Notes on AWS Developer Associate Exam DVA-C02

Reference: https://www.udemy.com/course/aws-certified-developer-associate-dva-c01/

Reference Page to lecture pdf version 29

Last update: 2024-07-23

Infrastructure

- 1 Region 3-6 AZ, most service are region-scoped
- 1 AZ may consist of many data centers
- "edge location" = "CDN" = "faster delivery."
- choose base on compliance(eg Europe), proximity/distance, price

P.21 IAM

- 1 user - many groups
- assign policies=JSON to user/groups, define permission

```
Effect: Allow/Deny
Principal: AWS ac/user/role
Action:
Resource:
```

- assign permission with IAM roles, eg EC2 Instance Roles, Lambda function Roles, CloudFormation
- Credentials Report (per account)
- Access Advisor (when permission is last used)
- Shared Responsibility Model
  - AWS: Infra, Vulnerability, Compliance
  - User: Policy management, MFA, review

Trust Policy

- define which principal/entity(who) can assume an IAM role `sts:AssumeRole`

Permission policy

- What the role can do

IAM Policy Simulator

- edit policy inside does not affect actual environment

P.40 EC2

- EC2, EBS, ELB, ASG
- Storage
  - network-attached: EBS, EFS(many)
  - hardware: EC2 instance store
- Firewall = security group
- Bootstrap script = EC2 user data, run once at start on root layer
- Security group, *ALLOW* rules only, reference by IP/ SG, control inbound/outbound
- can attached to multiple instance, scoped to region/VPC
- can attach SG To SG
- Any IP = 0.0.0.0/0
- Access timeout = SG blocked
- connection refused = app error
- default no inbound, all outbound opened

P.62 EC2 pricing

- on-demand, billed per seconds(min charge 60s per instance)
- reserved(cheap)/ convertible reserved(a bit expensive)
  - reserved: charged per clock-hour. cannot use for 2 instance simultaneously
- saving plan
  - zonal reserved instance has capacity reservation, regional does not
- spot instance, short workload, cheap but may lose instance
- dedicated (license issue/ compliance)
  - dedicated instance = may share other instance in same AWS account, cheaper
  - dedicated host = whole physical server
- EC2 auto scaling support ALB or NLB

P.73 EBS

- network drive inside a AZ to mount on EC2
- EBS snapshot, backup/ copy snapshot to move EBS across AZ/region
- provisioned capacity, speed related to size
- on deleting EC2, root EBS is deleted and other attached is remained
  - use `DeleteOnTermination` in cli to disable if already launched
- can use EBS archive tier if long time no access

EBS Encryption

- encryption is region-based
- a snapshot of encrypted EBS is always encrypted
- a volume restored from encrypted snapshot is encrypted
- to make an volume encrypted, create a snapshot and enable encryption

P.79 AMI

- custom EC2 instance template, can be shared by AWS marketplace
- or you may use AWS provided AMI
- start EC2, stop the instance and build AMI
- EC2 instance store, higher performance(IOPS) but lost on stop/termination
- only SSD can be used as boot volume
- General Purpose SSD (GP2)
  - Provisioned IOPS SSD (IO1)
  - support multi-attach (16 EC2 instance)
  - Throughput Optimized HDD (ST1)

P.89 EFS

- network file system, mount on many(~100) EC2, for Linux only
- accessible in one region, multi AZ
- use security group to control access
- Throughput Mode
  - Bursting (default)
  - Provisioned (set min/max throughput)
  - Elastic, automatically scales
- different storage tier
  - Standard
  - Infrequent Access (EFS-IA)

  - One Zone (EFS)
  - Infrequent Access One Zone (EFS-IA)

- Availability

  - standard is multi-AZ, but can choose one zone

- EFS vs EBS vs Instance Store?

