**4:01**

**25. AWS Serverless: SAM - Serverless Application Model**

383. SAM Overview

sam - cli debugging

* locally build, test and debug your serverless applications that are defined using aws sam templates
* provides a lambda like execution environment locally
* sam cli + aws toolkits => step-through and debug your code
* supported IDEs: aws cloud9, visual studio code, jetbrains, pycharm, intellij
* aws toolkits: IDE plugins which allows you to build, test, debug, deploy and invoke lambda functions built using aws sam

**0:58**

**25. AWS Serverless: SAM - Serverless Application Model**

383. SAM Overview

aws sam

* sam = serverless application model
* framework for developing and deploying serverless applications
* all the configuration is yaml code
* generate complex cloudformation from simple sam yaml file
* supports anything from cloudformation: outputs, mappings, parameters, resources
* only two commands to deploy to aws
* sam can use codedeploy to deploy lambda functions
* sam can help you to run lambda api gateway, dynamodb locally

**1:35**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

381. CodeGuru - Agent Configuration

amazon codeguru - agent configuration

* maxstackdepth -  the maximum depth of the stacks in the code that is represented in the profile
  + example : if codeguru profiler finds a method A, which calls method B, which calls method c, which calls method D then the depth is 4
  + if the maxstackdepth is set to 2, then the profiler evaluates a and b
* memoryusagelimitpercent - the memoty percentage used by the profiler
* minimumtimeforreportinginmilliseconds - the minimum time between sending reports (milliseconds)
* reportingintervalinmilliseconds - the reporting interval used to report profiles (milliseconds)
* samplingintervalinmilliseconds - the sampling interval that is used to profile samples (milliseconds)
  + reduce to have a higher sampling rate

**3:13**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

380. CodeGuru - Overview

amazon codeguru profiler

* helper understand the runtime behaviour of your application
* example: identify if your application is consuming excessive cpu capacity on a logging routine
* features:
  + identify and remove code inefficiencies
  + improve application performance
  + decrease compute costs
  + provides heap summary (identify which objects using up memory)
  + anomaly detection
* support applications running on aws or on-premise
* minimal overhead on application

**2:24**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

380. CodeGuru - Overview

amazon codeguru reviewer

* identify critical issues, security vulnerabilities , and hard-to-find bugs
* example: common coding best practices, resource leaks, security detection, input validation
* uses machine learning and automated reasoning
* hard-learned lessons across millions of code reviews on 1000s of open- source and amazon repositories
* supports java and python
* integrates with github, bitbucket, and aws codecommit

**1:27**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

380. CodeGuru - Overview

amazon codeguru

* an ML - powered service for automated code reviews and application performance recommendations
* provides two functionalities
  + codeguru reviewer: automated code reviews for static code analysis (development)
  + codeguru profiler: visibility/recommendations about application performance during runtime (production)

**5:56**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

378. CodeArtifact - Overview

codeartifact - resource policy

* can be used to authorize another account to access codeartifact
* a given principal can either read all the packages in a repository or none of them

**1:00**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

378. CodeArtifact - Overview

aws codeartifact

* software packages depend on each other to be built (also called code dependencies), and new ones are created
* storing and retrieving these dependencies is called artifact management
* traditionally you need to setup your own artifact management system
* codeartifact is a secure, scalable and cost-effective artifact management for software development
* works with common dependency management tools such as maven, gradle, npm, yarn, twine, pip, and NuGet
* developers and codebuild can then retrieve dependencies straight from codeartifact

**2:15**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

375. CodeDeploy Overview

codedeploy - ec2/on-premises platform

* can deploy to ec2 instances & on-premises servers
* perform in-place deployments or blue/green deployments
* must run the codedeploy agent on the target instances
* define deployment speed
  + allatonce: most downtime
  + halfatatime: reduced capacity by 50%
  + oneatatime: slowest, lowest availability impact
  + custom: define your %

**1:09**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

375. CodeDeploy Overview

aws codedeploy

* deployment service that automates application deployment
* deploy new applications versions to EC2 instances, on-premises servers, Lambda functions, ECS services
* Automated Rollback capability in case of failed deployments, or trigger cloudwatch alarm
* gradual deployment control
* a file named appspec.yml defines how the deployment happens
* AWS CodeDeploy is a fully managed deployment service that automates software deployments to a variety of computing services such as EC2, Fargate, Lambda, and your on-premises servers. You can define the strategy you want to execute such as in-place or blue/green deployments.

**4:16**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

372. CodeBuild Overview

codebuild - buildspec.yml

* buildspec.yml file must be at the root of your code
* env - define environment variables
* phases - specify commands to rn:
* artifacts - what to upload to s3 (encrypted with KMS)
* cache - files to cache (usually dependencies) to s3 for future build speedup
* You can configure CodeBuild to run its build containers in a VPC, so they can access private resources in a VPC such as databases, internal load balancers

**0:57**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

372. CodeBuild Overview

aws codebuild

* source - codecommit, s3, Bitbucket, Github
* build instructions: code file buildspec.yml or insert manually in console
* output logs can be stored in amazon  s3 & cloudwatch logs
* use cloudWatch metrics to monitor build statistics
* use eventbridge to detect failed builds and trigger notifications
* use cloudwatch alarms to notify if you need thresholds for failures
* build projects can be defined within codepipeline or codebuild
* AWS CodeBuild is a fully managed continuous integration (CI) service that compiles source code, runs tests, and produces software packages that are ready to deploy. It is an alternative to Jenkins.
* CodeBuild containers are deleted at the end of their execution (success or failure). You can't SSH into them, even while they're running.
* CodeBuild can run any commands, so you can use it to run commands including build a static website and copy your static web files to an S3 bucket.

**2:58**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

369. CodePipeline Overview

codepipeline - artifacts

* each pipeline stage can create artifacts
* artifacts stored in an s3 bucket and passed on to the next stage

**3:52**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

369. CodePipeline Overview

CodePipeline - troubleshooting

* for codepipeline pipeline/action/stage execution state changes
* use CloudWatch events (amazon eventBridge). Example:
  + you can create events for failed pipelines
  + you can create events for cancelled stages
* if codepipeline fails a stage, your pipeline stops, and you can get information in the console
* if pipeline can't perform an action, make sure the "IAM service role" attached does have enough IAM permissions (IAM policy)
* AWS CloudTrail can be used to audit AWS API calls

**1:25**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

369. CodePipeline Overview

* visual workflow to orchestrate your cicd
* AWS CodePipeline is a fully managed continuous delivery (CD) service that helps you automate your release pipeline for fast and reliable application and infrastructure updates. It automates the build, test, and deploy phases of your release process every time there is a code change. It has direct integration with Elastic Beanstalk.
* sosurce - codecommit, ecr, s3, bitbucket, github
* build - codebuild, jenkins, cloudBees, teamcity
* test - codeBuild, aws device farm, 3rd party tools, ..
* deploy - codeDeploy, elastic beanstalk, cloudFormation, ECS , s3,...
* invooke - lambda, sstep functions
* consists of stages:
  + each stage can have sequential actions ans/or parallel actions
  + example: build->test->deploy-> load testing -> ...
  + manual approval can be defined at any stage

**3:09**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

366. CodeCommit Overview

codecommit - security

* interactions are done using Git (standard)
* authentication
  + ssh keys - aws users can configure ssh keys in their iam console
  + https - with aws cli credential helper or git credentials for IAM user
* authorization
  + IAM policies to manage users/roles permissions to repositories
* encryption
  + repositories are automatically encrypted at rest using AWS KMS
  + encrypted inn transit (can only use HTTPS or SSH - both secure)
* cross-account access
  + do not share your ssh keys or your aws credentials
  + use an IAM role in your aws account and use aws sts (AssumeRole API)

**0:56**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

366. CodeCommit Overview

aws codecommit

* version control is the ability to understand the various changes that happened to the code over time (and possibly roll back)
* all these are enabled by using a version control system such as git
* a git repository can be synchronised on your computer, but it usually is uploaded on a central online repository
* AWS CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories. It is an alternative to GitLab and GitHub.
* benefits are:
  + collaborate with other developers
  + make sure the code is backed-up somewhere
  + make sure it's fully viewed and auditable

**1:53**

**24. AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy**

365. Introduction to CICD in AWS

CICD - introduction

* this section is all about automating the deployment we've done so far while adding increased safety
* we'll learn about:
  + AWS CodeCommit - stroing our code
  + aws codepipeline - automating our pipeline from code to Elastic beanstalk
  + aws codebuild - buildng and testing our code
  + aws codedeploy - deploying the code to ec2 instances (not elastic beanstalk)
  + aws codestart - manage software development activities in one place
  + aws codeArtifact - store, publish, and share software packages
  + aws codeGuru - automated code reviews using machine learning

**1:42**

**23. AWS Serverless: API Gateway**

363. API Gateway - Architecture

api gateway - architecture

* create a single interface for all the microservices in your company
* use api endpoints with various resources
* apply a simple domain name and ssl certificates
* can apply forwarding and transformation rules at the api gateway level

**6:18**

**23. AWS Serverless: API Gateway**

362. API Gateway Websocket API

api gateway - websocket api - routing

* incoming json messages are routed in different backend
* if no routes => sent to $default
* you request a route selection expression to select the field on JSON to route from
* sample expression: $request.body.action
* the result is evaluated against the route keys available in your API gateway
* the route is then connected to the backend you've setup through API gateway

**1:23**

**23. AWS Serverless: API Gateway**

361. API Gateway REST API vs HTTP API

api gateway - http api vs rest api

* http api
  + low-latency, cost-effective aws lambda proxy, http proxy apis and private integration (no data mapping)
  + support OIDC and OAuth 2.0 authorization , and built-in support for CORS
  + no usage plans and API keys
* rest api
  + all features (except native OpenID connect/OAuth 2.0)

**6:35**

**23. AWS Serverless: API Gateway**

359. API Gateway Authentication and Authorization

api gateway - security - summary

* custom authorizer:
  + great for 3rd party tokens
  + very flexible in terms of what IAM policy is returned
  + handle authentication verification + authorization in the lambda function
  + pay per lambda invocation, results are cached
* cognito user pool:
  + you manage your own user pool (can be backed by facebook, google login etc...)
  + no need to write any custom code
  + must implement authorization in the backend

IAM:

* great for users/roles already within your aws account, + resource policy for cross account
* handle authentication + authorization
* leverages signature v4

**5:34**

**23. AWS Serverless: API Gateway**

359. API Gateway Authentication and Authorization

api gateway - security

lambda authorizer (formerly custom authorizers)

* token-based authorizer (bearer token) - ex JWT (JSON web token) or oauth
* a request parameter-based lambda authorizer (headers, query string, stage var)
* lambda must return an IAM policy for the user, result policy is cached
* authentication = external  | authorization = lambda function

**3:35**

**23. AWS Serverless: API Gateway**

359. API Gateway Authentication and Authorization

api gateway - security

cognito  user pools

* cognito fully manages user lifecycle , token expires automatically
* api gateway verifies identity automatically from aws cognito
* no custom implementation required
* Authentication=cognito user pools | authorization = api gateway methods

**2:14**

**23. AWS Serverless: API Gateway**

359. API Gateway Authentication and Authorization

api gateway - resource policies

* resource policies (similar to lambda resource policy)
* allow for cross account access (combined with IAM security)
* allow for a specific source IP address
* allow for a VPC endpoint

**1:24**

**23. AWS Serverless: API Gateway**

359. API Gateway Authentication and Authorization

api gateway - security

IAM permissions

* create an IAM policy authorization and attach to user/role
* authentication=IAM  | authorization = IAM policy
* good to provide access within AWS (ec2, lambda, IAM users....)
* leverages "Sig v4" capability where IAM credential are in headers

**5:03**

**23. AWS Serverless: API Gateway**

357. API Gateway Monitoring, Logging and Tracing

API gateway - errors

* 4xx means Client errors
  + 400: bad request
  + 403:access denied, WAF filtered
  + 429:Quota exceeded, throttle
* 5xx means server errors
  + 502:bad gateway exception, usually for an incompatible output returned from a lambda proxy integration backend and occasionally for out-of-order invocations due to heavy loads
  + 503: service unavailable exception
  + 504:integration failure - ex endpoint request timed-out exception
  + API gateway requests time out after 29 second maximum

**4:01**

**23. AWS Serverless: API Gateway**

357. API Gateway Monitoring, Logging and Tracing

api gateway throttling

* account limit
* in case of throttling => 429 too many requests (retriable error)
* can set stage limit & method limits to improve performance
* or you can  define usage plans to throttle per customer
* just like lambda concurrency, one API that is overloaded, if not limited, can cause the other APIs to be throttled

**2:59**

**23. AWS Serverless: API Gateway**

357. API Gateway Monitoring, Logging and Tracing

api gateway - cloudwatch metrics

* metrics are by stage, possibility to enable detailed metrics
* CacheHitCount & CacheMissCount: efficiency of the cache
* Count: the number API requests in a given period
* IntegrationLatency: the time between when API gateway relays a request to the backend and when it receives a response from the backend
* Latency: the time between when API gateway receives a request from a client and when it returns a response to the client. the latency includes the integration latency and other API gateway overhead
* 4XXError (client-side) & 5XXError (server-side)

**1:14**

**23. AWS Serverless: API Gateway**

357. API Gateway Monitoring, Logging and Tracing

api gateway - logging & tracing

* cloudwatch logs
  + log contains information about request/response body
  + enable cloudwatch logging at the stage level (with log level - error, debug , info)
  + can override settings on a per api basis
* x-ray
  + enable tracing to get extra information about requests in api gateway
  + x-ray api gateway + aws lambda gives you the full picture

**1:14**

**23. AWS Serverless: API Gateway**

356. API Gateway Usage Plans & API Keys

api gateway - usage plans & API keys

* if you want to make an api available as an offering ($) to your customers
* usage plan:
  + who can access one or more deployed api stages and methods
  + how much and how fast they can access them
  + uses api keys to identify api clients and meter access
  + configure throttling limits and quota limits that are enforced on individual client
* api keys:
  + alphanumeric string values to distribute to your customers
  + Ex: WBiHXtoAb4WPKBC&
  + can use with usage plans to control access
  + throttling limits are applied to the api keys
  + quotas limits is the overall number of maximum requests

**1:57**

**23. AWS Serverless: API Gateway**

355. API Gateway Caching

api gateway cache invalidation

* able to flush the entire cache immediately
* clients can invalidate the cache with header: chache-control: max-age=0 (with proper IAM authorization)
* if you don't impose an invalidateCache policy (or choose the require authorization check box in the console), any client can invalidate the API cache

**1:17**

**23. AWS Serverless: API Gateway**

355. API Gateway Caching

caching api responses

* caching reduces the number of calls made to the backend
* default TTL (time to live) is 300 seconds (min:0s, max:3600s)
* caches are defined per stage
* possible to override cache settings per method
* cache encryption option
* cache capacity between 0.5gb to 237gb
* cache is expensive, makes sense in production, may not make sense in dev/test

**2:05**

**23. AWS Serverless: API Gateway**

353. API Gateway Open API

rest api - request validation

* you can configure api gateway to perform basic validation of an api request before proceeding with the integration request
* setup request validation by import open API definition file
* when the validation fails, api gateway immediately fails the request
  + returns a 400-error response to the caller
* this reduces unnecessary calls to the backend
* checks:
  + the required request parameters in the URI , query string, and headers of an incoming request are included and non-blank
  + the applicable request payload adheres to the configured json schema request model of the method

**1:13**

**23. AWS Serverless: API Gateway**

353. API Gateway Open API

api gateway - open api spec

* common way of defining REST APIs, using api definition as code
* import existing  openapi 3.0 spec to api gateway
  + method
  + method request
  + integration request
  + method response
  + + AWS extensions for api gateway and setup every single option
* can export current api as openapi spec
* openAPI specs can be written in yaml or json
* using openAPI we can generate sdk for our applications

**5:03**

**23. AWS Serverless: API Gateway**

351. API Gateway Integration Types & Mappings

mapping example: json to xml with soap

* soap api are xml based, whereas rest api are json based
* in this case, api gateway should:
  + extract data from the request: either path, payload or header
  + build soap message based on request data (mapping template)
  + call soap service and receive xml response
  + transform xml response to desired format (like json), and respond to the user

**3:21**

**23. AWS Serverless: API Gateway**

351. API Gateway Integration Types & Mappings

api gateway - integration types

* integration type HTTP\_PROXY
  + no mapping template
  + the http request is passed to the backend
  + the http response from the backend is forwarded by api gateway
  + possibility to add http headers if need be (ex: api key)

**2:21**

**23. AWS Serverless: API Gateway**

351. API Gateway Integration Types & Mappings

api gateway - integration types

* integration type AWS\_PROXY (Lambda proxy):
  + incoming request from the client is the input to lambda
  + the function is responsible for the logic of request/response
  + no mapping template, headers, query string parameters .. are passed as arguments

**1:14**

**23. AWS Serverless: API Gateway**

351. API Gateway Integration Types & Mappings

api gateway - integration types

* integration type MOCK
  + API gateway returns a response without sending the request to the backend
* integration type HTTP/AWS (Lambda & AWS services)
  + you must configure both the integration request and integration response
  + setup data mapping using mapping templates for the request & response

**4:09**

**23. AWS Serverless: API Gateway**

351. API Gateway Integration Types & Mappings

mapping templates (AWS & http integration)

