge locations
* Dynamic content delivery is achieved using HTTP cookies forwarded from your origin
* Supports Adobe Flash Media Server’s RTMP protocol but you have to choose RTMP delivery methods
* Web distributions also support media streaming and live streaming but use HTTP or HTTPS

**Origins:**

* S3, EC2, ELB, or another web server
* Multiple origins can be configured
  + Use Behaviors to configure severing up origin content based on URL path

An example for using Behaviors would be having CloudFront route static content to an S3 bucket and route dynamic content to your fleet of web servers.



**Invalidation Requests:**

* Simply delete the file from the origin and wait for the TTL to expire
* Use the AWS Console to request invalidation for all content or a specific path such as /images/\*
* You can use the CloudFront API to submit an invalidation request
* Use 3rd party tools to perform CloudFront invalidations

Support Zone Apex Domain Names.

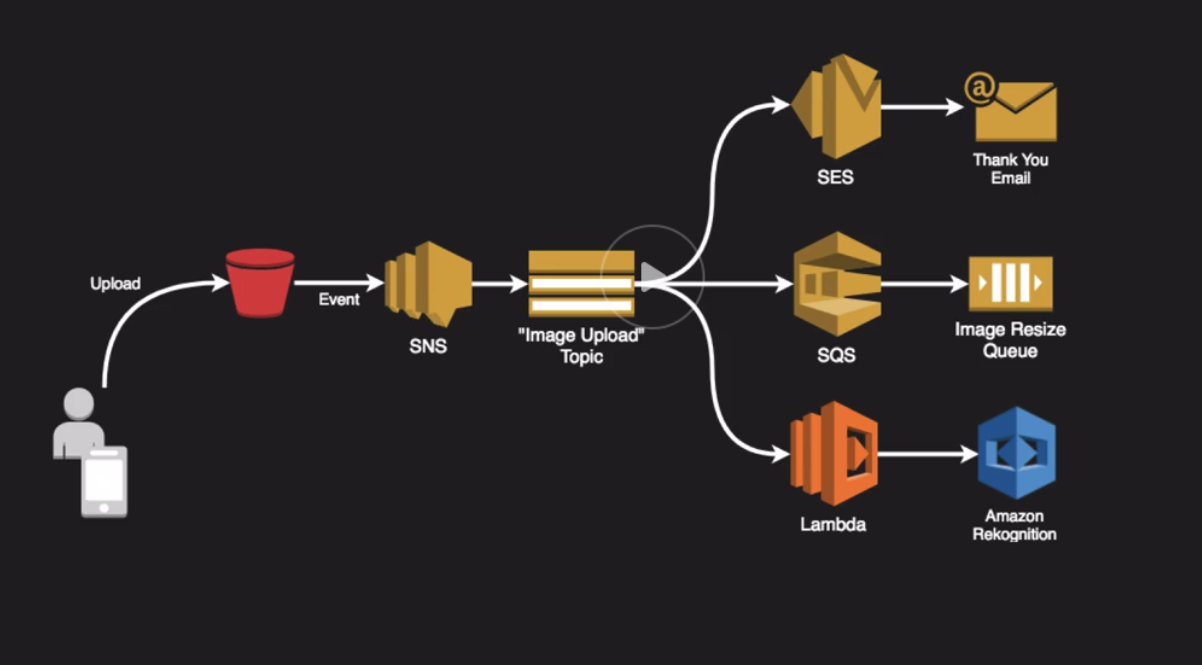
Also supports Geo-Restriction.

**SNS:**

* Enables pub/sub design patterns
* Topics- A channel for publishing a notification
* Subscription- Configuring an endpoint to receive messages published on the topic
* Endpoint Protocols- HTTP, HTTPS, Email, SMS, SQS, Amazon Device Messaging (Push Notifications), and Lambda

**Fan Out Architecture:**

* Very useful for doing things in parallel
  + Parallel- Doing things in unison
* SNS is very useful when you have several processes that need to be ran in parallel
* A good way to achieve a loosely coupled architecture



**SQS:**

* Reliable, highly-scalable, hosted message queuing service
* Available integrations with KMS for encrypted messaging
* Transient storage-default is 4 days, with up to 14 days
* Optionally supports First-In First-Out queue ordering
* Max message size of 256KB but using a special Java SQS SDK, you can have messages as large as 2 GB
* Benefits more loosely coupled architectures

**Standard vs FIFO Queue:**

**Standard:**

* No assurances that messages will enter or leave the queue based on when they arrive
* Presents risk for an ongoing process

**FIFO:**

* Maintains the order in which messages are received
* Enforcing order processing means if a message fails or gets stuck it will hold up all other messages

**Amazon MQ:**

* Managed implementation of Apache ActiveMQ
  + Active MQ is a message broker
* Fully managed and highly available within a region
* Supports ActiveMQ API and support for JMS, NMS, MQTT, WebSocket
* Designed to be a drop-in-replacement for on-premise message brokers
* Use SQS if you are creating a new app from scratch

**Lambda:**

* Allows you to run code on-demand without the need for infrastructure
* Supports Node.js, Python, Java, Go, and C#
* Extremely useful option for creating serverless architectures
* Code is stateless and executed on an event basis (SNS, SQS, S3, DynamoDB Streams etc..)
* No fundamental limits to scaling a function since AWS dynamically allocates capacity in relation to events
* Works in a fanout architecture

Dead letter queue- For messages that have an error or cannot be processed.