P.96 Availability, Scalability

- Vertical Scaling = increase size of instance (but with hardware limit)

- Horizontal Scaling = increase number of instance (distributed system)

  - Auto Scaling Group (ASG) = scale out/in based on metrics
  - Elastic Load Balancer (ELB) = distribute traffic across EC2

- High Availability = run in multi-AZ, survive AZ failure, but more expensive

  - ASG and ELB in different AZ

- ASG span in one region, multi AZ, cannot cross region

P.101 Load Balancer

- load balancing = server forward traffic to multiple server downstream

- expose single point of access, handle failure, health check, HTTPS

- stickiness with cookie, high availability across AZ

- ELB = managed load balancer, AWS guarantee, easy setup

- health check(port/route), if unhealthy, stop sending traffic and start new instance

- Classic Load Balancer(Deprecated)

  - need one per application

- ALB, HTTP(S), Websocket

  - support routing based on path, hostname, query string, headers

  - fit for micro service, containers

  - port mapping, redirect to dynamic port in ECS

    - in ECS, use task definition to map port

  - Target Group

    - EC2 instance = HTTP

        - EC2 task = HTTP
        - Lambda = HTTP with jSON event
        - private IP

- NLB, TCP(/TLS = secure TCP), UDP (Layer 4)
    - less latency than ALB, handle millions of request
    - one static IP per AZ, support elastic IP
    - Target Group
        - EC2 instance
        - Private IP
        - ALB
- Gateway Load Balancer, operates at layer 3(network)
    - Transparent Network Gateway + LB
    - used in firewalls
    - Target Group
        - EC2 instance
        - Private IP
- LB supports sticky sessions (user keep session data)
    - Cookies Name
        - AWSALB (duration based)
        - AWSELB (duration based, CLB)
        - AWSALBAPP (application cookie)
        - or custom cookie
- Cross-Zone Load Balancing
    - ALB default enabled, no charge for inter AZ
    - NLB GLB default disable, with charges
- SSL/ TLS
    - SSL certificate managed by AWS certificate manager/ custom import
    - SNL(server name indication), multiple SSL cert on one server

ECS de-registration

- terminate instance in
    - RUNNING state: deregister from ELB
    - STOPPED state: do not deregister automatically

ECS task placement

- binpack, select on least available CPU/memory (reduce instance needed)
- spread, to different instances

- random
  - can provide the other constraints that you specified
  - makes sure tasks are scheduled on instances with enough resources to run

example

```
"placementStrategy": [
    {"field": "instanceId", (or "attribute:ecs.availability-zone")
    "type": "spread"}
]
```

AWS Certificate Manager

- simplify SSL/TLS certificate management and deployment
- Connection Draining/ De-registration Delay
  - stop sending new request after 1-3600 seconds

ALB Traffic

- IP (must be reachable by VPC), instance, lambda
- when route to IP, use any private IP
- when route to instance, use primary private IP

ALB request tracing

- add header to trace individual request(traffic flow)
- `X-Forwarded-For` to trace original IP instead of ALB(level 7) ip
  - NLB(level 4) do not modify incoming connection and keep original IP

P.128 Auto Scaling Group(ASG)

- auto scale in/ out(with min, max limit and desired limit) to match demand
- auto register new instance if old are unhealthy
- free, charge applied on EC2 instance
- define launch template + scaling policy & capacity
  - once deployed cannot be modified
  - then create new launch template and update ASG
- monitor (custom) metric in CloudWatch Alarm
  - target tracking e.g. average CPU usage, RequestCountPerTarget, network in/out
  - simple/ step, add 1 unit if CPU > 80%

  - scheduled
  - AWS has predictive scaling
  - cool down: 300s to stabilize metrics

- in EC2 health check type, default is EC2 (do not terminate if failed), change to ELB to auto replace unhealthy instance

P.139 RDS (Relational Database Service)