* mapping templates can be used to modify request/responses
* rename/modify query string parameters
* modify body content
* add headers
* uses velocity template language (VTL): for loop , if etc...
* filter output results (remove unnecessary data)
* content-type can be set to application/json or application/xml

**6:36**

**23. AWS Serverless: API Gateway**

344. API Gateway Overview

api gateway - security

* user authentication through
  + iam roles (useful for internal applications)
  + cognito (identity for external users - example mobile users)
  + custom autorizer (your own logic) - this is lambda
* custom domain name https security through integration with aws certificate manager (acm)
  + if using edge-optimized endpoint, then the certificate must be in us-east-1
  + if using regional endpoint, the certificate must be in the API gateway region
  + must setup CNAME or A-alias record in route 53

**2:54**

**23. AWS Serverless: API Gateway**

346. API Gateway Stages and Deployment

api gateway - stage variables

* stage variables are like environment variables for API gateway
* use them to change often changing configuration values
* they can be used in:
  + lambda function ARN
  + HTTP endpoint
  + parameter mapping templates
* use cases:
  + configure HTTP end points your stages talk to (dev,test, prod)
  + pass configuration parameters to aws lambda through mapping templates
* stage variables are paased to the context object in aws lambda
* Formate:${stageVariables.variableName}
* To redirect your API Gateway stage to the correct AWS Lambda Alias

**0:45**

**23. AWS Serverless: API Gateway**

346. API Gateway Stages and Deployment

api gateway - deployment stages

* making changes in the api gateway does not mean they're effective
* you need to make a deployment for them to be in effect
* it's a common source of confusion
* changes are deployed to stages (as many as you want)
* use the naming you like for stages (dev, test, prod)
* each stage has its own configuration parameters
* stages can be rolled back as a history of deployments is kept

**5:26**

**23. AWS Serverless: API Gateway**

344. API Gateway Overview

api gateway - endpoint types

* edge-optimized (default): for global clients
  + requests are routed through the cloudfront edge locations (improves latency)
  + the API gateway still lives in only one region
* regional:
  + for clients within the same region
  + could manually combine with cloudfront (more control over the caching strategies and the distribution)
* private:
  + can only be accessed from your VPC using an interface VPC endpoint (ENI)
  + use a resource policy to define access

**3:22**

**23. AWS Serverless: API Gateway**

344. API Gateway Overview

api gateway - integrations high level

* lambda function
  + invoke lambda function
  + easy way to expose REST API backed by aws lambda
* http
  + expose http endpoints in the backend
  + example: internal http api on premise, application load balancer...
  + why? add rate limiting, caching, user authentications, API keys, etc...
* aws service
* expose any aws api through the api gateway?
* example: start an aws step function workflow, post a message to sqs
* why ? add authentication, deploy publicly, rate control....

**2:18**

**23. AWS Serverless: API Gateway**

344. API Gateway Overview

aws api gateway

* aws lambda + API gateway: no infrastructure to manage
* support for the websocket protocol
* handle API versioning (v1,v2,....)
* handle different environments (dev, test,prod,....)
* handle security (authentication and authorization)
* create API keys, handle request throttling
* swagger/ open api import to quickly define APIs
* transform and validate requests and responses
* generate SDK and API specifications cache API responses

**3:25**

**22. AWS Serverless: DynamoDB**

342. DynamoDB Security & Other

dynamodb - fine-grained access control

* using web identity federation or cognito identity pools, each user gets aws credentials
* you can assign an IAM role to these users with a condition to limit their API access to dynamodb
* leadingkeys- limit row-level access for users on the primary key
* attributes - limit specific attributes the user can see

**1:17**

**22. AWS Serverless: DynamoDB**

342. DynamoDB Security & Other

dynamodb - security & other features

* security
  + VPC endpoints available  to access dynamodb without using the internet
  + access fully controlled by IAM
  + encryption at rest using aws kms and in-transit using ssl/TLS
* backup and restore feature available
  + point- in -time recovery PITR like RDS
  + no performance impact
* gobal tables
  + multi-region, multi-active, fully replicated , high performance
* dynamodb local
  + develop and test apps locally without accessing the dynamodb web service (without internet)
* aws data base migration service (aws dms) can be used to migrate to dynamodb (from mongodb, oracle, mysql, s3,...)

**1:50**

**22. AWS Serverless: DynamoDB**

341. DynamoDB Operations

dynamodb operations

* table cleanup
  + option 1 : scan + deleteItem
    - very slow, consumes RCU & WCU, expensive
  + option2 : drop table + recreate table
    - fast, efficient checp
* copying a dynamodb table
  + option1 : using aws data pipeline
  + option 2: backup and restore into a new table
    - takes some time
  + option 3: scan + PutItem or BatchWriteItem
    - write your own code

**2:42**

**22. AWS Serverless: DynamoDB**

340. DynamoDB Patterns with S3

dynamodb - indexing s3 objects metadata

**1:50**

**22. AWS Serverless: DynamoDB**

339. DynamoDB Conditional Writes, Concurrent Writes & Atomic Writes

dynamodb - writes types

* concurrent writes
* conditional writes
* atomic writes
* batch writes

**1:17**

**22. AWS Serverless: DynamoDB**

338. DynamoDB Partitioning Strategies

dynamoDB write sharding

* imagine we have a voting application with two candidates, candidate A and candidate B
* if partition key is candidate\_id, this results into two partitions, which will generate issues
* a strategy that allows better distribution of items evenly across partitions
* add a suffix to partition key value
* two methods:
  + sharding using random suffix
  + sharding using calulated suffix

**2:12**

**22. AWS Serverless: DynamoDB**

337. DynamoDB Session State

dynamodb as session state cache

* its common to use dynamoDB to store session state
* vs  elasticache:
  + elasticache is in-memory, but dynamodb is serverless
  + both are key/value stores
* vs EFS
  + efs must be attached to ec2 instances as network drive
* vs ebs & instance store:
  + ebs & instance can only be used for local caching, not shared caching
* vs s3:
  + s3 is higher latency, and not meant for small objects

**3:37**

**22. AWS Serverless: DynamoDB**

336. DynamoDB Transactions

dynamodb transactions - capacity computations

* important for the exam!
* example1: 3 transactional writes per second, with item size 5 kb
  + we need 3\*(5kb/1kb)\*2(transactional cost)=30 wcu
* example2: 5 transaction reads per second, with item size 5kb
  + we need 5\*(8kb/4kb)\*2(transactional cost) = 20RCU
  + 5 gets rounded to the upper 4 kb

**4:23**

**22. AWS Serverless: DynamoDB**

332. DynamoDB Streams

dynamoDB streams & AWS Lambda

* you need to define an event source mapping to read from a dynamoDB streams
* you need to ensure the lambda function has the appropriate permissions
* your lambda function is invoked synchronously

**3:43**

**22. AWS Serverless: DynamoDB**

332. DynamoDB Streams

dynamodb streams

* ability to choose the information that will be written to the stream:
  + KEYS\_ONLY - only the key attributes of the modified item
  + NEW\_IMAGE - the entire item, as it appears after it was modified
  + OLD\_IMAGE - the entire item, as it appeared before it was modified
  + NEW\_AND\_OLD\_IMAGES - both the new and the old images of the item
* dynamoDB streams are made of shards, just like kinesis data streams
* you don't provision shards, this is automated by AWS
* records are not retroactively populated in a stream after enabling it
* DynamoDB Streams allows you to capture a time-ordered sequence of item-level modifications in a DynamoDB table. It's integrated with AWS Lambda so that you create triggers that automatically respond to events in real-time.

**1:20**

**22. AWS Serverless: DynamoDB**

332. DynamoDB Streams

dynamoDB streams

* ordered stream of item-level modifications (create/update/delete) in a table
* stream records can be:
  + sent to kinesis data streams
  + read by AWS Lambda
  + read by kinesis client library applications
* data retention for up to 24 hours
* use cases:
  + react to changes in real-time (welcome email to users)
  + analytics
  + insert into derivative tables
  + insert into opensearch service
  + implement cross-region replication

**1:41**

**22. AWS Serverless: DynamoDB**

330. DynamoDB DAX

dynamoDB accelerator (DAX)

* fully-managed, highly available, seamless in-memory cache for dynamodb
* microseconds latency for cached reads & queries
* doesn't require application logic modification (compatible with existing dynamoDB APIs)
* solves the "Hot Key" problem (too many reads)
* 5 minutes TTL for cache (default)
* up to 10 nodes in the cluster
* multi-AZ (3 nodes minimum recommended for production)
* secure (encryption at rest with KMS, VPC, IAM, cloudTrail,....)

**4:08**

**22. AWS Serverless: DynamoDB**

326. DynamoDB Indexes (GSI + LSI)

dynamodb - indexes and throttling

* global secondary index (GSI):
  + if the writes are throttled on the GSI, then the main table will be throttled!
  + even if the WCU on the main tables are fine
  + choose your GSI partition key carefully!
  + assign your WCU capacity carefully!
  + Global Secondary Index (GSI) uses an independent amount of RCU and WCU and if they are throttled due to insufficient capacity, then the main table will also be throttled.
* local secondary index (LSI):
  + uses the WCU and RCU of the main table
  + no special throttling considerations

**3:23**

**22. AWS Serverless: DynamoDB**

326. DynamoDB Indexes (GSI + LSI)

dynamodb - global secondary index (GSI)

* alternate primary key (HASH or HASH + RANGE) from the base table
* speed up queries on non-key attributes
* the index key consists of scalar attributes (String, Number, or binary)
* Attribute Projections - some or all the attributes of the base table (KEYS\_ONLY, INCLUDE, ALL)
* must provision RCU & WCU for the index
* can be added/modified after table creation

**1:35**

**22. AWS Serverless: DynamoDB**

326. DynamoDB Indexes (GSI + LSI)

dynamoDB - local secondary index (LSI)

* alternative sort key for your table (same partition key as that of base table)
* the sort key consists of one scalar attribute (String, Number, or Binary)
* Up to 5 local secondary indexes per table
* must be defined at table creation time
* Attribute projections - can contain some or all the attributes of the base table (KEYS\_ONLY, INCLUDE, ALL)

**5:30**

**22. AWS Serverless: DynamoDB**

325. DynamoDB - Conditional Writes

conditional writes - example of String comparisons

aws dynamodb delete-item \

--table-name ProductCatalog \

--key '{"Id" : { "N" : "456"} }' \

--condition-expression "begins\_with(Pictures.FrontView, :v\_sub)"  \

--expression-attribute-values file://values.json

values.json:

{

":v\_sub" : {

      "s" : "http://"

}

}

* begins\_with - check if prefix matches
* contains - check if string is contained in another string

**3:25**

**22. AWS Serverless: DynamoDB**

325. DynamoDB - Conditional Writes

conditional writes - example on delete item

* attribute\_exists
  + Opposite of attribute\_not\_exists
  + aws dynamodb delete-item \
  + --table-name ProductCatalog \
  + --key '{"Id": {"N": "456"}}'  \
  + --condition-expression "attribute\_exists(ProductReviews.OneStar)"
* attribute\_not\_exists
  + only succeds if the attributes doesn't exist yet (no value)
  + aws dynamodb delete-item  \
  + --table-name ProductCatalog \
  + --key '{"Id" :{ "N"  :"456"}}'  \
  + --condition-expression "attribute\_not\_exists(Price)"

**2:30**

**22. AWS Serverless: DynamoDB**

325. DynamoDB - Conditional Writes

conditional writes - example on update item:

aws dynamodb update-item \

--item-name ProductCatalog \

--key '{"Id" : { "N": "456" } }' \

--update-expression "SET Price = Price - :discount" \

--condition-expression "Price>:limit" \

--expression-attribute-values file://values.json

values.json:

{

":discount":{

           "N": "150"

          },

          ":limit": {

              "N":  "500"

          }

}

}

**1:16**

**22. AWS Serverless: DynamoDB**

325. DynamoDB - Conditional Writes

dynamoDB - conditional writes

* for PutItem, UpdateItem, DeleteItem, and BatchWriteItem
* you can specify a condition expression to determine which items should be modified:
  + attribute\_exists
  + attributes\_not\_exists
  + attribute\_type
  + contains(for string)
  + begins\_with (for string)
  + ProductCategory IN (:cat1,:cat2) and price between :low and :high
  + size (string length)
* Note: filter expression filters the results of read queries, while condition expressions are for write operations

**7:53**

**22. AWS Serverless: DynamoDB**

323. DynamoDB - Basic Operations

dynamodb - partiQL

* SQL - compatible query language for dynamoDB
* allows you to select , insert, update, and delete data in dynamodb using sql
* run queries across multiple dynamoDB tables
* Run PartiQL queries from:
  + AWS management console
  + NoSQL workbench for DynamoDB
  + dynamoDB APIs
  + AWS CLI
  + AWS SDK

**6:30**

**22. AWS Serverless: DynamoDB**

323. DynamoDB - Basic Operations

dynamoDB - batch operations

* allows you to save in latency by reducing the number of API calls
* operations are done in parallel for better efficiency
* part of a batch can fail; in which case we need to try afain for the failed items
* BatchWriteItem
  + up to 25 PutItem and/or DeleteItem in one call
  + up to 16MB of data written, up to 400KB of data per item
  + can't update items (use UpdateItem)
  + UnprocessedItems for failed write operations (exponential backoff or add WCU)
* BatchGetItem
  + return items from one or more tables
  + up to 100 items, up to 16MB of data
  + items are retrieved in parallel to minimize latency
  + UnprocessedKeys for failed read operations (exponential backoff or add RCU)

**3:48**

**22. AWS Serverless: DynamoDB**

323. DynamoDB - Basic Operations

dynamoDB - reading data (scan)

* scan the entire table and then filter out data (inefficient)
* returns up to 1 MB of data - use pagination to keep on reading
* consumes a lot of RCU
* limit impact using limit or reduce the size of the result and pause
* for faster performance, use parallel scan
  + Multiple workers scan multiple data segments at the same time
  + increases the throughput and RCU consumed
  + limit the impact of parallel scans just like you would for scans
* can use ProjectionExpression & FilterExpression (no changes to RCU)

**2:32**

**22. AWS Serverless: DynamoDB**

323. DynamoDB - Basic Operations

dynamoDB - reading data (query)

* query returns items based on:
  + keyconditionExpression
    - partition key value (must be = operator) -required
    - sort key value (=,<,<=,>,>=, between, begins with) - optional
  + FilterExpression
    - additional filtering after the query operation (before data returned to you)
    - use only with non-key attributes (does not allow HASH or RANGE attributes)
* returns:
  + the number of items specified in limit
  + or up to 1 MB of data
* ability to do pagination on the results
* can query table, a local secondary index, or a global secondary index

**11:02**

**22. AWS Serverless: DynamoDB**

321. DynamoDB WCU & RCU - Throughput

R/W capacity modes - on-demand

* read/writes automatically scale up/down with your workloads
* no capacity planning needed (WCU/RCU)
* unlimited WCU & RCU, no throttle, more expensive
* you're charged for reads/writes/ that you use in terms of RRU and WRU
* read request units (RRU) - throughput for reads (same as RCU)
* write request units (WRU) - throughput for writes (same as WCU)
* 2.5x more expensive than provisioned capacity (use with care)
* use cases: unknown workloads, unpredictable application traffic,...

**10:04**

**22. AWS Serverless: DynamoDB**

321. DynamoDB WCU & RCU - Throughput

dynamoDB - throttling

* if we exceed provisioned RCU or WCU, we get "ProvisionedThroughputExceededException"
* Reasons:
  + hot keys - one partition key is being read too many times (e.g., popular item)
  + hot partitions
  + very large items, remember RCU and WCU depends on size of items
* solutions:
  + exponential backoff when exception is encountered (already in SDK)
  + distributed partition keys as much as possible
  + if RCU issue, we can use dynamoDB accelerator (DAX)

**8:38**

**22. AWS Serverless: DynamoDB**

321. DynamoDB WCU & RCU - Throughput

dynamoDB - partitions internal

* data is stored in partitions
* partition keys go through  a hashing algorithm to know to which partition they go to
* to compute the number of partitions:
  + # of partitions (by capacity) = (RCU (total)/3000) + (WCU(total)/1000)
  + # of partitions (by size) =  total size/10gb
  + # of partitions = ceil (max(# of partitions (by capacity), # of partitions (by size)))
* WCU and RCU are spread evenly across partitions
* Remember RCUs and WCUs are spread across all the table's partitions.