**Simple Workflow Service:**

* Managed status tracker
* Create distributed asynchronous systems as workflows
* Supports both sequential and parallel processing
* Tracks the state of your workflow which you interact and update via API
* Best suited for human-enabled workflows like an order fulfillment or procedural requests
* AWS recommends new applications to use Step Functions over SWF

**Activity Worker-** A program that interacts with the AWS SWF service to get tasks, process tasks, and return results.

**Decider-** A program that controls coordination of tasks such as their ordering, concurrency, and scheduling.

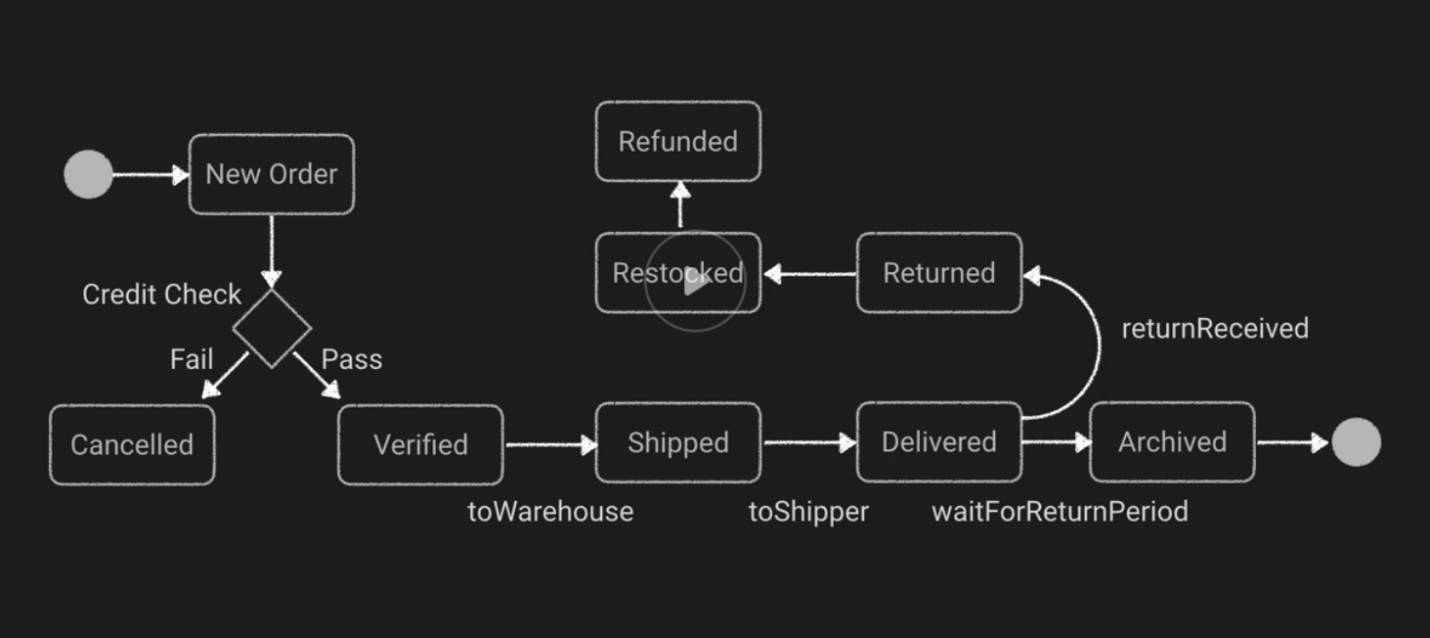
**Step Functions and Batch:**

**Step Functions:**

* Managed workflow and orchestration platform
* Scalable and highly available
* Define your app as a state machine
  + State machine: Create tasks, sequential steps, parallel steps, branching paths, or timers
* Amazon State Language declarative JSON
* Apps can interact and update the stream via Step Function API
* Step Functions has a visual interface that helps describe flow and real-time status
* Detailed logs of each step execution

**Finite State Machine:**

* Allows an object to assume different state of status during a process

****

**AWS Batch:**

* Management tool for creating, managing, and executing batch-oriented tasks using EC2 instances
* Steps:
  + Create a Compute Environment: Managed or Unmanaged, Sport or On-Demand. How much CPUs
  + Create a Job Queue with priority and assigned to a Compute Environment
  + Create Job Definition: Script or JSON, environment variables, Mount Points, IAM Role, Container Image, ETC.
  + Schedule the Job

**Comparisons:**

**Step Functions:**

* When: Out-of-the-box coordination of AWS service components
* Use Case: Order Processing Flow

**Simple Workflow Service:**

* When: Need to support external processes or specialized execution logic
* Use Case: Loan application process with manual review steps
* For newer apps try to look at Step Functions