- DB *managed* by AWS
  - Postgres, MySQL, MariaDB, Oracle, MS SQL, AWS Aurora
  - In MS SQL, support Transparent Data Encryption(TDE), two-tier key(master + data key)
- automatic patching
- continuous backup and restore to specific timestamp(point in time)
- read replica
- multi AZ for disaster
- scaling vertical/horizontal and backup by EBS
- auto scale storage space
- read replica(up to 15)
- replication is async, can within AZ/ cross AZ(free) or cross region($)
- standby instance(auto replicate data), auto route to standby if failure occurs, cannot route manually.
- multi-AZ for Disaster Recovery
  - sync replication, better availability, not for scaling
  - how to sync?
    - take snapshot, restore in new AZ

RDS Authentication

- password
- IAM user/role
  - generate token instead of password, valid for 15 mins
  - MySQL/Postgres only

RDS backup

- automatic backup, limited to one region
- manual backup across multiple region

RDS resource usage

- RDS Enhanced Monitoring, get metric from agent on in RDS instance (more accurate), can check # of process/threads
- CloudWatch get metric from the hypervisor for a DB instance

P.147 Amazon Aurora

- AWS own database, support psql, MySQL
- 5x performance over MySQL, 3x in psql
- auto grows from 10GB to 128TB, 15 read replicas, high availability
- 20% expensive but more efficient
- at least 6 copies across 3AZ , 4W+3R=6 (1 master for W+R)
- self healing, auto expanding, cross region replication
- RDS Proxy
  - allow sharing DB connections, must be accessed from VPC
  - reduce load and failover time

P.153 ElastiCache

- fast, in-memory cache for Redis(Read Replica, backup/restore) and Memcached
- set up is complicated
- (DB cache) app ask for ElastiCache, if cache hit then good, otherwise query from RDS and write to ElastiCache
- (user session) app write session to ElastiCache, other app retrieve session data
- Redis support complex data type, data persistence options by snapshot, transactions
- Memcached is key-value pair, no transactions (more memory efficient)
- *Redis(more complex, more function!)*
  - cluster mode(horizontal scaling, higher performance, with multi-key issues)
    - cluster must locate in same region, and use multi-AZ
    - suffer from data loss even if cluster is enabled
    - cannot promote any replica to be primary

P.158 Caching

- Lazy loading VS Write Through

- in lazy loading, request data in cache, if no then load from db and cache it, only write into DB (good for read frequent and write less often)
- in write through, write to both cache & DB to ensure data consistency

- cache evictions
    - explicit delete it
    - memory full and LRU policy
    - set time-to-live (if using lazy loading)

- ElastiCache for Redis VS MemoryDB for Redis
    - for cache, persistency not required -> ElastiCache
    - primary data store, durability, high availability, and strong consistency -> MemoryDB

P.166 Route 53

- DNS managed by AWS
- Record Types
    - hostname -> IP
        - A, map hostname to IPV4
        - AAAA, map to IPV6
    - hostname -> hostname
        - CNAME, map hostname to another hostname with A/AAAA record
        - alias record, route traffic to AWS resources only
    - name records(NS), allow DNS to know where should route to
    - Records TTL(time to live)
- Hosted Zones
    - private - with VPC
- CNAME(Canonical Name) vs Alias
    - both for routing
    - CNAME cannot route from root domain name, cause charge
    - Alias is Route-53 specific, free, can map root domain or subdomain to AWS resource
    - Alias cannot custom TTL, cannot map to EC2 DNS
- Routing Policy
    - If multiple values returned, client random select one
    - monitor up to 256 child health checks by many checkers, can custom (OR/AND/NOT)
    - Failover, route to other place if health check fail

  - weighted/ latency based
  - by geo-location (country)
  - by geo-proximity, with custom bias to shift traffic
  - by IP-range
  - multi-value (up to 8), helps map to health resource only/ some load balance(not to replace LB)
- Domain register != DNS service, but usually can provide