**7:08**

**22. AWS Serverless: DynamoDB**

321. DynamoDB WCU & RCU - Throughput

dynamoDB - read capacity units (RCU)

* one read capacity unit (RCU) represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4KB in size
* if the items are larger than 4 kb, more RCU are consumed
* example 1: 10 strongly consistent reads per second, with item size 4KB
  + we need 10\*(4kb/4kb) = 10RUC
* example 2:16 eventually consistent reads per second, with item size 12kb
  + we need (16/2) \* (12kb/4kb) = we RCU
* example 3:10 strongly consistent reads per second, with item size 6kb
  + we need 10\*(8kb/4kb) = 20 RCU (we must round up 6kb to 8kb)

**5:28**

**22. AWS Serverless: DynamoDB**

321. DynamoDB WCU & RCU - Throughput

strongly consistent read vs eventually consistent read

* eventually consistent read (default)
  + if we read just after a write, it's possible we'll get some stale data because of replication
* strongly consistent read
  + if we read just after a write, we will get the correct data
  + set "ConsistentRead" parameter to True in API calls (GetItem, BatchGetItem, Query, Scan)
  + consumes twice the RCU

**3:44**

**22. AWS Serverless: DynamoDB**

321. DynamoDB WCU & RCU - Throughput

dynamoDB - write capacity units (WCU)

* one write capacity unit (wcu) represents one write per second for an item up to 1 kb in size
* if the items are larger than 1 kb, more wcu are consumed
* example 1: we write 10 items per second, with item size 2kb
  + we need 10 \* (2kb/1kb) = 20WCU
* example2:we write 6 items per second, with item size 4.5kb
  + we need 6\*(5kb/1kb) = 30WCU (4.5 gets rounded to the upper kb)
* example 3: we write 120 items per minute, with item size 2 kb
  + we need (120/60)\*(2kb/1kb) = 4 WCU

**4:15**

**22. AWS Serverless: DynamoDB**

319. DynamoDB Overview

dynamodb - basics

* dynamodb is made of tables
* each table has a primary key (must be decided at creation time)
* each table can have an infinite number of items (= rows)
* each item has attributes (can be added over time - can be null)
* maximum size of an item is 400kb
* RCU and WCU are decoupled, so you can increase/decrease each value separately.
* data types supported are:
  + scalar types - string, number, binary, Boolean , Null
  + documentTyptes - list, map
  + setTypes - string set, number set, binary set

**3:16**

**22. AWS Serverless: DynamoDB**

319. DynamoDB Overview

amazon dynamodb

* fully managed, highly available with replication across multiple AZs
* nosql database - not a relational database
* scales to massive workloads, distributed database
* millions of requests per seconds, trillions of rows, 100s of tb of storage
* fast and consistent in performance (low latency on retrieval)
* integrated with iam for security, authorization and administration
* enables event driven programming with dynamodb streams
* low cost and auto-scaling capabilities
* standard & infrequent access (IA) table class
* DynamoDB is serverless with no servers to provision, patch, or manage and no software to install, maintain or operate. It automatically scales tables up and down to adjust for capacity and maintain performance. It provides both provisioned (specify RCU & WCU) and on-demand (pay for what you use) capacity modes.

**2:27**

**22. AWS Serverless: DynamoDB**

319. DynamoDB Overview

Nosql databases

* nosql databases are non-relational databases and are distributed
* nosql databases include mongodb, dynamodb
* nosql databases do not support query joins (or just limited support)
* all the data that is needed for a query is present in one row
* nosql databases don't perform aggregations such as sum, avg,...
* nosql databases scale horizontally
* there's no right or wrong for nosql v sql , they just require to model the data differently and think about use queries differently

**1:20**

**22. AWS Serverless: DynamoDB**

319. DynamoDB Overview

traditional architecture

* traditional applications leverage RDBMS databases
* these databases have the sql query language
* strong requirements about how the data should be modeled
* ability to do query joins, aggregations, complex computations
* vertical scaling (getting a more powerful CPU/RAM/IO)
* horizontal scaling (increasing reading capability by adding EC2/RDS read replicas)

**1:42**

**21. AWS Serverless: Lambda**

316. Lambda Limits

aws lambda limits to know - per region

* execution:
  + memory allocation: 128MB - 10GB (1MB increments)
  + maximum execution time: 900 seconds (15 minutes)
  + environment variables (4kb)
  + disk capacity in the function container (in /tmp) : 512MB to 10GB
  + concurrency executions:1000 (can be increased)
  + Lambda's maximum execution time is 15 minutes. You can run your code somewhere else such as an EC2 instance or use Amazon ECS.
* deployment:
  + lambda function deployment size (compressed .zip): 50MB
  + size of uncompressed deployment (code + dependencies): 250MB
  + can use the /tmp directory to load other files at startup
  + size of environments variables: 4kb

**0:56**

**21. AWS Serverless: Lambda**

315. Lambda - CodeGuru Integration

lambda and codeguru profiling

* gain insights into runtime performance of your lambda functions using codeguru profiler
* codeguru creates a profiles group for your lambda function
* supported for java and python runtimes
* activate from AWS lambda console
* when activated, lambda adds:
  + codeguru profiler layer to your function
  + environment variables to your function
  + AmazonCodeGuruProfilerAgentAccess policy to your function

**3:50**

**21. AWS Serverless: Lambda**

313. Lambda Function URL

lambda - function URL security

* AuthType AWS\_IAM - IAM is used to authenticate and authorize requests
  + both principals identity-based policy & resource-based policy are evaluated
  + principal must have lambda:InvokeFunctionUrl permissions
  + same account- identity-based policy OR Resource-based policy as ALLOW
  + cross account - identity-based policy AND resource based policy as allow

**2:37**

**21. AWS Serverless: Lambda**

313. Lambda Function URL

lambda - function URL security

{

   "version": "2012-10-17",

    "statement":[

            {

               "effect":"allow",

               "principal":"\*",

               "action":"lambda:InvokeFunctionUrl",

               "Resource": "urn:aws:lambda:us-east-1:123456789012:function:my-function",

               "condition":{

                      "stringEquals":{

                             "lambda:functionUrlAuthType":"NONE"

                               }

                    }

              }

        ]

}

**2:04**

**21. AWS Serverless: Lambda**

313. Lambda Function URL

lambda - function URL security

* resource-based policy
  + authorize other accounts/ specific CIDR/ IAM principals
* cross-origin resource sharing (CQRS)
  + if you call your lambda function URL from a different domain

**1:21**

**21. AWS Serverless: Lambda**

313. Lambda Function URL

lambda - function URL

* dedicated HTTP(s) endpoint for your lambda function
* a unique URL endpoint is generated for you (never changes)
* invoke via a web browser, curl, postman, or any HTTP client
* access your function URL through the public internet only
* supports resource-based policies & CORS configurations
* can be applied to any function alias or to $LATEST (can't be applied to other function versions)
* create and configure using aws console or aws API
* throttle your function by using reserved concurrency

**2:49**

**21. AWS Serverless: Lambda**

312. Lambda and CodeDeploy

lambda & codedeploy - AppSpec.yml

* name (required) - the name of the lambda function to deploy
* Alias (required) - the name of the alias to the lambda function
* CurrentVersion (required) - the version of the lambda function traffic currently points to
* TargetVersion(required) - the version of the lambda function traffic is shifted to

**2:03**

**21. AWS Serverless: Lambda**

312. Lambda and CodeDeploy

lambda & codeDeploy:

* codeDeploy can help you automate traffic shift for lambda aliases
* feature is integrated within the SAM framework
* linear: grow traffic every N minutes until 100%
  + linear10PercentEvery3Minutes
  + linear10percentEvery10minutes
* canary: try X percent then 100%
  + canary10Percent5Minutes
  + Canary10Percent30Minutes
* AllAtOnce: immediate
* can create pre & post traffic hooks to check the health of the lambda function

**3:04**

**21. AWS Serverless: Lambda**

310. Lambda Versions and Aliases

aws lambda aliases

* aliases are pointers to lambda function versions
* we can define a dev, test, prod aliases and have them point at different lambda versions
* aliases are mutable
* aliases enable canary deployment by assigning weights to lambda functions
* aliases enable stable configuration of our event triggers/destinations
* aliases have their own ARNs
* aliases cannot reference aliases
* Every time you release a Lambda function version, it gets a new number and you have to manually update all the AWS resources linked to your function (e.g., event triggers). then use alias
* You have updated a Lambda function and created a new version. Now, you want to test out the new version and ensure it can sustain production traffic. You are risk-averse and don't want to take down your whole application. use lambda alias, point to the new and old versions and then assign weights

**1:11**

**21. AWS Serverless: Lambda**

310. Lambda Versions and Aliases

aws lambda versions

* when you work on a lambda function, we work on $LATEST
* when we're ready to publish a lambda function, we create a version
* versions are immutable
* versions have increasing version numbers
* versions get their own ARN (amazon resource name)
* version = code + configuration (nothing can be changes - immutable)
* each version of the lambda function can be accessed

**4:38**

**21. AWS Serverless: Lambda**

309. Lambda Container Images

lambda container images - best practices

* strategies for optimizing container images:
  + use aws-provided base images
    - stable, built on amazon linux 2, cached by lambda service
  + use multi-stage builds
    - build your code in larger preliminary images, copy only the artifacts you need in your final container image, discard the preliminary steps
  + build from stable to frequently changing
    - make your most frequently occurring changes as late in your dockerfile as possible
  + use a single repository for functions with large layers
    - ecr compares each layer of a container image when it is pushed to avoid uploading and storing duplicates
* use them to upload loarge lambda functions (up to 10GB)

**1:42**

**21. AWS Serverless: Lambda**

309. Lambda Container Images

lambda container images

* deploy lambda function as container images of up to 10gb from ecr
* pack complex dependencies, large dependencies in a container
* base images are available for python, node.js, java, .net, go , ruby
* base image must implement the lambda runtime api
* can create your own image as long as it implements the lambda runtime api
* test the containers locally using the lambda runtime interface emulator
* unified workflow to build apps

**5:31**

**21. AWS Serverless: Lambda**

303. Lambda Concurrency

cold starts & provisioned concurrency

* cold start:
  + new instance => code is loaded and code outside the handler run (init)
  + if the init is large (code, dependencies, SDK....) this process can take some time
  + first request served by new instances has higher latency than the rest
* provisioned concurrency:
  + concurrency is allocated before the function is invoked (in advance)
  + so the cold start never happens and all invocations have low latency
  + application auto scaling can manage concurrency (schedule or target utilization)
* note:
  + cold starts in vpc have been dramatically reduced in oct & nov 2019

**3:51**

**21. AWS Serverless: Lambda**

303. Lambda Concurrency

concurrency and asynchronous invocations

* if the function doesn't have enough concurrency available to process all events, additional requests are throttled
* for throttling errors (429) and system errors (500 - series), lambda returns the event to the queue and attempts to run the function again for up to 6 hours
* the retry interval increases exponentially for 1 second after the first attempt to a maximum of 5 minutes

**1:17**

**21. AWS Serverless: Lambda**

303. Lambda Concurrency

lambda concurrency and throttling

* concurrency limit: up to 1000 concurrent executions
* can set a reserved concurrency at the function level
* each invocation over the concurrency limit will trigger a throttle
* if you need a higher limit, open a support ticket
* throttle behavior:
  + if synchronous invocation => return throttleerror - 429
  + if asynchronous invocation => retry automatically and then go to DLQ

**5:29**

**21. AWS Serverless: Lambda**

298. Lambda Function Performance

lambda functions /tmp space

* if your lambda function needs to download a big file to work...
* if your lambda function needs disk space to perform operations..
* you can use the /tmp directory
* max size is 10gb
* the directory content remains when the execution context is frozen, providing transient cache that can be used for multiple invocations (helpful to checkpoint your work)
* for permanent persistence of object (non temporary), use s3
* to encrypt content on /tmp, you must generate KMS data keys

**2:38**

**21. AWS Serverless: Lambda**

298. Lambda Function Performance

lambda execution context:

* the execution context is a temporary runtime environment that initializes any external dependencies of your lambda code
* great for database connections, HTTP clients, SDK clients....
* the execution context is maintained for some time in anticipation of another lambda function invocation
* the next function invocation can re-use the context to execution time and save time in initializing connections objects

**1:43**

**21. AWS Serverless: Lambda**

298. Lambda Function Performance

lambda function configuration

* RAM
  + from 128MB to 10 GB in 1MB increments
  + the more RAM you add, the more vcpu credits you get
  + at 1,792MB, a function has the equivalent of one full vcpu
  + after 1792 MB, you get more than one CPU, and need to use multi-threading in your code to benefit from it
* if your application is cpu-bound (computation heavy), increase RAM
* timeout: default 3 seconds, maximum is 900 seconds (15 minutes)

**4:19**

**21. AWS Serverless: Lambda**

296. Lambda in VPC

lambda in vpc - internet access

* a lambda function is your vpc does not have internet access
* deploying a lambda function in a public subnet does not give it internet access or a public IP
* deploying a lambda function in a private subnet gives it internet access if you have a NAT Gateway/intance
* you can use VPC endpoints to privately access aws services without a NAT

**1:18**

**21. AWS Serverless: Lambda**

296. Lambda in VPC

lambda in VPC

* you must define the vpc id, the subnets and the security groups
* lambda will create an eni (elastic network interface) in your subnets
* lambda function need awslambdavpcaccessexecutionrole to create eni

**0:53**

**21. AWS Serverless: Lambda**

296. Lambda in VPC

lambda by default

* by default, your lambda function is launched outside your own VPC (in an aws-owned VPC)
* therefore it cannot access resources in your VPC (RDS, elasticache, internal elb...)

**5:35**

**21. AWS Serverless: Lambda**

295. Lambda@Edge & CloudFront Functions

cloudfront functions vs lambda@edge - use cases

lambda@edge:

* longer execution time (several ms)
* adjustable CPU or memory
* your code depends on a 3rd libraries (e.g., AWS SDK to access other aws services)
* network access to use external services for processing
* file system access or access to body of HTTP requests

cloudfront functions:

* cache key normalization
  + transform request attributes (headers, cookies, query strings, URL) to create an optimal cache key
* header manipulation
  + insert/modify /delete HTTP headers in the request or response
* URL rewrites or redirects
* request authentication & authorization
  + create and validate user-generated tokens (e.g., JWT) to allow/deny requests

**3:39**

**21. AWS Serverless: Lambda**

295. Lambda@Edge & CloudFront Functions

lambda@edge

* lambda functions written in nodejs or python
* sclaes to 1000s of requests/second
* author your functions in one aws region ,then cloudfront replicates to its locations
* used to change cloudfront requests and responses:
  + viewer request - after cloudfront receives a request from a viewer
  + origin request - before cloudfront forwards the request to the origin
  + origin response- after cloudfront receives the response from the origin
  + viewer response - before cloudfront forwards the response to the viewer

**2:57**

**21. AWS Serverless: Lambda**

295. Lambda@Edge & CloudFront Functions

cloudfront functions:

* lightweight functions written in javascript
* for high-scale, latency-sensitive CDN customizations
* sub-millisecond startup times, millions of requests/second
* used to change viewer requests and responses:
  + viewer request: after cloudfront receives a request from a viewer
  + viewer response: before cloudfront forwards the response to the viewer
* native feature of cloudfront (manage code entirely within cloudfront)

**1:41**

**21. AWS Serverless: Lambda**

295. Lambda@Edge & CloudFront Functions

cloudfront functions & lambda@edge

use cases:

* website security and privacy
* dynamic web application at the edge
* search engine optimization (SEO)
* intelligently route across origins and data centers
* bot mitigation at the edge
* real-time image transformation
* A/B testing
* user authentication and authorization
* user prioritization
* user tracking and analytics

**1:07**

**21. AWS Serverless: Lambda**

295. Lambda@Edge & CloudFront Functions

customizatio at the edge:

* many modern applications execute some form of the logic at the edge
* edge function:
  + a code that you write and attach to cloudfront distributions
  + runs close to your users  to minimize latency
* cloudfront provides two types: cloudfront functions & lambda@edge
* you don't have to manage any servers, deployed globally
* use case: customize the CDN content
* pay only for what you use
* fully serverless

**1:38**

**21. AWS Serverless: Lambda**

293. Lambda Monitoring & X-Ray Tracing

lambda tracing with x-ray

* enable in lambda configuration (active tracing)
* runs the x-ray daemon for you
* use aws x-ray sdk in code
* ensure lambda function has a correct IAM execution role
  + the managed policy is called AWSXRayDaemonWriteAccess
* environment variables to communicate with x-ray
  + \_X\_AMZN\_TRACE\_ID: contains the tracing header
  + AWS\_XRAY\_CONTEXT\_MISSING: by default, LOG\_ERROR
  + AWS\_XRAY\_DAEMON\_ADDRESS: the x-ray daemon IP\_ADDRESS:PORT

**0:47**

**21. AWS Serverless: Lambda**

293. Lambda Monitoring & X-Ray Tracing

lambda logging & monitoring

* cloudwatch logs:
  + aws lambda execution logs are stored in aws cloudwatch logs
  + make sure your aws lambda function has an execution role with an IAM policy that authorized writes to cloudwatch logs
* cloudwatch metrics:
  + aws lambda metrics are displayed in aws cloudwatch metrics
  + invocations, durations, concurrent executions
  + error count, success rates, throttles
  + async delivery failures
  + iterator age (kinesis & dynamoDB streams)

**0:47**

**21. AWS Serverless: Lambda**

291. Lambda Environment Variables

lambda environment variables

* environment variables = key/value pair in "string" form
* adjust the function behavior without updating code
* the environment variables are available to your code
* lambda service adds its own system environment variables as well
* helpful to store secrets (encrypted by KMS)
* secrets can be encrypted by the lambda service key , or your own CMK

**1:47**

**21. AWS Serverless: Lambda**

286. Lambda Event & Context Objects

lambda - event and context objects

* event object
  + JSON-formatted document contains data for the function to process
  + contains information from the invoking service (e.g., eventbridge, custom...)
  + lambda runtime converts the event to an object (e.g., dict type in python)
  + example: input arguments, invoking service arguments,...
* context object
  + provides methods and properties that provide information about the invocation, function, and runtime environment
  + passed to your function by lambda at runtime
  + example: aws\_request\_id, function\_name,memory\_limit\_in\_mb,.....