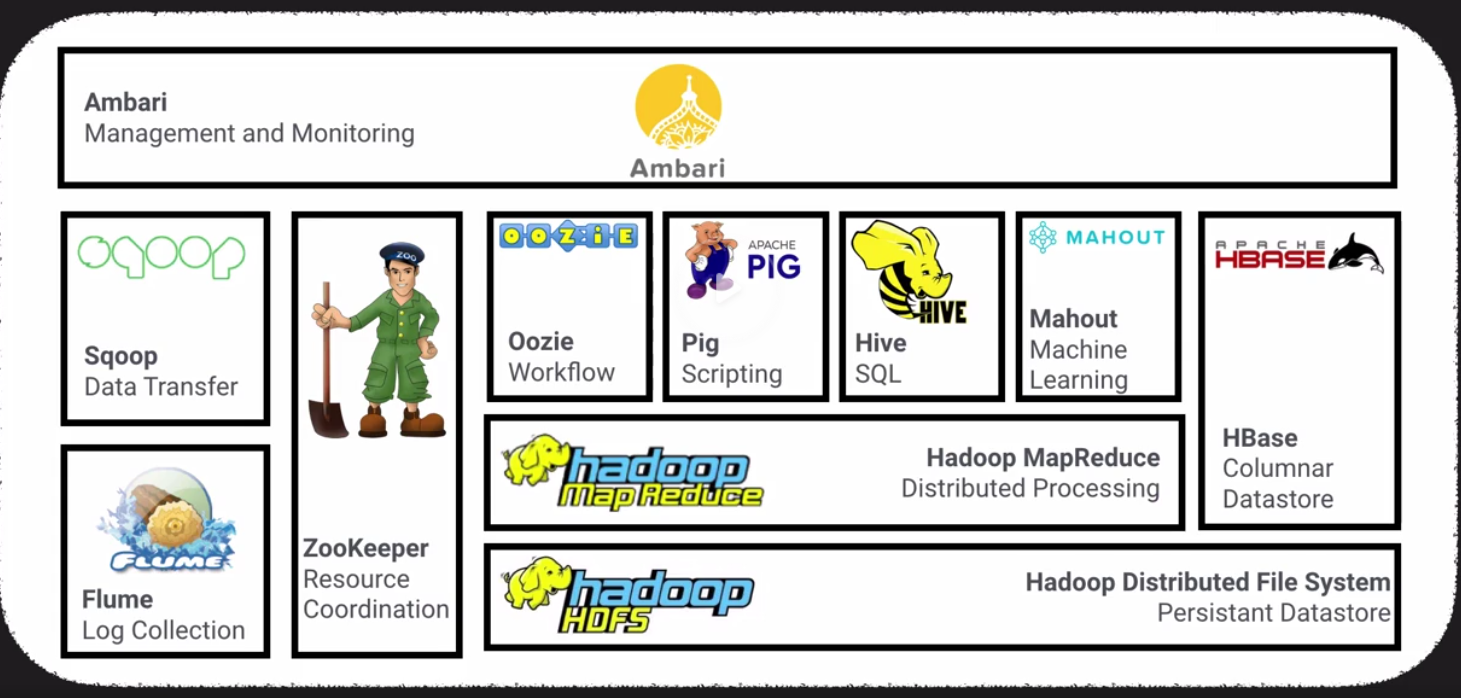
**Simple Queue Service:**

* When: Messaging Queue; Store and forward patterns
* Use Case: Image Resize Process

**AWS Batch:**

* When: Scheduled or reoccurring tasks that do not require heavy logic
* Use Case: Rotate logs daily on firewall appliance

**Elastic MapReduce:**

* Is not a single product, but rather a collection of products
* Hadoop HDFS- Persistent file system where the data gets stored in
* Hadoop Map Reduce- Distributed Processing
* Managed Hadoop framework for processing huge amount of data
* Also supports Apache spark, HBase, Presto, and Flink
* Most commonly used for log analysis, financial analysis, or ETL activities

**Step-** a programmatic task for preforming some process on the data

**Cluster-** a collection of EC2 instances provisioned by EMR to run your steps

**Components of EMR:**

**Master Node-**

**Core Node-** Has the HDFS storage so it can store your persistent data

**Task Node-** Worker nodes with ephemeral storage. There to work on the steps. Scale out Task nodes to work on many TB of data

**Extra Notes:**

How big can SNS messages be?

SNS messages can be anywhere from 64 to 256KB.

How big can Kinesis Data records be?

1MB is the max.

How big can SQS messages be?

256KB, but can be extended to up to 2GB using the SQS client library for Java.

How do Step Scaling Policies work? Do they have a cooldown period?

Step scaling policies are where you define certain number of steps that your fleet of EC2 instances have to fully reach before scaling.

They do not have a cooldown period you can define because in theory these steps are granular and should be triggered based on specific scenarios that need your environment to react in a timely fashion to scaling up or down.

What is a Warmup period and where does it apply?

This is a period of time you can define for your Auto-Scaling groups to wait before using new instances in your metric calculations for the basis of triggering additional scaling actions.

You cannot edit a Launch Configuration.