P.197 VPC - Virtual Private Cloud

- inside a AZ, there is public subnet and private subnet(not accessible outside)
- use route table to define access
- Internet Gateway(IGW) exposes outside AZ, help VPC instance connect to Internet.
- one subnet can only have one route table
- IGW <-> public subnet
- IGW <-> NAT gateway(AWS managed)/ NAT instance(self managed) <-> private subnet
- network ACL(NACL), firewall controls traffic from/to subnet, by IP, stateless(in/out independent), process rule by order
- security group, controls ENI/EC2 instance, allow rules only, stateful(if can in then can out), process all rules
- Flow Logs
  - VPC flow log
  - Subnet flow log
  - Elastic Network Interface flow log
- log to S3, CloudWatch log, kinesis data firehose
- VPC peering (private connect two VPC)
  - cannot have overlapping CIDR (cannot duplicate IP)
  - not transitive, if A<>B<>C then A cannot <> C
- VPC Endpoint, connect between VPN and AWS service using private network
  - VPN endpoint gateway, S3, DynamoDB
  - VPN endpoint interface
  - used with VPC
- Site to site VPN, encrypt over public

- Direct Connect(DX) physical private cable

P.215 S3

- bucket in region, name must be unique in global
- object key: s3 path
- 5GB-5TB, use multi-part upload
- tags for security/lifecycle, with version ID and metadata
- security controlled by user-based, resource-based(bucket policy = json, object/bucket ACL)
- allow = IAM permission ALLOW or resource policy ALLOW + no explicit deny

(S3) Access Control List

- service policies that allow you to control which principals in another account can access a *resource*

P.229 S3 Replication

- with versioning enabled, support Cross-Region(CRR) or Same-Region(SRR)
- only new objects are replicated after enable
- S3 Glacier storage class, lower cost if not accessed, -> Deep Archive
- but may take longer retrieval time, up to 2 days
    - standard, standard IA, intelligent tiering, one-zone IA, glacier instant ...
- use intelligent tiering to shift storage classes (with cost)
- use lifecycle rules for transition and deletion

P.240 EC2 Instance Metadata

- allow EC2 get info without IAM role, userdata = launch script
- AWS CLI = Python SDK (boto3)
- AWS CLI `--test` to validate permission and test api request

P.259 S3 Event

- S3:ObjectCreated, Removed, Restore, Replication event -> SNS, SQS, Lambda
- need to set IAM permission to those service to get S3 event
- Amazon EventBridge, support filter with JSON rules, multiple destination, archive/replay event
- PUT/READ limit is by file prefix (/folder1/sub1/)
- GET techniques

    - multi-part upload for 5GB+, transfer acceleration outside->edge location->s3 bucket in target region
    - parallel GET/ get partial data
    - S3/ Glacier Select, like SQL, to perform server-side filtering
    - metadata/ tags not searchable, need external db

- CORS, default block other origin traffic
- S3 access logs store to specific logging bucket
- pre-signed URL, allow temporary access/upload
- S3 access point(with own DNS name and access policy) for file security management

S3 Encryption

- server-side
    - s3 managed keys (aws fully control)
    - kms keys in aws kms (customer have control)
    - customer provided (must use HTTPS)
- client-side
- header
    - `'x-amz-server-side-encryption': 'AES256'` for SSE-S3 (server side encrypt with S3)
    - `'x-amz-server-side-encryption': 'aws:kms'` for SSE-KMS(with KMS)
    - `sse:s3`, `sse:kms` are descriptive terms in AWS contexts, not for header

S3 Object Lambda/ Access Point

- access point
    - use with better control for multiple apps/ teams
    - dedicated policy without affect main bucket
    - create S3 Access Point
    - then create S3 Object Lambda Access Point
- object lambda
    - must use with object lambda
    - run on getItem() request, with lambda function, return processed data
    - usually use to redact PII(personally identifiable information)/ add extra
    - cost: storage + lambda request

S3 Select

- use SQL statement to filter S3 object, reduce data transfer

S3 Lifecycle policy

- delete object after certain time
- move object to different storage class (eg to Standard-IA, Glacier) to reduce cost