**7:03**

**21. AWS Serverless: Lambda**

284. Lambda Event Source Mapping

lambda event mapper scaling

* kinesis data streams & dynamoDB streams:
  + one lambda invocation per stream shard
  + if you use parallelization, up to 10 batches processed per shard simultaneously
* sqs standard:
  + lambda adds 60 more instances per minute to scale up
  + up to 1000 batches of messages processed simultaneously
* sqs fifo:
  + messages with the same groupid will be processed in order the lambda function scales to the number of active message groups

**6:08**

**21. AWS Serverless: Lambda**

284. Lambda Event Source Mapping

queues & lambda

* lambda also supports in-order processing for FIFO queues, scaling up to the number of active message groups
* for standard queues, items aren't necessarily processed in order
* lambda scales up to process a standard queue as quickly as possible
* when an error occurs, batches are returned to the queue as individual items and might be processed in a different grouping than the original batch
* occasionally, the event source mapping might receive the same item from the queue twice, even if no function error occurred
* lambda deletes items from the queue after they're processed successfully
* you can configure the source queue to send items to a dead-letter queue if they can't be processed

**4:57**

**21. AWS Serverless: Lambda**

284. Lambda Event Source Mapping

lambda - event source mapping sqs & sqs fifo

* event source mapping will poll sqs (long polling)
* specify batch size (1-10 messages)
* recommended: set the queue visibility timeout to 6x the timeout of your lambda function
* to use a DLQ
  + set-up on the sqs queue, not lambda (DLQ for lambda is only for async invocations)
  + or use a lambda destination for failures

**3:50**

**21. AWS Serverless: Lambda**

284. Lambda Event Source Mapping

streams & lambda - error handling

* by default, if your function returns an error, the entire batch is reprocessed until the function succeeds, or the items in the batch expire
* to ensure in-order processing, processing for the affected shard is paused until the error is resolved
* you can configure the event source mapping to :
  + discard old events
  + restrict the number of retries
  + split the batch on error (to work around lambda timeout issues)
* discarded events can go to a destination

**2:59**

**21. AWS Serverless: Lambda**

284. Lambda Event Source Mapping

streams & lambda (kinesis & dynamoDB)

* an event source mapping creates an iterator for each shard, processes items in order
* start with new items, from the beginning or from timestamp
* processed items aren't removed from the stream (other consumers can read them)
* low traffic: use batch window to accumulate records before processing
* you can process multiple batches in parallel
  + up to 10 batches per shard
  + in-order processing is still guaranteed for each partition key

**1:06**

**21. AWS Serverless: Lambda**

284. Lambda Event Source Mapping

lambda - event source mapping

* kinesis data streams
* sqs & sqs fifo queue
* dynamodb streams
* common denominator: records need to be polled from the source
* your lambda function is invoked synchronously

**1:15**

**21. AWS Serverless: Lambda**

282. Lambda & S3 Event Notifications

s3 events notifications

* s3: ObjectCreated, s3: ObjectRemoved, s3:ObjectRestore, s3:Replication ...
* object name filtering possible (\*.jpg)
* use case: generate thumbnails of images uploaded to s3
* s3 event notifications typically deliver events in seconds but can sometimes take a minute or longer
* if two writes are made to a single non-versioned object at the same time, it is possible that only a single event notification will be sent
* if you want to ensure that an event notification is sent for every successful write, you can enable versioning on your bucket
* You have enabled and configured Event Notifications in your S3 bucket to invoke a Lambda function every time an object is uploaded to your S3 bucket. You have noticed that there's duplicate logging into CloudWatch Logs with the same request ID. What do you think is the reason for this?
* the lambda function has failed and retries have happend

**3:09**

**21. AWS Serverless: Lambda**

278. Lambda Asynchronous Invocations & DLQ

lambda - asynchronous invocations - services

* amazon simple storage service (s3)
* amazon simple notification service (SNS)
* amazon cloudwatch events/eventbridge
* aws codecommit (codecommit trigger: new branch, new tag, new push)
* aws codepipeline (invoke a lambda function during the pipeline, lambda must callback)
* -----other------
* amazon cloudwatch logs (log processing)
* amazon simple email service
* aws cloudformation
* aws config
* aws IoT
* aws IoT events

**2:10**

**21. AWS Serverless: Lambda**

278. Lambda Asynchronous Invocations & DLQ

lambda - asynchronous invocations

* s3, sns, cloudwatch events..
* the events are placed in an event queue
* lambda attempts to retry on errors
* make sure the processing is idempotent (in case of retries)
* if the function is retried, you will see duplicate logs entries in cloudwatch logs
* can define a DLQ (dead-letter queue) - SNS or SQS - for failed processing (need correct IAM permissions)
* asynchronous invocations allow you to speed up the processing if you don't need to wait for the result (ex: you need 1000 files processed)

**0:57**

**21. AWS Serverless: Lambda**

276. Lambda & Application Load Balancer

lambda integration with ALB

* to expose a lambda function as an HTTP(s) endpoint
* you can use the application load balancer (or an API gateway)
* the lambda function must be registered in a target group

**3:04**

**21. AWS Serverless: Lambda**

272. AWS Lambda Overview

aws lambda language support

* node.js
* python
* java (java 8 compatible)
* golang
* c# (.net core)
* ruby
* custom runtime API (community supported, example Rust)
* lambda container image
  + the container image must implement the lambda runtime API
  + ECS /fargate is prefered for running arbitrary docker images

**2:15**

**21. AWS Serverless: Lambda**

272. AWS Lambda Overview

benefits of aws lambda

* easy pricing:
  + pay per request and compute time
  + free tier of 1,000,000 AWS lambda requests and 400,000 GBs of compute time
* intergrated with the whole aws suite of services
* integrated with many programming languages
* easy monitoring through aws cloudwatch
* easy to get more resources per functions (upto 10GB of RAM)
* increasing RAM will also improve CPU and network!

**1:10**

**21. AWS Serverless: Lambda**

272. AWS Lambda Overview

why aws lambda

amazon lambda:

* virtual functions - no servers to manage!
* limited by time - short executions
* run on-demand
* scaling is automated!

amazon ec2:

* virtual servers in the cloud
* limited by RAM and CPU
* continuously running
* scaling means intervention to add / remove servers

**1:18**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

268. CloudTrail vs CloudWatch vs X-Ray

cloudtrail vs cloudwatch vs x-ray

* cloudtrail:
  + audit api calls made by users /services / aws console
  + useful to detect unauthorized calls or root cause of changes
* cloudwatch:
  + cloudwatch metrics over time for monitoring
  + cloudwatch logs for storing application log
  + cloudwatch alarms to send notifications in case of unexpected metrics
* x-ray:
  + automated trace analysis & central service map visualization
  + latency, errors and fault analysis
  + request tracking across distributed systems

**5:18**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

265. CloudTrail

cloudtrail events retention

* events are stored for 90 days in cloudtrail
* to keep events beyond this period, log them to s3 and use athena

**4:45**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

265. CloudTrail

cloudtrail insights

* enable cloudtrail insights to detect unusual activity in your account:
  + inaccurate resource provisioning
  + hitting service limits
  + bursts of aws iam actions
  + gaps in periodic maintenance activity
* cloudtrail insights analyzes normal management events to create a baseline
* and the continously analyzes write events to detect unusual patterns
  + anomalies appear in the cloudtrail console
  + event is sent to amazon s3
  + an eventbridge event is generated (for automation needs)

**3:28**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

265. CloudTrail

cloudtrail events

* management events:
  + operations that are performed on resources in your aws account
  + examples:
    - configuring security (IAM AttachRolePolicy)
    - configuring rules for routing data (amazon EC2 CreateSubnet)
    - setting up logging (AWS cloudtrail CreateTrail)
  + by default, trails are configured to log management events
  + can seperate read events (that don't modify resources ) from write events (that may modify resources)
* data events:
  + by default, data events are not logged (because high volume operations)
  + amazon s3 object-level activity (ex: GetObject, DeleteObject, PutObject): can separate read and write events
  + aws lambda function execution activity (the Invoke API)

**1:03**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

265. CloudTrail

aws cloudtrail:

* provides governance, compliance and audit for your aws account
* cloudtrail is enabled by default
* get an history of events api call made within your aws account by console, sdk, cli, aws services
* can put logs from cloudtrail into cloudwatch logs or s3
* a trail can be applied to all regions (default) or a single region
* if a resource is deleted in aws, investigate cloudtrail first!
* AWS CloudTrail allows you to log, continuously monitor, and retain account activity related to actions across your AWS infrastructure. It provides the event history of your AWS account activity, audit API calls made through the AWS Management Console, AWS SDKs, AWS CLI. So, the EC2 instance termination API call will appear here. You can use CloudTrail to detect unusual activity in your AWS accounts.

**1:50**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

264. AWS Distro for OpenTelemetry

aws distro for opentelemetry:

* secure, production-ready aws-supported distribution of the open-source project opentelemetry project
* provides a single set of APIs, libraries, agents, and collector services
* collects distributed traces and metrics from your apps
* collects metadata from your aws resources and services
* auto-instrumentation agents to collect traces without changing your code
* send traces and metrics to multiple aws service and partner solutions  x-ray, cloudwatch, prometheus...
* instrument your apps running on aws (e.g.. ec2, ecs, eks, fargate, lambda) as well as on-premises
* migrate from x-ray to aws distro for telemetry if you want to standardize with open-source APIs from telemetry or send traces to multiple destinations simultaneously

**2:47**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

261. X-Ray APIs

x-ray read APIs

* GetServiceGraph: main graph
* BatchGetTraces: Retrieves a list of traces specified by ID. Each trace is a collection of segment documents that originates from a single request.
* GetTraceSummaries: Retrieves IDs and annotations for traces available for a specified time frame using an optional filter. TO get the full traces, pass the trace IDs to BatchGetTraces.
* GetTraceGraph: Retrieves a service graph  for one or more specific trace IDs

**1:46**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

261. X-Ray APIs

s-ray write APIs (used by the x-ray daemon)

* PutTraceSegments: uploads segment documents to AWS x-ray
* putTelemetryRecords: used by the AWS x-ray daemon to upload telemetry.  setmentsReceivedCount, SegmentsRejectedCounts, BackendConnectionErrors...
* GetSamplingRules: retriece all sampling rules (to know what/when to send)
* GetSamplingTargets & GetSamplingStatisticSummaries : advanced
* the x-ray daemon needs to have an IAM policy authorizing the correct API calls to function correctly

**4:39**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

259. X-Ray: Instrumentation and Concepts

x-ray custom sampling rules

* you can create your own rules with the reservoir and rate
* example higher minimum rate for POSTs
  + rule name = post minimum
  + priority - 100
  + reservoir - 10
  + rate- 0.10
  + service name -\*
  + service type - \*
  + host - \*
  + HTTP method- post
  + URL path - \*
  + resource ARN - \*
* example debugging rule to trace all requests for problematic route
  + high - priority rule applied temporarily for debugging
  + rule name = debug history-updates
  + priority - 1
  + reservoir - 1
  + rate- 1
  + service name -  scorekeep
  + service type - \*
  + host - \*
  + HTTP method- put
  + URL path - /history/\*
  + resource ARN - \*

**3:25**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

259. X-Ray: Instrumentation and Concepts

x-ray sampling rules

* with sampling rules, you control the amount of data that you record
* you can modify sampling rules without changing your code
* by default, the x-ray sdk records the first request each second, and five percent of any additional requests
* one request per second is the reservior, which ensures that at least one trace is recorded each second as long the service is serving requests
* five percent is the rate at which additional requests beyond the reservoir size are sampled

**2:40**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

259. X-Ray: Instrumentation and Concepts

x-ray concepts

* segments: each application/service will send them
* subsegments: if you need more details in your segment
* trace: segments collected together to form an end-to-end trace
* sampling: decrease the amount of requests sent to x-ray , reduce cost
* annotations: key value pairs used to index traces and use with filters
* metadata: key value pairs, not indexed, not used for searching
* the x-ray daemon/agent has a config to send traces cross account:
  + make sure the IAM permissions are correct - the agent will assume the role
  + this allows to have a central account for all your application tracing

**1:15**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

259. X-Ray: Instrumentation and Concepts

x-ray instrumentation in your code

* instrumentation means the measure of product's performance, diagnose errors , and to write trace informantion
* to instrument your application code, you use the x-ray sdk
* many sdk require only configuration changes
* you can modify your application code to customize and annotation the data that the sdk sends to x-ray, using interceptors, filters, handlers, middleware

**7:00**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

257. X-Ray Overview

aws x-ray troubleshooting

* if x-ray is not working on ec2
  + ensure the ec2 IAM role has the proper permissions
  + ensure the ec2 instance is running the x-ray daemon
* to enable on aws lambda:
  + ensure it has an IAM execution rolewith proper policy (AWSX-RayWriteOnlyAccess)
  + ensure that x-ray is imported in the code
  + enable lambda x-ray active tracing

**5:58**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

257. X-Ray Overview

aws x-ray

how to enable  it?

1. your code  (java, python, go, node.js ,.net must import th3 aws x-ray SDK)
   * very little code modification needed
   * the application SDK will then capture:
     + calls to aws services
     + HTTP/HTTPS requests
     + database calls (Mysql , postgresql, dynamoDB)
     + queue calls(SQS)
2. install the x-ray daemon or enable x-ray aws integration
   * x-ray deamon works as a low level UDP packet interceptor (linux /windows/mac ...)
   * aws lambda /other aws services already run the x-ray daemon for you
   * each application must have the IAM rights to write data to x-ray

**4:12**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

257. X-Ray Overview

aws x-ray leverages tracing

* tracing is an end to end way to following a "request"
* each component dealing with the request adds its own trace
* tracing is made of  segments (+ sub segments)
* annotations can be added to traces to provide extra-information
* ability to trace:
  + every request
  + sample request (as a % for example or a rate per minute)
* x-ray security
  + iam for authorization
  + kms for encryption at rest

**2:52**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

257. X-Ray Overview

aws x-ray advantages

* troubleshooting performance (bottlenecks)
* understand dependencies in a microservice architecture
* pinpoint service issues
* review request behavior
* find errors and exceptions
* are we meeting time SLA?
* where I am throttled?
* identify users that are impacted

**1:21**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

256. Amazon EventBridge - Multi-Account Aggregation

eventbridge - multi-account aggregation

**6:56**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

254. Amazon EventBridge

amazon eventbridge - resource-based policy

* manage permissions for a specific event bus
* example: allow/deny events from another AWS account or aws region
* use case: aggregate all events from you aws organization in a single aws account or aws region

**5:42**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

254. Amazon EventBridge

amazon eventbridge - schema registry

* eventbridge can analyze the events in your bus and infer the schema
* the schema registry allows you to generate code for your application, that will know in advance how data is structured in the event bus
* schema can be versioned

**4:50**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

254. Amazon EventBridge

amazon eventbridge

* default event bus
* partner event bus
* custom event bus
* aws services send their events to default event bus
* event buses can be accessed by other aws accounts using resource-based policies
* you can archine events (all/filter) sent to an event bus (indefinitely or set period)
* ability to replay archived events

**1:10**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

254. Amazon EventBridge

amazon eventbridge ( formerly cloudwatch events)

* schedule : cron jobs (scheduled scripts)
* event pattern: event rules to react to a service doing something
* trigger lambda functions, send SQS/SNS messages...