S3 Api

`aws s3api list-objects` - `--page-size` to limit number of object per page (internal, does not affect output) - `--max-items` to limit number of object to list - response will include `NextToken`, use `--starting-token` with that token to continue

S3 Eventually/Strong Consistent

- S3 object is strongly consistent, immediately available after write
- S3 bucket is eventually consistent, list bucket after delete may show old result

P.288 CloudFront

- cached at edge to improve network performance, with DDoS protection, shield and firewall
- origin can be S3 or HTTP(ALB/EC2/S3 static/HTTP backend)
- cloudfront vs S3 Cross-Region Replication(limited to several regions)
- can control TTL (force update by CloudFront Invalidation), and cache type
- cache key: hostname + resource portion
  - can include header, coolies, query string with cache policies (None/Whitelist/All)
  - all include in cache key also include in origin request
- max cache hit: separate static(S3) and dynamic(ALB. EC2)

CloudFront failover

- CloudFront always route all traffic to primary, even if previous request fail. (not like unhealthy instance)
- CloudFront fail over to secondary origin for GET, HEAD, OPTIONS

CloudFront Keys

- control private content distributed in CloudFront
- user cloudfront key pair to create signed URL/ signed cookies
- only be created by AWS root account, at most 2 pair
- cloudfront signed URL by trusted key group/ by AWS account
  - private key: for app to sign URL, public key for verify
- used to distribute paid content, time-limited access

- pricing, with different tiers class(include less edge location)
- origin group: increase availability. If primary fail then go to secondary
- field level encryption(encrypt at Edge, closer to user)
- logging, cloudfront -> kinesis data stream -> lambda(real time) OR kinesis data firehose(near real)

P.317 Docker

- docker image store on docker hub or Amazon ECR(public/private)
- ECS/ EKS/ Fargate/ ECR
- Fargate + EFS = serverless, with auto scaling applied
- CloudWatch metric -> CloudWatch alarm -> scale ECS
- updates: rolling update
- trigger: other event/schedule -> EventBridge -> run ecs task -> ...
- Task Definition(ECS): JSON about how to run a docker container, and define multiple container/ data volume
  - pod specifications is for K8S

P.355 Amazon EKS

- managed K8S cluster on AWS, good to use if already using K8S in other cloud/ on-premise
- with logs and metrics using CloudWatch container insights
- node types:
  - managed node groups
  - self-managed nodes (using EC2)
  - aws fargate, no maintenance and nodes needed
- data volumes: EBS, EFS, FSx

P.361 Elastic Beanstalk

- code as infrastructure, consistent across different env/apps
- managed service, auto handle capacity provisioning, load balancing, scaling, application health monitoring, instance configuration
- can support different deployment methods but less control than CodeDeploy
- enable X-Ray daemon in `xray-daemon.config`

Deploy flow

---

create app -> upload new version -> launch env -> manage env

- support many language or use custom platform
- for dev: single instance, 1 RDS
- for prod: ALB(work) or ELB(web), 2AZs, many EC2s and RDS

Deployment

- all at once: with downtime
- rolling (with additional batch to maintain traffic flow)
- immutable: new instance in new ASG, then put into current ASG ans remove old instance
- blue green: create new env, then swap traffic slowly. no downtime. will have DNS change, hight cost
- traffic splitting: canary testing

Practice

- support at most 1000 versions, use lifecycle policy to delete old version
- define different environment for dev, test, prod
- `eb config` for environment-specific settings, eg instance type, environment variables, or load balancer configurations

Deployment rollback

- all at once, rolling: redeploy old version
- immutable: terminate new instance
- blue green: swap url

console: upload zip -> create new version -> deploy cli: create new version using cli -> upload zip

RDS with elastic beanstalk Practice

- better to separate, don't bind to EB lifecycle
- create RDS DB snapshot, choose protect RDS from deletion, create new EB without RDS...