**2:54**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

253. CloudWatch Synthetics

cloudwatch synthetics canary blueprints

* heartbeat monitor - load URL, store screenshot and an HTTP archive file
* API canary - test basic read and write functions of rest api
* broken link checker - check all links inside the URL that you are testing
* visual monitoring - compare a screenshot taken during a canary run with a baseline screenshot
* canary recorder - used with cloudwatch synthetics recorder (record your actions on a website and automatically generates a script for that)
* GUI workflow builder - verifies that actions can be taken on your webpage (e.g. test a webpage with a login form)

**1:49**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

253. CloudWatch Synthetics

cloudwatch synthetics canary

* configurable script that monitor your APIs, URLs websites,...
* Reproduce what your customers do programmatically to find issues before customers are impacted
* checks the availability and latency of your endpoints and can store load time data and screenshots of the UI
* integration with cloudwatch alarms
* scripts written in node.js or python
* programmatic access to a headless google chrome browser
* can run once or on a regular schedule

**3:57**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

251. CloudWatch Alarms

cloudwatch alarm: goo to know

* alarms can be created based on cloudwatch logs metrics filters
* to test alarms and notifications, set the alarm state to alarm using CLI
* aws cloudwatch set-alarm-state --alarm-name "myalarm" --state-value ALARM --state-reason "testing purposes"

**2:38**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

251. CloudWatch Alarms

cloudwatch alarms - composite alarms

* cloudwatch alarms are on a single metric
* composite alarms are monitoring the states of multiple other alarms
* AND and OR conditions
* helpful to reduce alarm nose by creating complex composite alarms

**0:38**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

251. CloudWatch Alarms

cloudwatch alarms

* alarms are used to trigger notifications for any metric
* various options (sampling , %, mac min, etc...)
* You have an application hosted on a fleet of EC2 instances managed by an Auto Scaling Group that you configured its minimum capacity to 2. Also, you have created a CloudWatch Alarm that is configured to scale in your ASG when CPU Utilization is below 60%. Currently, your application runs on 2 EC2 instances and has low traffic and the CloudWatch Alarm is in the **ALARM** state. What will happen?
* the cloudwatch alarn will remain in alarm state. however the EC2 instance in asg won't decrease less then minimum capacity
* alarm states:
  + OK
  + INSUFFICIENT\_DATA
  + ALARM
* period:
  + length of time in seconds to evaluate the metric
  + high resolution custom metrics: 10 sec, 30 sec or multiples of 60 sec

**1:10**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

249. CloudWatch Logs - Metric Filters

cloudwatch logs metric filter

* cloudwatch logs can use filter expressions
  + for example, find a specific IP inside of a log
  + or count occurrences of ERROR in your logs
  + metric filters can be used to trigger alarms
* filters do not retroactively filter data, filters only publish the metric data points for events that happen after the filter was created
* ability to specify up to 3 dimensions for the metric filter (optional)

**3:12**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

248. CloudWatch Agent & CloudWatch Logs Agent

cloudwatch unified agent - metrics

* collected directly on your linux server / EC2 instance
* cpu (active , guest, idle, system, user, steal)
* disk metrics (free,used,total), disk IO (writes,reads,bytes,iops)
* RAM (free, inactive, used, total, cached)
* Netstat (number of TCP and UDP connections, net packets, bytes)
* processes (total, dead, bloqued, idle, running, sleep)
* swap space (free, used, used %)
* reminder: out-of-the box metrics for EC2 - disk, CPU, network (high level)
* use unified cloudwatch agent to push memory usage as a custom metric to cloudwatch

**1:52**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

248. CloudWatch Agent & CloudWatch Logs Agent

cloudwatch logs agent & unified agent

* for virtual servers (ec2 instances, on-premise servers...)
* cloudwatch logs agent
  + old version of the agent
  + can only send to cloudwatch logs
* cloudwatch unified agent
  + collect additional system-level metrics such as RAM, processes etc..
  + collect logs to send to cloudwatch logs
  + centralized configuration using ssm parameter store

**0:50**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

248. CloudWatch Agent & CloudWatch Logs Agent

cloudwatch logs for EC2

* by default, no logs from your EC2 machine will ge to cloudwatch
* you need to run a cloudwatch agent on EC2 to push the log files you want
* make sure IAM permissions are correct
* the cloudwatch log agent can be setup on-premises too

**5:59**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

245. CloudWatch Logs

cloudwatch logs subscriptions

* cross-account subscription - send log events to resources in a different AWS acount (KDS, KDF)

**4:15**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

245. CloudWatch Logs

cloudwatch logs subscriptions

* get a real-time log events from cloudwatch logs for processing and analysis
* send to kinesis data streams, kinesis data firehose, or lambda
* subscription filter - filter which logs are events delivered to your destination

**3:44**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

245. CloudWatch Logs

cloudwatch logs - s3 export

* log data can take upto 12 hours to become available for export
* the API call is createExportTask
* not near-real time or real-time ... use Logs subscriptions instead

**3:21**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

245. CloudWatch Logs

  cloudwatch logs insights

* search and analyze log data stored in cloudwatch logs
* example: find a spcefic IP inside a log , count occurences of ERROR in your logs...
* provides a purpose-built query language
  + automatically discovers fields from aws services and JSON log events
  + fetch desired event fields, filter based on conditions, calculate aggregate statistics, sort events, limit number of events..
  + can save queries and add them to cloudwatch dashboards
* can query multiple Log groups in different AWS acounts
* its a query engine, not a real-time engine

**1:43**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

245. CloudWatch Logs

cloudwatch logs - sources

* sdk, cloudwatch logs agent, cloudwatch unified agent
* elastic beanstalk: collection of logs from application
* ecs: collection from containers
* aws lambda: collection from function logs
* vpc flow logs: vpc specific logs
* api gateway
* cloudtrail based on filter
* route53: log DNS queries

**1:03**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

245. CloudWatch Logs

cloudwatch logs

* log groups: arbitrary name, usually representing an application
* log stream: instances within application/log files/ containers
* can define log expiration policies (never expire, 1 day to 10 years...)
* cloudwatch logs can send logs to :
  + amazon s3 (exports)
  + kinesis data streams
  + kinesis data firehose
  + aws lambda
  + opensearch
* logs are encrypted by default
* can setup KMS-based encryption with your won keys

**1:17**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

244. CloudWatch Custom Metrics

cloudwatch custom metrics

* possibility to define and send your own custom metrics to cloudwatch
* example: memory (RAM) usage, disk space, number of logged in users...
* use API  call PutMetricData
* ability to use dimensions (attributes) to segment metrics
  + Instance.id
  + Environment.name
* metric resolution (StorageResolution API parameter- two possible value):
  + standard: 1 minute (60 seconds)
  + high resolution :1/5/10/30 seconds - higher cost
* important: accepts metric data points two weeks in the past and two hours in the future (make sure to configure your EC2 instance time correctly)

**1:22**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

243. CloudWatch Metrics

EC2 detailed monitoring

* Ec2 instance metrics have metrics every 5 minutes
* with detailed monitoring (for a cost), you get data "every 1 minute"
* use detailed monitoring if you want to scale faster for your ASG!
* the AWS free tier allows us to have 10 detailed monitoring metrics
* Note: EC2 memory usage is by default not pushed (must be pushed from inside the instance as a custom metric)
* This is a paid offering and is disabled by default. When enabled, the EC2 instance's metrics are available in 1-minute periods.

**2:44**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

242. Monitoring Overview in AWS

monitoring in AWS

* AWS cloudwatch:
  + Metrics: collect and track key metrics
  + Logs: collect, monitor, analyze and store log files
  + events: send notifications when certain events happen in your aws
  + alarms: react in real-time to metrics / events
  + Amazon CloudWatch is a monitoring service that allows you to monitor your applications, respond to system-wide performance changes, optimize resource utilization, and get a unified view of operational health. It is used to monitor your applications' performance and metrics.
* aws x-ray:
  + troubleshooting application performance and errors
  + distributed tracing of microservices
* aws cloudtrail:
  + internal monitoring of API calls being made
  + audit changes to aws resources by your users

**1:31**

**20. AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail**

242. Monitoring Overview in AWS

why monitoring is important

* we know how to deploy applications
  + safely
  + automatically
  + using infrastructure as code
  + leveraging the best AWS components!
* our applications are deployed and our users don't care how we did it..
* out users only care that the application is working!
  + application latency: will it increase over time?
  + application outages: customer experience should not be degraded
  + troubleshooting and remediation
* internal monitoring:
  + can we prevent issues before they happen?
  + performance and cost
  + trends (scaling patterns)
  + learning and improvement

**2:54**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

240. SQS vs SNS vs Kinesis

sqs vs sns vs kinesis

* sqs
  + consumer "pull data"
  + data is deleted after being consumed
  + can have as many workers (consumers) as we want
  + no need to provision throughput
  + ordering guarantees only on FIFO queues
  + individual message delay capability
* sns
  + push data to many subscribers
  + up to 12,500,000 subscribers
  + data is not persisted (lost if not delivered)
  + pub/sub
  + upto 100,000 topics
  + no need to provision throughput
  + integrates with sqs for fan-out architecture pattern
  + FIFO capability for sqs FIFO
* kinesis
  + standard: pull data
    - 2 MB per shard
  + enhanced fan-out: push data
    - 2MB per shard per consumer
  + Meant for real-time big data, analytics and ETL
  + ordering at the shard level
  + data expires after x days
  + provisioned mode or on-demand capacity mode

**6:59**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

239. Data Ordering for Kinesis vs SQS FIFO

kinesis vs sqs ordering

* lets assume 100 trucks, 5 kinesis shards, 1 sqs fifo
* kinesis data streams:
  + on average you'll have 20 trucks per shard
  + trucks will have their data ordered within each shard
  + the maximum amount of consumers in parallel we can have is 5
  + can receive up to 5 MB/s of data
* sqs fifo
* you only have one sqs fifo queue
* you will have 100 groupid
* you can have up to 100 conosumers (due to the 100 groupid)
* you have up to 300 messages per second (or 3000 if using batching)

**5:19**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

239. Data Ordering for Kinesis vs SQS FIFO

ordering data into sqs

* for sqs standard, there is no ordering
* for sqs fifo , if you don't use a group ID , messages are consumed in the order they are sent, with only one consumer
* you want to scale the number of consumers, but you want messages to be grouped when they are related to each other
* then use a groupID (similar to partition key in kinesis)

**3:52**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

239. Data Ordering for Kinesis vs SQS FIFO

ordering data in kinesis

* imagine you have 100 trucks on the road sending their GPS positions regularly into aws
* you want to consume the data in order for each truck, so that you can track their movement accurately
* how should you send that data into kinesis?
* answer: send using a "partition key" value of the truck\_id
* the same key will always go to the same shard
* Kinesis Data Stream uses the partition key associated with each data record to determine which shard a given data record belongs to. When you use the identity of each user as the partition key, this ensures the data for each user is ordered hence sent to the same shard.

**3:30**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

238. Kinesis Data Analytics

kinesis data analytics for apache flink

* use flink (java,scala or SQL) to process and analyze streaming data
* Run any apache flink application on managed cluster on AWS
  + provisioning compute resources, parallel computation , automatic scaling
  + application backups (implemented as checkpoints and snapshots)
  + use any apache flink programming features
  + flink does not read from firehose (use kinesis analytics for SQL instead)

**1:50**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

238. Kinesis Data Analytics

kinesis data analytics (sql application)

* real-time analytics on kinesis data streams & firehose using SQL
* add reference data from amazon s3 to enrich streaming data
* fully managed, no servers to provision
* automatic scaling
* pay for actual consumption rate
* output:
  + kinesis data streams: create streams out of the real-time analytics queries
  + kinesis data firehose: send analytics query results to destinations
* use cases:
  + time-series analytics
  + real-time dashboards
  + real-time metrics

**4:51**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

236. Kinesis Data Firehose Overview

kinesis data streams vs firehose

* You are running an application that produces a large amount of real-time data that you want to load into S3 and Redshift. Also, these data need to be transformed before being delivered to their destination. What is the best architecture would you choose?
* This is a perfect combo of technology for loading data near real-time data into S3 and Redshift. Kinesis Data Firehose supports custom data transformations using AWS Lambda.
* kinesis data streams
  + streaming service for ingest at scale
  + write custom code (producer/consumer)
  + real-time (~200ms)
  + manage scaling (shard splitting/merging)
  + data storage for 1 to 365 days
  + supports replay capability
* kinesis data firehose
  + load streaming data into s3/redshift/opensearch/3rd party/custom http
  + fully managed
  + near real-time (buffer time min. 60sec)
  + automatic scaling
  + no data storage
  + doesn't support replay capability

**3:32**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

236. Kinesis Data Firehose Overview

kinesis data firehose

* fully managed service, no administration, automatic scaling, serverless
  + aws:redshift/amazon s3/ opensearch
  + 3rd party partner: splunk/mongodb/datadog/newrelic/...
  + custom: send to any http endpoint
* pay for data going through firehose
* near realtime
* supports many data formats, conversions, transformations, compression
* supports custom data transformations using AWS lambda
* can send failed or all data to a backup s3 bucket

**2:13**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

235. Kinesis Operations

kinesis operation - merging shards

* decrease the stream capacity and save costs
* can be used to group two shards with low traffic (cold shards)
* old shards are closed and will be deleted once the data is expired
* can't merge more than two shards in a single operation

**1:32**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

235. Kinesis Operations

kinesis operation - shard splitting

* used to increase the stream capacity (1MB/s data in per shard)
* used to divide a "hot shard"
* the old shard is closed and will be deleted once the data is expired
* No automatic scaling (manually increase/decrease capacity)
* can't split into more than two shards in a single operation

**1:23**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

234. Kinesis Client Library

kinesis client library (KCL)

* When using Kinesis Client Library, each shard is to be read-only by one KCL instance. So, if you have 10 shards, then the maximum KCL instances you can have is 10.
* a java library that helps read record from a kinesis data stream with distributed applications sharing the read workload
* each shard is to be read by only one kcl instance
  + 4 shards = max. 4 KCL instances
  + 6 shards = max. 6 KCL instances
* progress is checkpointed into dynamoDB (needs IAM access)
* track other workers and share the work amongst shards using dynamodb
* KCL can run on EC2, elastic beanstalk, and on-premises
* records are read in order at the shard level
* versions:
  + KCL 1.x (supports shared consumer)
  + KCL 2.X (supports shared & enhanced fan-out consumer)

**5:01**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

232. Kinesis Consumers

  kinesis consumers - aws lambda

* supports classic & enhanced fan-out consumers
* read records in batches
* can configure batch size and batch window
* if error occurs, lambda retrie until succeeds or data expired
* can process up to 10 batches per shard simultaneously
* lambda funtions process data and save it in dynamodb

**3:54**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

232. Kinesis Consumers

kinesis consumers types

* enhanced fan-out consumer - pull
  + multiple consuming applications for the same stream
  + 2MB/sec per consumer per shard
  + latency ~70ms
  + higher costs
  + kinesis pushes data to consumers over HTTP/2 (SubscribeToShard API)
  + soft limit of 5 consumer applications (KCL) per data stream (default)
* shared (classic) Fan-out consumer - pull
  + low number of consuming applications
  + read throughput: 2MB/sec per shard across all consumers
  + max. 5 GetRecords API calls/sec
  + latency ~200ms
  + minimize cost($)
  + consumers poll data from kinesis using GetRecords API call
  + returns upto 10MB (then throttle for 5 seconds) or upto 10000 records

**2:43**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

232. Kinesis Consumers

enhanced fan-out consumer

* 2MB/sec per consumer per shard
* push model
* consumer has to subscribe to shard
* each consumer can get data at the rate of  2MB/sec

kinesis consumers - custom consumer

* shared (classic) fan-out consumer
* pull model
* the more consumer we add the more limitation we have
* 2 MB/sec per shard across all consumers

**4:28**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

231. Kinesis Producers

kinesis - provisionedThroughputExceeded - if we over produce into a shard then we are going to get provisionedThroughputExceeded exception

* You have a Kinesis data stream with 6 shards provisioned. This data stream usually receiving 5 MB/s of data and sending out 8 MB/s. Occasionally, your traffic spikes up to 2x and you get a ProvisionedThroughputExceededException exception. What should you do to resolve the issue?
* The capacity limits of a Kinesis data stream are defined by the number of shards within the data stream. The limits can be exceeded by either data throughput or the number of reading data calls. Each shard allows for 1 MB/s incoming data and 2 MB/s outgoing data. You should increase the number of shards within your data stream to provide enough capacity.
* solution
  + use highly distributed partition key
  + retries with exponential backoff
  + increase shards (scaling)

**1:16**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

231. Kinesis Producers

kinesis producers

* You have a Kinesis Data Stream where you intermittently get a ProvisionedThroughputExceededException exception in your producers' applications. The following can be used to resolve the error,
  + use highly distributed partition key
  + retry with exponential backoff
  + add more shards
* puts data records into data streams
* data record consist of:
  + sequence number (unique per partition-key within shard)
  + partition key (must specify while put records into stream)
  + data blob (up to 1 mb)
* producers:
  + aws sdk: simple producer
  + kinesis producer library (KPL): c++,java,batch,compression,retries
  + kinesis agent : monitor log files
* write throughput: 1mb/sec or 1000 records/sec per shard
* PutRecord API
* use batching with PutRecords API to reduce costs & increase throughput

**5:45**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

230. Kinesis Data Streams Overview

kinesis data streams security

* control access/authorization using IAM policies
* encryption in flight using https endpoints
* encryption at rest using kms
* you can implement encryption / decryption of data on client side (harder)
* vpc endpoints available for kinesis to access within vpc
* monitor api calls using cloudtrail

**5:01**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

230. Kinesis Data Streams Overview

kinesis data streams - capacity modes

* provisioned mode:
  + you choose the number of shards provisioned, scale manually or using API
  + each shard gets 1MB/s in (or 1000 records per second)
  + each shard gets 2mb/s out (classic or enhanced fan-out consumer)
  + you pay per shard provisioned per hour
* on-demand mode:
  + nod need to provision or manage the capacity
  + default capacity provisioned (4mb/s in or 4000 records per second)
  + scales automatically based on observed throughput peak during the last 30 days
  + pay per stream per hour & data in/out per GB

**3:45**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

230. Kinesis Data Streams Overview

kinesis data streams

* retention between 1 day to 365 days
* ability to reprocess (replay) data
* once data is inserted in kinesis, it can't be deleted (immutability)
* data that shares the same partition goes to the same shard (ordering)
* producers: aws sdk , kinesis producer library (kpl), kinesis agent
* consumers:
  + write your own : kinesis client library (kcl), aws sdk
  + managed : aws lambda, kinesis data firehose, kinesis data analytics

**1:11**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

229. Kinesis Overview

kinesis overview

* make it easy to collect, process, and analyze streaming data in real-time
* ingest real-time data such as : application logs, metrics, website clickstreams, IoT telemetry data..
* kinesis data streams: capture, process and store data streams
* kinesis data firehose: load data process into aws data stores
* kinesis data analytics: analyze data streams with sql or apache flink
* kinesis video streams: capture, process and store video streams

**5:56**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

227. Amazon SNS and SQS - Fan Out Pattern

sns - message filtering

* JSON policy used to filter messages sent to SNS topic's subscriptions
* if a subscription doesn't have a filter policy, it receives every message

**3:52**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

227. Amazon SNS and SQS - Fan Out Pattern

amazon sns - FIFO topic

* FIFO = first in first out (ordering of messages in the topic)
* similar features as sqs fifo:
  + ordering by message group id (all messages in the same group are ordered)
  + deduplication using a deduplication id or content based deduplication
* can have sqs standatd and fifo queues as subscribers
* limited throughput (same throughput as sqs fifo)

**3:08**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

227. Amazon SNS and SQS - Fan Out Pattern

application: sns to amazon s3 through kinesis data firehouse

**2:25**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

227. Amazon SNS and SQS - Fan Out Pattern

application: s3 events to multiple queues

* for the same combination of: event type (e.g. object create) and prefix (e.g. images/) you can only have one s3 event rule
* if you want to send the same s3 event to many sqs queues, use fan-out

**1:26**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

227. Amazon SNS and SQS - Fan Out Pattern

sns +sqs : fan out

* push once in sns, receive in all sqs queues that are subscribers
* This is a common pattern where only one message is sent to the SNS topic and then "fan-out" to multiple SQS queues. This approach has the following features: it's fully decoupled, no data loss, and you have the ability to add more SQS queues (more applications) over time.
* fully decoupled, no data loss
* sqs allows for : data persistence, delayed processing and retries of work
* ability to add more sqs subscribers over time
* make sure your sqs queue access policy allows for sns for write
* cross-region delivery: works with sqs queues in other regions

**4:13**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

226. Amazon SNS

amazon sns -security

* encryption
  + in-flight encryption using HTTPS API
  + at-rest encryption using KMS keys
  + client-side encryption if the client wants to perform encryption/decryption itself
* access controls: IAM policies to regulate access to the SNS API
* SNS Access policies (similar to s3 bucket policies)
  + useful for cross-account access to sns topics
  + useful for allowing other services (s3..) to write to an sns topic

**3:28**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

226. Amazon SNS

aws sns - how to publish

* topic publish (using the sdk)
  + create a topic
  + create a subscription (or many)
  + publish to the topic
* direct publish (for mobile apps sdk)
  + create a platform application
  + create a platform endpoint
  + publish to the platform endpoint
  + works with google GCM, Apple APNS, Amazon ADM...

**2:48**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

226. Amazon SNS

SNS integrates with a lot of AWS services

* many AWS services can send data directly to SNS for notifications
* following aws services can publish data to SNS
  + cloudwatch alarms
  + aws budgets
  + lambda
  + auto scaling group (notifications)
  + s3 bucket (events)
  + dynamoDB
  + cloudformation (state changes)
  + AWS DMS (new replic)
  + RDS events

**2:21**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

226. Amazon SNS

amazon SNS

* the event producer only sends message to one sns topic
* as many event receivers (subscriptions) as we want to listen to the sns topic notifications
* each subscriber to the topic will get all the messages (note: new feature to filter messages)
* up to 12,500,000 subscriptions per topic
* 100,000 topics limit
* SNS can publish data to SQS, lambda, kinesis data firehose, emails, sms & mobile notifications, http(s) endpoints
* Note: Kinesis Data Firehose is now supported for subscription, but not Kinesis Data Streams.

**2:55**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

225. SQS - FIFO Queues Advanced

sqs fifo - message grouping

* You can have as many consumers as "MessageGroupID" for your SQS FIFO queues.
* if you specify the same value of MessageGroupID in an SQS FIFO queue, you can only have one consumer, and all the messages are in order
* to get ordering at the level of a subset of messages, specify different values for MessageGroupID
  + messages that share a common message group id will be in order within the group
  + each group id can have a different consumer (parallel processing)
  + ordering across groups is not guaranteed

**1:18**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

225. SQS - FIFO Queues Advanced

sqs FIFO - deduplication

* de-duplication interval is 5 minutes
* two de-duplication methods:
  + content-based deduplication; will do a SHA-256 hash of the message body
  + explicitly provide a message deduplication ID

**1:28**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

224. SQS - FIFO Queues

amazon sqs - fifo queue

* FIFO = first in first out (ordering of messages in the queue)
* limited throughput: 300msg/sec without batching, 3000 msg/s with
* exactly-once send capability (by removing duplicates)
* messages are processed in order by the consumer
* SQS FIFO (First-In-First-Out) Queues have all the capabilities of the SQS Standard Queue, plus the following two features. First, The order in which messages are sent and received are strictly preserved and a message is delivered once and remains available until a consumer process and deletes it. Second, duplicated messages are not introduced into the queue.

**4:50**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

223. SQS - Certified Developer concepts

sqs - must know API

* CreateQueue (MessageRetentionPeriod), DeleteQueue
* PurgeQueue: delete all the messages in queue
* SendMessage (DelaySeconds), ReceiveMessage, DeleteMessage
* MaxNumberOfMessages: default 1, max 10 (for ReceiveMessage API)
* ReceiveMessageWaitTimeSeconds: long polling
* ChangeMessageVisibility: change the message timeou
* batch APIs for SendMessage, DeleteMessage, CHangeMessageVisibility helps decrease your costs

**3:24**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

223. SQS - Certified Developer concepts

sqs extended client

* message size limit is 256Kb, how to send large messages , e.g. 1 GB?
* using the sqs extended client (java library)

**1:50**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

223. SQS - Certified Developer concepts

Amazon sqs - long polling

* when a consumer requests messages from the queue, it can optionally "wait" for messages to arrive if there are none in the queue
* this is called long polling
* longpolling decreases the number of API calls made to sqs while increasing the efficiency and decreasing the latency of your application
* the wait time can be between 1 sec to 20 sec (20 sec preferable)
* long polling is prefereable to short polling
* long polling can be enabled at the queue level or at the API level using "ReceiveMessageWaitTimeSeconds"

**0:52**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

222. SQS - Delay Queues

amazon sqs - delay queue

* delay a message (consumer don't see it immediately) upto 15 minutes
* default is 1 seconds (message is available right away)
* can set a default at queue level
* can override the default on send using the DelaySeconds parameter
* SQS Visibility Timeout is a period of time during which Amazon SQS prevents other consumers from receiving and processing the message again. In Visibility Timeout, a message is hidden only after it is consumed from the queue. Increasing the Visibility Timeout gives more time to the consumer to process the message and prevent duplicate reading of the message. (default: 30 sec., min.: 0 sec., max.: 12 hours)

**2:42**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

220. SQS - Dead Letter Queues

sqs dlq - redrive to source

* feature to help consume messages in the dlq to understand what is wrong with them. it helps to inspect manually and debug
* when our code is fixed, we can redrive the messages from the DLQ back into the source queue (or any other queue) in batches without writing custom code

**1:48**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

220. SQS - Dead Letter Queues

amazon sqs - dead letter queue (DLQ)

* if a consumer fails to process a message within the visibility timeout... the message goes back to the queue!
* SQS Dead Letter Queue is where other SQS queues (source queues) can send messages that can't be processed (consumed) successfully. It's useful for debugging as it allows you to isolate problematic messages so you can debug why their processing doesn't succeed.
* we can set a threshold of how many times a message can go back to the queue
* after the MaximumReceives threshold is exceeded, the message goes into a dead letter queue (DLQ)
* useful for debugging
* DLQ of a FIFO queue must also be a FIFO queue
* DLQ of a standard queue must also be a standard queue
* make sure to process the messages in the DLQ before the expire:
  + good to set a retention of 14 days in the DLQ

**1:20**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

219. SQS - Message Visibility Timeout

sqs - message visibility timeout

* after a message is polled by a consumer, it becomes invisible to other consumers
* by default, the "message visibility timeout" is 30 seconds
* that means the message has 30 seconds to be processed
* after the message visibility timeout is over, the message is visible in sqs
* if a message is not processed within the visibility timeout, it will be processed twice
* a consumer could call the ChangeMessageVisibility API to get more time
* If visibility timeout is high (hours), and consumer crashes, re-processing will take time
* if visibility timeout is too low (seconds), we may get duplicates

**2:06**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

218. SQS Queue Access Policy

sqs queue access policy:

sqs queue will need to give permission to the s3 bucket to write a message to it. therefore we need to create our own sqs queue access policy and it looks as

{

   "version":"2012-10-17",

   "statement":[{

         "Effect":"Allow",

         "principal":{"AWS":"\*"},

         "Action":["sqs:SendMessage"],

         "Resource":"arn:aws:sqs:us-east-1:4445555666:queue1",

         "Condition":{

               "ArnLike":{"aws:SourceArn":"arn:aws:s3:\*:\*:bucket1"},

               "stringEquals":{"aws:SourceAccount":"<bucket1\_owner\_account\_id>"},

         }

    }]

}

**10:30**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

216. Amazon SQS - Standard Queues Overview

amazon SQS -security

* encryption:
  + in-flight encryption using HTTPS API
  + At-rest encryption using KMS keys
  + client-side encryption if the client wants to perform encryption/decryption itself
* access controls: IAM policies to regulate access to the sqs API
* sqs access policies (similar to s3 bucket policies)
  + useful for cross-account access to sqs queues
  + useful for allowing other services (SNS,S3..) to write to a SQS queue

**7:18**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

216. Amazon SQS - Standard Queues Overview

sqs with auto scaling group (asg)

* the ASG has to be scaling on some kind of metric. and the metric that is available to use is called the queue length and its called "ApproximateNumberOfMessages". it is a cloudwatch metric that is available in any sqs.
* we could setup an alarm such as when ever the queue length go over a certain level, then please setup a cloudwatch alarm and this alarm should increase the capacity of ASG by x amount.

**6:50**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

216. Amazon SQS - Standard Queues Overview

sqs - multiple ec2 instances consumers

* consumers receive and process messages in parallel
* at least once delivery
* best-effort message ordering
* consumers delete messages after processing them
* we can scale consumers horzontally to improve throughput of processing

**5:55**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

216. Amazon SQS - Standard Queues Overview

sqs - consuming messages

* consumers (running on ec2 instances, or AWS lambda)
* poll sqs for messages (receives up to 10 messages at a time)
* process the messages
* delete the messages using the DeleteMessage API

**4:19**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

216. Amazon SQS - Standard Queues Overview

sqs - producing messages

* produced to sqs using the SDK (sendmessage API)
* the message is persisted in sqs until a consumer deletes it
* message retention: default 4 days, upto 14 days
* example: send an order to be processed
* sqs standard: unlimited throughput

**3:10**

**19. AWS Integration & Messaging: SQS, SNS & Kinesis**

216. Amazon SQS - Standard Queues Overview

amazon SQS - standard queue

* over 10 years old
* fully managed service , used to decouple applications
* attributes:
  + unlimited throughput, unlimited number of messages in queue
  + default retention of messages: 4 days, maximum of 14 days
  + low latency (<10ms on publish and receive)
  + limitation of 256KB per message sent
* can have duplicate messages (at least once delivery, occasionally)
* can have out of order messages (best effort ordering)
* SQS Delay Queues is a period of time during which Amazon SQS keeps new SQS messages invisible to consumers. In SQS Delay Queues, a message is hidden when it is first added to the queue. (default: 0 mins, max.: 15 mins)

**6:15**

**18. AWS CloudFormation**

196. CloudFormation - Overview

cloudformation building blocks:

templates helpers:

1. references
2. functions

template components

1. resources: your AWS resources declared in the template
2. parameters: the dynamic inputs for your template
3. Mappings: the static cariables for your template
4. outputs:references to what has been created
5. conditionals: list of conditions to perform resource creation
6. metadata

**5:01**

**18. AWS CloudFormation**

196. CloudFormation - Overview

deploying cloudfromation templates

* manual way:
* editing templates in the cloudformation designer
* using the console to input parameters, etc
* automated way:
  + editing templates in a yaml file
  + using the aws cli to deploy the templates
  + recommended way when you fully want to automate your flow

**4:34**

**18. AWS CloudFormation**

196. CloudFormation - Overview

How CloudFormation works

* templates have to be uploaded in s3 and then referenced in CloudFormation
* to update a template, we can't edit previous onces. we have to reupload a new version of the template to AWS
* stacks are identified by a name
* deleting a stack deletes every single artifact that was created by cloudformation

**3:16**

**17. AWS Elastic Beanstalk**

194. Beanstalk Migrations

Elastic beanstalk migration: decouple RDS

1. create a snapshot of RDS DB (as a safeguard)
2. go to the RDS console and protect the RDS database from deletion
3. create a new elastic beanstalk environment, without RDS, point your application to existing RDS
4. perform a CNAME swap (blue/green) or route 53 update, confirm working
5. terminate the old environment (RDS won't be deleted)
6. terminate the old environment (RDS won't be deleted)
7. delete cloudfromation stack (in DELETE\_FAILED state)

**1:56**

**17. AWS Elastic Beanstalk**

194. Beanstalk Migrations

RDS with elastic beanstalk

* RDS can be provisioned with beanstalk, which is great for dev/test
* this is not great for prod as the database lifecycle is tied to the beanstalk environment lifecycle
* the best for prod is to separately create an RDS database and provideour EB application with the connection string

**1:18**

**17. AWS Elastic Beanstalk**

194. Beanstalk Migrations

Elastic beanstalk migration : load balancer

* after creating an elastic beanstalk environment, you cannot change the elastic load balancer type (only the configuration)
* to migrate:
  1. create a new environment with same configuration except LB (can't clone)
  2. deploy your application onto the new environment
  3. perform a CNAME swap or route 53 update

**0:45**

**17. AWS Elastic Beanstalk**

193. Beanstalk Cloning

Elastic beanstalk cloning

* clone an environment with the exact same configuration
* useful for deploying a test version of your application
* all resources and configuration are preserved:
  + load balancer type and configuration
  + RDS database type (but the data is not preserved)
  + environment variables
* after cloning an environment, you can change settings

**1:23**

**17. AWS Elastic Beanstalk**

191. Beanstalk Extensions

elastic beanstalk extensions

* a zip file containing our code must be deployed to elastic beanstalk
* all the parameters set in the UI can be configured with code using files
* resources managed by .ebextensions get deleted if the environment goes away
* Requirements:
  + in the .ebextensions/ directory in the root of source code
  + YAML / JSON format
  + .config extensions (Example: logging.config)
  + able to modify some default settings using: option\_settings
  + ability to add resources such as RDS, Elasticcache, dynamoDB, etc...

**0:52**

**17. AWS Elastic Beanstalk**

190. Beanstalk Lifecycle Policy Overview + Hands On

Beanstalk Lifecycle policy

* Elastic beanstalk can store at most 1000 application versions
* if you don't remove old versions, you won't be able to deploy anymore
* to phase out old application versions, use a lifecycle policy
  + based on time (old versions are removed)
  + based on space (when you have too many versions)
* versions that are currently used won't be deleted
* option not to delete the source bundle in s3 to prevent data loss

**2:05**

**17. AWS Elastic Beanstalk**

189. Beanstalk CLI and Deployment Process

Elastic Beanstalk deployment process

* describe dependencies (requirements.txt for python, package.json for Node.js)
* Package code as Zip, and describe dependencies
  + python: requirements.txt
  + Node.js: package.json
* console: upload zip file (creates new app version), and then deploy
* CLI: create new app version using CLI (uploads zip), and then deploy
* Elastic Beanstalk will deploy the zip on each EC2 instance, resolve dependencies and start the application

**0:50**

**17. AWS Elastic Beanstalk**

189. Beanstalk CLI and Deployment Process

elastic beanstalk CLI

* we can install an additional CLI called the "EB cli" which makes working with beanstalk from the CLI easier
* Basic commands are:
  + eb create
  + eb status
  + eb health
  + eb events
  + eb logs
  + eb open
  + eb deploy
  + eb config
  + eb terminate
* it helpful for your automated deployment pipelines!

**10:44**

**17. AWS Elastic Beanstalk**

187. Beanstalk Deployment Modes

Elastic beanstalk deployment - Traffic splitting

* canary testing
* New application version is deployed to a temporary ASG with the same capacity
* A small % of traffic is sent to the temporary ASG for a configurable amount of time
* Deployment health is monitored
* If there's a deployment failure, this triggers an automated rollback (very quick)
* No application downtime
* New instances are migrated from the temporary to the original ASG
* Old application version is then terminated
* You're running an application on Elastic Beanstalk. You have just finished a major update to your application. You want to deploy the new version then direct a small percentage of traffic to the new version so you can test and fall back if there're any issues.

**9:08**

**17. AWS Elastic Beanstalk**

187. Beanstalk Deployment Modes

Elastic beanstalk deployment

Blue/green:

* not a "direct feature" of elastic beanstalk
* zero downtime and release facility
* create a new stage environment and deploy v2 there
* the new environment (green) can be validated independently and roll back if issues
* Route 53 can be setup using weighted policies to redirect a little bit of traffic to the stage environment
* using beanstalk, swap URLs when done with the environment test

**7:04**

**17. AWS Elastic Beanstalk**

187. Beanstalk Deployment Modes

Elastic beanstalk deployment

Immutable:

* zero downtime
* new code is deployed to new instances on temporary ASG
* high cost, double capacity
* longest deployment
* quick rollback in case of failures (just terminate new ASG)
* great for production
* In this mode, a full set of new instances running the new version of the application in a separate Auto Scaling Group is launched. To roll back quickly, this mode terminates the ASG holding the new application version, while the current one is untouched and already running at full capacity.

**5:31**

**17. AWS Elastic Beanstalk**

187. Beanstalk Deployment Modes

Elastic beanstalk deployment:

Rolling with additional batches:

* Application is running at capacity
* can set the bucket size
* Application is running both versions simultaneously
* small additional cost
* additional batch is removed at the end of the deployment
* Longer deployment
* Good for production

**3:30**

**17. AWS Elastic Beanstalk**

187. Beanstalk Deployment Modes

Elastic beanstalk deployment :

Rolling:

* application is running below capacity
* can set the bucket size
* application is running both versions simultaneously
* No additional cost
* Long deployment

**1:07**

**17. AWS Elastic Beanstalk**

187. Beanstalk Deployment Modes

Beanstalk deployment options for updates

* all at once (deploy all in one go) - fastest, but instances aren't available to serve traffic for a bit (downtime)
* Rolling: update a few instances at a time (bucket), and then move onto the next bucket once the first bucket is healthy
* Rolling with additional batches: Rolling with additional batches: like rolling, but spins up new instances to move the batch (so that the old application is still available)
* immutable: spins up new instances in a new ASG, deploys version to these instances, and then swaps all the instances when everything is healthy
* blue green: create a new environment and switch over when ready
* Traffic splitting: canary testing - send a small % of traffic to new deployment

**3:01**

**17. AWS Elastic Beanstalk**

184. Elastic Beanstalk Overview (High level)

Elastic Beanstalk - components

* application: collection of Elastic beanstalk components (environments, versions, configurations, ...)
* Application version: an iteration of your application code
* Environment
  + collection of AWS resources running an application version (only one application version at a time)
  + Tiers: web server environment tier & worker environment tier
  + you can create multiple environments (dev, test, prod, ....)

**2:01**

**17. AWS Elastic Beanstalk**

184. Elastic Beanstalk Overview (High level)

Elastic beanstalk - overview

* Elastic beanstalk is a developer centric view of deploying an application on AWS
* it uses all the components we've seen before: EC2,ASG,ELB,RDS,...
* managed service
  + automatically handles capacity provisioning, load balancing, scaling, application health monitoring, instance configuration,...
  + just the application code is the responsibility of the developer
* we still have full control over the configuration
* beanstalk is free but you pay for the underlying instances

**1:21**

**16. ECS, ECR & Fargate - Docker in AWS**

180. AWS CoPilot - Overview

AWS Copilot

* CLI tool to build, release and operate production-ready containerized apps
* Run your apps on AppRunner, ECS and Farget
* Helps you focus on building apps rather than setting up infrastructure
* Provisions all required infrastructure for containerized apps (ECS,VPC,ELB,ECR..)
* Automated deployments with one command using codepipeline
* Deploy to multiple environments
* Troubleshooting, logs, health status...

**5:55**

**16. ECS, ECR & Fargate - Docker in AWS**

176. Amazon ECS - Task Placements

ECS task placement constraints

* distinctInstance: place each task on a different container instance

"placementConstraints":{ {"type":"distinctInstance"} }

* memberOf: places task on instances that satisfy an expression
  + uses the cluster query language (advanced)

"placementConstraints":{{"expression":"attribute:ecs.instance-type=~t2.\*", "type":"memberOf"}}

**4:20**

**16. ECS, ECR & Fargate - Docker in AWS**

176. Amazon ECS - Task Placements

ECS task placement strategies

"placementStrategy":{

{

"field" : "attribute:ecs.availability-zone" ,

"type":"spread"

}

}

* spread
  + place the task evenly based on the specified value
  + Example: instanceId,
  + attribute:ecs.availability-zone

**3:14**

**16. ECS, ECR & Fargate - Docker in AWS**

176. Amazon ECS - Task Placements

ecs task placement strategies

* random
  + place the task randomly

**3:07**

**16. ECS, ECR & Fargate - Docker in AWS**

176. Amazon ECS - Task Placements

ECS task placement strategies

"placementStrategy":{

{

"field": "memory",

"type":"binpack"

}

}

* binpack
  + place task based on the least available amount of CPU of memory
  + this minimizes the number of instances in use (cost savings)

**1:08**

**16. ECS, ECR & Fargate - Docker in AWS**

176. Amazon ECS - Task Placements

ECS tasks placement:

* when a task of type EC2 is launched, ECS must determine where to place it, wit the constraints of CPU, memory, and available port
* similarly, when a service scales in, ECS needs to determine which task to terminate
* To assist with this you can define a task placement strategy and task placement constraints
* Note: this only for ECS with EC2 , not for fargate

**8:56**

**16. ECS, ECR & Fargate - Docker in AWS**

174. Amazon ECS Task Definitions - Deep Dive

amazon ECS Data volumes (Bind Mounts)

* share data between multiple containers in the same task definition
* works for both EC2 and Fargate tasks
* EC2 tasks - using EC2 instance storage
  + Data are tied to the lifecycles of the EC2 instance
* Fargate tasks- using ephemeral storage
  + Data are tied to the containers(s) using them
  + 20GB - 200 GB (default 20 GB)
* usecases:
  + share ephemeral data between multiple containers
  + sidecar container pattern, where the sidecar container used to send metrics/logs to other destinations (seperation of concerns)

**5:42**

**16. ECS, ECR & Fargate - Docker in AWS**

174. Amazon ECS Task Definitions - Deep Dive

IAM roles are assigned per task definition

* the role is defined  at the task definition level not at this service level. and so, therefore all the tasks within your service, are going to get access to amazon s3.
* and if you define another task definition, you can add another role on it. and this role, for example, can access dynamoDB.

**4:38**

**16. ECS, ECR & Fargate - Docker in AWS**

174. Amazon ECS Task Definitions - Deep Dive

Amazon ECS - load balancing (Fargate)

* each task has a unique private IP
* only define the container port (host port is not applicable)
* Example
  + ECS ENI security group
    - allow port 80 from the ALB
  + ALB security group
    - Allow port 80/443 from web

**3:39**

**16. ECS, ECR & Fargate - Docker in AWS**

174. Amazon ECS Task Definitions - Deep Dive

Amazon ECS - load balancing (EC2 launch type)

* we get a dynamic host port mapping if you define only the container port in the task definition
* the ALB finds the right port on your EC2 instances
* you must allow on the EC2 instance's security group any port from the ALB's security group
* Dynamic port mapping is useful when you want to run multiple instances of the same task or service on a single EC2 instance without having to manually manage port conflicts.

**2:02**

**16. ECS, ECR & Fargate - Docker in AWS**

174. Amazon ECS Task Definitions - Deep Dive

Amazon ECS - task definitions

* task definitions are metadata in JSON form to tell ECS how to run a docker container
* it contains information, such as:
  + image name
  + port binding for container and host
  + memory and CPU required
  + environment variables
  + networking information
  + IAM role
  + Logging configuration (example cloudwatch)
* can define up to 10 containers in a task definition
* To enable random host port, set host port = 0 (or empty), which allows multiple containers of the same type to launch on the same EC2 container instance.

**2:25**

**16. ECS, ECR & Fargate - Docker in AWS**

171. Amazon ECS - Auto Scaling

EC2 launch Type - Auto scaling EC2 instances

* Accommodates ECS service scaling by adding underlying Ec2 instances
* Auto scaling group scaling
  + scale your ASG based on CPU utilization
  + Add EC2 instances over time
* ECS cluster capacity provider
  + used to automatically provision and scale the infrastructure for your ECS tasks
  + capacity provider paired with an auto scaling group
  + add EC2 instances when you're missing capacity (CPU,RAM...)

**1:21**

**16. ECS, ECR & Fargate - Docker in AWS**

171. Amazon ECS - Auto Scaling

ECS service Auto scaling

* Automatically increase/decrease the desired number of ECS tasks
* amazon ECS Auto scaling uses AWS application auto scaling
  + ECS service average CPU utilization
  + ECS service average memory utilization - scale on RAM
  + ALB request count per target - metric coming from the ALB
* target tracking - scale based on target value for a specific caloudwatch metric
* step scaling - scale based on a specified cloudwatch alarm
* scheduled scaling - scale based on a specified data/time (predictable changes)
* ECS service auto scaling (task level)  is not equals to EC2 auto scaling (EC2 instance level)
* Fargate auto scaling is much easier to setup (because serverless)

**6:41**

**16. ECS, ECR & Fargate - Docker in AWS**

167. Amazon ECS

amazon ECS - data volumes (EFS)

* mount EFS file systems onto ECS tasks
* works for both EC2 and fargate launch types
* tasks running in any AZ will share the same data in the EFS file system
* Fargate + EFS = serverless
* use cases:persistent multi-AZ shared storage for your containers
* Note
  + Amazon s3 cannot be mounted as a file system

**5:17**

**16. ECS, ECR & Fargate - Docker in AWS**

167. Amazon ECS

Amazon ECS - Load Balancer INtegrations

* Application load balancer supported and works for most use cases
* network load balancer recommended only for high throughput/high performance use cases, or to pair it with AWS private link
* Classic load balancer supported but not recommended  (no advanced features - no fargate)

**4:13**

**16. ECS, ECR & Fargate - Docker in AWS**

167. Amazon ECS

amaon ECS - IAM Roles for ECS

* EC2 instance profile (EC2 launch type only):
  + used by the ECS agent
  + makes API calls to ECS service
  + send container logs to cloudwatch logs
  + pull dockerimage from ECR
  + reference sensitive data in secrets manager or ssm parameter store
* ECS task role:
  + allows each task to have a specific role
  + use different roles for the different ECS services you run
  + Task Role is defined in the task definition

**2:34**

**16. ECS, ECR & Fargate - Docker in AWS**

167. Amazon ECS

amazon ECS - Fargate launch type

* launch docker containers on AWS
* you do not provision the infrastructure (on EC2 instances to manage)
* its all serverless!
* you just create task definitions
* AWS just runs ECS tasks for you based on theCPU/RAM you need
* To scale, just increase the number of tasks. Simple - no more EC2 instances
* Security Groups do not matter when an EC2 instance registers with the ECS service. By default, Security Groups allow all outbound traffic.

**1:27**

**16. ECS, ECR & Fargate - Docker in AWS**

167. Amazon ECS

amazon ECS - Ec2 launch type

* ECS = elastic container service
* launch docker containers on AWS = launch ECS tasks on ECS clusters
* EC2 launch type : you must provision & maintain the infrastructure (the Ec2 instances)
* Each Ec2 instance must run the Ecs Agent to register in the ECS cluster
* Aws takes care of starting/stopping containers
* docker containers are placed on amazon EC2 instances that we provision in advance
* Security Groups do not matter when an EC2 instance registers with the ECS service. By default, Security Groups allow all outbound traffic.

**5:00**

**16. ECS, ECR & Fargate - Docker in AWS**

166. Docker Introduction

docker containers management on AWS

* amazon elastic container service (amazon ECS)
  + amazons own container platform
* amazon elastic kubernetes service (amazon EKS)
  + amazon's managed kubernetes (open source)
* aws fargate
  + amazon's own serverless container platform
  + works with ecs and with eks
* amazon ECR
  + store container images

**1:27**

**15. CloudFront**

165. CloudFront - Real Time Logs

cloudfront - real time logs

* get real-time requests received by cloudfront sent to kinesis data streams
* monitor, analyze , and take actions based on content delivery performance
* Allows you to choose:
  + shampling rate - percentage of requests for which you want to receive
  + specific fields and specific cache behaviors (path patterns )

**7:05**

**15. CloudFront**

164. CloudFront Advanced Concepts

cloudfront - field level encryption

* protect user sensitive information through application stack
* Adds an additional layer of security along with HTTPS
* sensitive information encrypted at the edge close to user
* uses asymmetric encryption
* usage:
  + specify set of fields in POST requests that you want to be encrypted (up to 10 fields)
  + specify the public key to encrypt them

**1:44**

**15. CloudFront**

164. CloudFront Advanced Concepts

cloudfront - price classes

* you can reduce the number of edge locations for cost reduction
* three price classes:
  + price class All: all regions -best performance
  + price class 200: most regions, but excludes the most expensive regions
  + price class 100: only the least expennsive regions

**2:59**

**15. CloudFront**

164. CloudFront Advanced Concepts

cloudfront  - multiple origin

* to route to different kind of origins based on the content type
* based on path pattern:
  + /images/\*
  + /api/\*
  + /\*

**4:53**

**15. CloudFront**

164. CloudFront Advanced Concepts

cloudfront - origin groups

* to increase high-availability and do failover
* origin group: one primary and one secondary origin
* if the primary origin fails, the second one is used

**0:00**

**15. CloudFront**

163. CloudFront Signed URL - Key Groups + Hands On

cloudfront signed URL process

* how we can generate keys to sign the URLs with cloudfront.
  + two types of signers:
    - either a trusted key groups (recommended)
    - can leverage APIs to create and rotate keys (and IAM for API security)
    - an AWS account that contains a cloudfront key pair
      * need to manage keys using the root account and the aws console
      * not recommended because you shouldn't use the root account for this
* in your cloudFront distribution create one or more trusted key groups
* you can generate your own public/private key
  + the private key is used by your applications (e.g.EC2) to sign URLs
  + the public key (uploaded ) is used by cloudfront to verify URLs

**3:14**

**15. CloudFront**

162. CloudFront Signed URL / Cookies

cloudfront signed URL vs s3 pre-signed URL:

* cloud front signed URL
  + allow access to a path, no matter the origin
  + account wide key-pair, only the root can manage it
  + can filter by IP, path,date,expiration
  + can leverage caching features
* s3 pre-signed URL
  + Issue a request as the person who pre-signed the URL, then the person who has the URL has the same rights as the person who issued the pre-signed URL.
  + uses the IAM key of the signing IAM principal
  + Limited lifetime
  + the idea is the client can access the objects in the s3 bucket using the pre-signed URL
  + without a cloudfront then pre-signed URL of s3 is a great usecase.

**1:15**

**15. CloudFront**

162. CloudFront Signed URL / Cookies

cloudfront signed URL / signed cookies

* you want to distribute paid shared content to premium users over the world
* we can use cloudfront signed URL/cookie. we attach a policy with:
  + include URL expiration
  + includes IP ranges to access the data from
  + Trusted signers (which AWS accounts can create signed URLs)
* how long should the URL be valid for?
  + shared content (movie,music): make it short (a few minutes)
  + private content (private to the user):you can make it last for years
* signed URL - access to individual files (one signed URL per file)
* singed cookies = access to multiple files (one signed cookie for many files)

**2:12**

**15. CloudFront**

158. CloudFront - Cache Behaviors

cloudfront - cache behaviors - sign in page

cloudront - maximize cache hits by separating static and dynamic distributions

**1:17**

**15. CloudFront**

158. CloudFront - Cache Behaviors

cloudfront - cache behaviors

* configure different setting for a given URL path pattern
* Example: on specific cache behavior to images/\*.jpg files on your origin web server
* Route to different kind of origins/origin groups based on the content type or path pattern
  + /images/\*
  + /api/\*
  + /\* (default cache behavior)
* when adding additional cahce behaviors, the default cache behavior is always the last to be processed and is always /\*

**6:31**

**15. CloudFront**

156. CloudFront - Caching & Caching Policies

cache policy vs origin request policy

**5:42**

**15. CloudFront**

156. CloudFront - Caching & Caching Policies

cloudfront policies - origin request policy

* specify values that you want to include in origin requests without including them in the cache key (no duplicated cached content)
* ability to add cloudfront HTTP headers and custom headers to an origin request that were not included in the viewer request
* create your own policy or use predefined managed policies
* you can include:
  + HTTP headers: none - whitelist - All viewer headers options
  + cookies: None - whitelist - All
  + query Strings: None - whitelist - All

**4:09**

**15. CloudFront**

156. CloudFront - Caching & Caching Policies

cloudfront caching - cache policy HTTP headers

* none
  + don't include any headers in the cache key (except default)
  + headers are not forwarded (except default)
  + best caching performance
* whitelist:
  + only specified headers included in the cache key
  + specified headers are also forwarded to origin

**3:17**

**15. CloudFront**

156. CloudFront - Caching & Caching Policies

cloudfront policies - cache policy

* cache based on :
  + HTTP headers: none - whitelist
  + cookies: none - whitelist - include All-Except - All
  + query strings: none - whitelist - include All-except - All
* control the TTL (0 seconds to 1 year), can be sent by the origin using the cache-control header, expires header..
* create your own policy or use predefined managed policies
* All HTTP headers, cookies, and query strings that you include in the cache key are automatically included in origin requests

**2:17**

**15. CloudFront**

156. CloudFront - Caching & Caching Policies

what is cloudfront cache key?

* a unique identifier for every object in the cache
* by default, consists of hostname + resource portion of the URL
* if you have an application that serves up content that varies based on user, device, language, location..
* you can add other elements (HTTP headers, cookies, query) to the cache key using cloudfront cache policies)

**1:01**

**15. CloudFront**

156. CloudFront - Caching & Caching Policies

cloudfront caching

* the cache lives at each cloudfront edge location
* cloudfront identifies each object in the cache using the cache key
* you want to maximize the cache hit ratio to minimize requests to the origin
* you can invalidate part of the cache using the createInvalidation API

**4:49**

**15. CloudFront**

154. CloudFront - Overview

cloudfront vs s3 cross region replication

* s3 cross region replication
  + must be setup for each region you want replication to happen
  + files are updated in near real-time
  + read only
  + great for dynamic content that needs to be available at low-latency in few regions
* cloudfront
  + global edge network
  + files are cached for a TTL (maybe a day)
  + great for static content that must be available everywhere

**2:15**

**15. CloudFront**

154. CloudFront - Overview

u can have cloudfront in front of any custom origin HTTP backend like

* ALB,
* ec2 instance,
* s3 website (must first enable the bucket as static s3 website )
* any HTTP backend you want

**1:52**

**15. CloudFront**

154. CloudFront - Overview

cloudfront - origins

* s3 bucket which is used to distribute files and cache them at the edgeusing cloudfront.
* to guarantee that only cloudfront can access your s3 bucket you can use something called the origin access control, OAC
* cloudfront can be used as an ingress (to upload files to s3)

**1:33**

**15. CloudFront**

154. CloudFront - Overview

AWS cloudFront

* content delivery network (CDN)
* Improves read performance, content is cached at the edge
* improves users experience
* DDoS protection (because worldwide), integration with shield, AWS web application firewall

**4:11**

**7. AWS Fundamentals: ELB + ASG**

68. Elastic Load Balancer - SSL Certificates

ssl - SNI - server name indication

* SNI solves the problem of loading multiple SSL ertificates onto one web server (to serve multiple websites)
* Its a newer protocol and requires the client to indicate the hostname of the target server in the initial SSL handshake
* the server will then find the correct certificate, or return the default one
* Note:
* only works for ALB or NLB (newer generation), cloudfront
* does not work for CLB (older gen)

**0:14**

**18. AWS CloudFormation**

205. CloudFormation - Intrinsic Functions

cloudformation Must know Intrinsic functions

1. Ref
2. Fn::GetAtt
3. FN::ImportValue
4. FN::Join
5. Fn::Sub
6. Condition Functions (Fn::If, Fn::Not, Fn::Equals, etc...)

**1:41**

**11. Amazon S3 Introduction**

113. S3 Overview

amazon s3 usecases:

1. backup and storage
2. disaster recovery
3. archive
4. hybrid cloud storage
5. aplication hosting
6. media hosting
7. datalakes & big data analytics
8. software delivery
9. static wevbsite

**3:19**

**10. VPC Fundamentals**

112. Three Tier Architecture

LAMP stack on EC2

* linux:os for EC2 instances
* apache: web server that run on linux (EC2)
* Mysql: database on RDS
* PHP: application logic (running on Ec2)
* can add Redis/Memcached (elasticache) to include a chacing tech
* to store local application data & software: EBS drive (root)

**2:06**

**10. VPC Fundamentals**

111. VPC Cheat Sheet & Closing Comments

vpc closing comments:

* VPC: virtual private cloud
* subnetsL tied to an AZ, network partition of the VPC
* internet gateway: at the VPC level, provide internet access
* NAT gateway / instances: give internet access to private subnets
* NACL: stateless, subnet rules for inbound and outbound
* security groups: stateful, operate at the EC2 instance level or ENI
* VPC peering: connect 2 VPC with non overlapping IP ranges, non transitive
* VPC endpoints: provide private access to AWS services within VPC
* VPC flow logs: network traffic logs
* site to site VPN: VPN over public internet between on-premises DC and AWS
* direct connect: direct private connection to a AWS

**3:52**

**10. VPC Fundamentals**

110. VPC Peering, Endpoints, VPN, DX

vpc endpoint gateway

vpc endpoint interface

**3:48**

**10. VPC Fundamentals**

110. VPC Peering, Endpoints, VPN, DX

VPC Endpoints:

* 1.endpoints allow you to connect to AWS services using private network instead of the public www network
* this gives you enhanced security and lower latency to acces AWS services
* VPC endpoint gateway: s3 & dynamoDB
* VPC endpoint interface : the rest
* only used with you VPC

**1:45**

**10. VPC Fundamentals**

110. VPC Peering, Endpoints, VPN, DX

vpc peering:

1. connect two VPC, privately using AWS network

2.make them behave as if they were in the same network

3.must not have overlapping CIDR (IP address range)

4.VPC peering connection is not transitive (must be established for each VPC that need to communicate with one another)

**4:29**

**10. VPC Fundamentals**

109. NACL, SG, VPC Flow Logs

VPC flow logs:

1.capture information about IP traffic going into your interfaces:

1.VPC flow logs

2.subnet flow logs

3.elastic network interface flow logs

2.helps to monitor & troubleshoot connectivity issues. examples:

1.subnets to internet

2.subnets to subnets

3.internet to subnets

3.captures network information form AWS managed interfaces too: elastic load balancer, elasticcache,RDS,Aurora, etc..

4.VPC flow logs data can go to s3, cloudwatch logs, and kinesis data firehose.

**2:09**

**10. VPC Fundamentals**

109. NACL, SG, VPC Flow Logs

network ACL & security groups

1. NACL (network ACL)

1.a firewall which controls traffic from and to subnet

2.can have ALLOW and DENY rules

3.Are attached at the subnet level.

4.Rules only include IP addresses

2.security groups:

1.A firewall that controls traffic to and from an ENI/an EC2 instance

2.an have only ALLOW rules

3.Rules include IP addresses and other security groups

**5:01**

**10. VPC Fundamentals**

108. VPC, Subnets, IGW and NAT

NAT gateways (AWS-managed) & NAT instances (self-managed) allow your instances in your private subnets to access the internet while remaining private.

**3:19**

**10. VPC Fundamentals**

108. VPC, Subnets, IGW and NAT

internet gateway & NAT gateways:

1.internet gateways helps our VPC instances connect with the internet

2.public subnets have a route to the internet gateway.

**1:56**

**9. Route 53**

94. Routing Policy - Simple

routing policies - simple

1. typically , route traffic to a single resource

2.can specify multiple values in the same record

3.if multiple values are returned, a random one is chosen by the client

4.when alias enabled, spcify only one AWS resource

5.Can't be associated with health checks

**0:53**

**9. Route 53**

94. Routing Policy - Simple

route53 - routing policies

1. define how route 53 responds to DNS queries

2.don't get confused by the word "Routing"

1. tis not the same as load balancer routing which routes the traffic

2.DNS does not route any traffic, it only reponds to the DNS queries

3.Route 53 supports the following Routing policies

1.simple

2.weighted

3.failover

4.latency based

5.geolocation

6.multi-value answer

7.GEOproximity (using Route 53 Traffic flow feature)

**3:06**

**9. Route 53**

93. Route 53 CNAME vs Alias

route 53 -  alias records targets

1.elastic load balancers

2.cloudfront distributions

3.API gateway

4. elastic beanstalk environments

5.s3 websites

6.vpc interface endpoints

7.Global accelerator accelerator

8. route53 record in the same hosted zone

note: you cannot set an alias record for an ec2 DNS name

**1:14**

**8. AWS Fundamentals: RDS + Aurora + ElastiCache**

86. Amazon MemoryDB for Redis - Overview

amazon memoryDB for redis:

1.redis-compatible, durable , in-memory database service

2.ultra-fast performance with over 160 millions request/second

3.durable in-memory data storage with multi-AZ transactional log

4.scale seamlessly from 10s GBs to 100s TBs of storage

5.use cases: web and mobile apps, online gaming, media streaming,..

**11:37**

**8. AWS Fundamentals: RDS + Aurora + ElastiCache**

85. ElastiCache Strategies

final words of wisdom

1.lazy loading/cache aside is easy to implement and worksfor many situations as a foundation, especially on the read side

2.write-through is usually combined with lazy loading as targeted for the queries or workloads that benefits from this optimization

3.setting a TTL is usually not a bad idea, except when you're using write-through. set it to be a sensible value for your application

4.only cache the data that make sense (user profiles, blogs, etc..)

Quote: there are only two hard things in computer science: cache invalidation and naming things.

**1:15**

**8. AWS Fundamentals: RDS + Aurora + ElastiCache**

83. ElastiCache Overview

amazon Elasticache overview:

1. the same way RDS is to get managed relational databases

2.elasticache is to get managed Redis or Memcached

3.caches are in-memory databases with really high performance, low latency

4.helps reduce load off of databases for read intensive workloads

5.helps make your application stateless

6.AWS takes care of OS maintenance/ patching, optimizations, setup, configuration, monitoring, failure recovery and backups

7. using elasticache involves heavy application code changes

**2:48**

**8. AWS Fundamentals: RDS + Aurora + ElastiCache**

82. RDS Proxy

amazon RDS Proxy:

1.fully managed database proxy for RDS

2.allows apps to pool and share DB connections established with the database.

3.improving database efficiency by reducing the stress on database resources (eg, CPU,RAM) and minimize open connections (and timeouts)

4.serverless, autoscaling , highly available (multi-AZ)

5.Reduced RDS & Aurora failover time by up 66%

6. supports RDS (Mysql, PostgresSQL, MariaDB, MS SQL server) and Aurora (mysql ,postgressql)

7.no code changes required for most apps

8. enforece IAM authentication for DB, and securely store credentials in AWS secrets manager

9. RDS proxy is never publicly accessible (must be accessed from VPC)

**2:29**

**8. AWS Fundamentals: RDS + Aurora + ElastiCache**

81. RDS & Aurora Security

At-rest encryption:

1.database master & replicas encryption using aws kms - must be defined as launch time

2. if the master is not encrypted, the read replicas cannot be encrypted

3.to encrypt an un-encrypted database, go through a DB snapshot & restore as encrypted

in-flight encryption: TLS-ready by default, us the AWS TLS root certificate client-side

IAM AUthentication: IAM roles to connect to your database (instead of username/pw)

security groups: control network acess to your RDS/Aurora DB

No SSH available except on RDS custom

Audit Logs can be enabled and sent to CloudWatch Logs for longer retention.

**5:23**

**7. AWS Fundamentals: ELB + ASG**

60. Application Load Balancer (ALB)

application load balancer (v2)

good to know

fixed hostname

the application servers don't see the IP of the client directly

1.the true ip of the client is inserted in the header X-Forwarded-For

2. we can also get port (X-Forwarded-Port) and proto (x-Forwarded-proto)

**2:09**

**7. AWS Fundamentals: ELB + ASG**

60. Application Load Balancer (ALB)

application load balancer (v2)

routing tables to different target groups:

1. routing based on path in URL

2.routing based on hostname in URL

3/routing based on query string, headers

ALB are a great fit for micro services & container-based application (coker and amazon ecs)

has a port mapping freature to redirect to a dynamic port in ECS

**2:50**

**7. AWS Fundamentals: ELB + ASG**

58. Elastic Load Balancing (ELB) Overview

why use an elastic load balancer?

An elastic load balancer is a managed load balancer

it costs less to setup your own load balancer but it will be a lot more effort on your end

it is integrated with many AWS offering/services

1.EC2, EC2 auto scaling groups, Amazon Ecs

2. AWS certificate manager (ACM),CloudWatch

3. Route 53, AWS WAF, AWS global Accelerator

**0:00**

**6. EC2 Instance Storage**

50. EC2 Instance Store

EC2 Instance Store provides the best disk I/O performance.

ec2 instance store good for buffer / cache/ scratch data/ temporary content

backups and replication are your responsibility

You can run a database on an EC2 instance that uses an Instance Store, but you'll have a problem that the data will be lost if the EC2 instance is stopped (it can be restarted without problems). One solution is that you can set up a replication mechanism on another EC2 instance with an Instance Store to have a standby copy. Another solution is to set up backup mechanisms for your data. It's all up to you how you want to set up your architecture to validate your requirements. In this use case, it's around IOPS, so we have to choose an EC2 Instance Store.

**4:00**

**6. EC2 Instance Storage**

53. Amazon EFS

EFS is a network file system (NFS) that allows you to mount the same file system on EC2 instances that are in different AZs.

**1:10**

**6. EC2 Instance Storage**

52. EBS Multi-Attach

EBS multi-attach  -  io1/io2 family

1. attach the same EBS volume to multiple EC2 instances in the same AZ

2.Each instance has full read & write permissions to the high-performance volume

3.use case:

  1. achieve higher application availability in clustered linux applications (ex:teradata)

  2. Applications must manage concurrent write operatioins.

  3. upto 16 Ec2 instances at a time.

  4. must use a file system that is cluster aware (not xfs,ext4 etc)

**0:00**

**6. EC2 Instance Storage**

48. AMI Overview

AMIs are built for a specific AWS Region, they're unique for each AWS Region. You can't launch an EC2 instance using an AMI in another AWS Region, but you can copy the AMI to the target AWS Region and then use it to create your EC2 instances.

**0:00**

**6. EC2 Instance Storage**

51. EBS Volume Types

EBS Volumes are created for a specific AZ. It is possible to migrate them between different AZs using EBS Snapshots.

By default, the Root volume type will be deleted as its Delete On Termination attribute is checked by default. Any other EBS volume types will not be deleted as its Delete On Termination attribute is disabled by default.

When creating EC2 instances, you can only use the following EBS volume types as boot volumes: gp2, gp3, io1, io2, and Magnetic (Standard).

256,000 is the maximum IOPS you can achieve when you're using the EBS io2 Block Express volume type.

**3:40**

**6. EC2 Instance Storage**

44. EBS Overview

ebs and ec2 are linked to availability zones

**4:30**

**5. EC2 Fundamentals**

34. Security Groups & Classic Ports Overview

security groups Good to know:

1.can be attached to multiple instances

2.locked down to a region/VPC combination

3.does live outside the EC2 - if traffic is blocked the EC2 instances won't see it

4.its good to maintain one seperate security group for SSH access

5.if your applcation is not accessible (timeout), then it's a security group issue

6.if your application gives a "connection refused" error, then its an application error or it's not launched

7.All inbound traffic is blocked by default

8.all outbound traffic is authorised by default

**0:42**

**5. EC2 Fundamentals**

34. Security Groups & Classic Ports Overview

introduction to security groups:

security groups are the fundamental of network security in AWS

they control how traffic is allowed into or out of our EC2 instances.

securit groups only contain allow rules

security groups rules can reference by IP or by security group

Security Groups can be attached to multiple EC2 instances within the same AWS Region/VPC.

**0:26**

**5. EC2 Fundamentals**

32. Create an EC2 Instance with EC2 User Data to have a Website Hands On

aws uses public key cryptography to secure the login information for your instance. A linux instance has no password. you use your key pair to login to your instance securely. you specify the name of your key pair when u launch the instance . then provide the private key when u login in using ssh.

Note: to launch instances in multiple AWS Regions, you'll need to create a key pair in each region.

Dedicated Hosts are good for companies with strong compliance needs or for software that have complicated licensing models. This is the most expensive EC2 Purchasing Option available.

**7:08**

**5. EC2 Fundamentals**

32. Create an EC2 Instance with EC2 User Data to have a Website Hands On

#!/bin/bash

#use this for your user data (script)

#install httpd (Linux 2 version)

yum update -y

yum install -y httpd

systemctl start httpd

systemctl enable httpd

echo "<h1> Hello world from $(hostname -f) </h1>" > /var/www/html/index.html

**2:09**

**5. EC2 Fundamentals**

31. EC2 Basics

ec2 sizing and configuration options:

1.operating system (os): linux , windows or Mac OS

2.how much compute power & cores (CPU)

3.how much random-access memory (RAM)

4.how much storage space:

   1.network-attached (EBS & EFS)

   2.hardware (EC2 instance store)

5.network card: speed of the card, public IP address

6.Firewall rules: security group

7.Bootstrap script (configure at first launch): EC2 User Data

**0:46**

**5. EC2 Fundamentals**

31. EC2 Basics

amzon EC2 features:

renting virtual machines (EC2)

storing data on virtual drives (EBS)

distributing load across machines (ELB)

scaling the services using an auto-scaling group (ASG)

**1:02**

**4. IAM & AWS CLI**

29. IAM Summary

iam section - summary:

users: mapped to a physical user, has a password for aws console

groups: contains users only

policies: JSON document that outlines permissions for users or groups

Roles: for EC2 instances or AWS services

security: MFA+password policy

AWS CLI: manage your AWS services using the command-Line

AWS SDK: manage your AWS services using a programming language

Access keys: access AWS using CLI or SDK

Audit: IAM credentials Reports & IAM Access Advisor

**0:30**

**4. IAM & AWS CLI**

23. IAM Roles for AWS Services

iam roles for services:

iam role will be jus like a user, but they are intended to be used not by a physical user but by an aws service.

**0:31**

**11. Amazon S3 Introduction**

123. S3 Replication - Hands On

replication only works if versioning is enabled

**1:50**

**11. Amazon S3 Introduction**

115. S3 Security: Bucket Policy

s3 bucket policies:

JSON based policies

  1.resources: buckets and obets

**1:19**

**11. Amazon S3 Introduction**

115. S3 Security: Bucket Policy

an IAM principal can access an s3 object if

1. the user IAM permissions allow it or the resource policy allows it

2. and there's no explicit deny

**4:02**

**11. Amazon S3 Introduction**

115. S3 Security: Bucket Policy

advanced: cross-account access - use bucket policy

**3:45**

**11. Amazon S3 Introduction**

115. S3 Security: Bucket Policy

ec2 instance access - use IAM Roles

to give  s3 access to ec2 instance u must create Ec2 instance role with IAM permissions

**3:11**

**11. Amazon S3 Introduction**

115. S3 Security: Bucket Policy

public access to s3 - use bocket policy

use access to s3 - IAM permissions to the user through the IAM policy. there the policy allows access to the s3 bucket then the  user can access our s3 bucket.

**3:11**

**4. IAM & AWS CLI**

15. IAM MFA Overview

google authenticator is phone only app

Authy is multi-device they both work the same except one is multi-device. it can be used on computer and phone.

1.Authy has support for multiple tokens on a single device, that means with a virtual MFA device you can have your root account , you IAM user and another account, and another IAM user,

its up to you, you can have as many users and accounts as you want on your Virtual MFA device.