Custom env

Put `.config` files (in json/yaml) in `./ebextensions` dir to customize EB env

- when the environment is deleted, all resources created will be deleted

Other Files

- `cron.yaml` for scheduled task
- `Dockerrun.aws.json` for multi-container docker
- `env.yaml` for environment variables, solution stack
- `appspec.yml` for manage deployment

Worker environment

- process background/ long running task
- SQS queue(store message to be processed) + EC2 instance(with ASG) + IAM role

Lifecycle policy

- auto remove old old version, by version number or days (so does not reach quota)
- can set retain the source bundle(.zip) in the S3 bucket

P.383 CloudFormation

- infrastructure as code, declarative way to outline infra. CF will create with *right order*
- template store in S3, editing = create new version
- manual: edit in CF Designer
- automated: edit template in yaml, using cli to deploy

CloudFormation CLI code

- `cloudformation package` packages the local artifacts (local paths) that your AWS CloudFormation template references. The command will upload local artifacts, such as your source code for your AWS Lambda function
- `cloudformation deploy` deploys the specified AWS CloudFormation template by creating and then executing a changeset

CloudFormation Template

```
AWSTemplateFormatVersion: "version date"

Description:
  String

Metadata:
  template metadata

Parameters:
  set of parameters (fixed value, no conditionals, conditionals use in resource!)

Rules:
  set of rules

Mappings:
  set of mappings

Conditions:
  set of conditions
```

```
Transform:
  set of transforms

Resources:
  set of resources

Outputs:
  set of outputs
  `Export` param to allow another stack to use
- in other template, use `Fn::ImportValue`. (`!Ref` for inside the same template
only)
```

CF helper script(python)

- `cfn-init`: retrieve and interpret resource metadata, install packages, create files, and start services
- `cfn-signal`: signal with a CreationPolicy or WaitCondition, so you can synchronize other resources in the stack when the prerequisite resource or application is ready
- `cfn-get-metadata`: retrieve metadata for a resource or path to a specific key
- `cfn-hup`: check for updates to metadata and execute custom hooks when changes are detected

Code in CF

- `{ "Fn::FindInMap" : [ "MapName", "TopLevelKey", "SecondLevelKey"] }`

P.419 CF Stack creation

- stack creation fail: delete everything, look at log
- stack update fail: roll back to last working state
- can trigger stack events to SNS(email, lambda...)
- update stack: AWS create change set, you view the change set and make changes.
- cross stack VS nested stack
  - cross stack: with different lifecycle
  - nested stack: component are reused, bundles components
- StackSet
  - create and manage stacks across multiple accounts, regions using admin ac
  - update all associated stack at onceTG
- CF Drift: check manual configuration changes
- stack policy: set whitelist for resources change, if not used then all resources can be updated

CF StackSet/ ChangeSet

- stack set allows you to create, update, or delete stacks across *multiple* AWS accounts and regions with a single operation
- change set is the preview the impact of deploying a new stack

P.429 Communication between services

Drawback of sync: cannot handle sudden traffic Async: decouple applications to scale automatically

SQS (queue)

- unlimited throughput, no of message, max 256KB
  - use SQS extended queue (store in S3) for large message
- low latency, retention default 4 days, can set to 1 minute -14 days, with encryption
- may have duplicate message
- Standard Queue: out of order
- FIFO Queue: in order, 300 transactions per second, 300 x 10 transactions per second with batching (exactly-one send)
- remove all messages: PurgeQueue
- message sent is persist until consumer delete it
- poll SQS message, up to 10 at a time, then send delete api
- can receive in parallel
- SQS queue -> Cloudwatch metic -> Alarm -> ASG increase EC2 instance
- has SQS access policy, cross-account access
- message visibility: if one poll message, then invisible for a while, if not deleted then visible again, default 30s
- if fail many times, move to DLQ (dead letter queue), set retention period, good for debug
- DLQ can re-enter into source queue
- Delay Queue: 0-15 minutes
- long polling: 1-20s, wait for message to arrive, reduce api call/cost
- more size? SQS extended client(Java)
- batch API to reduce cost
- deduplicate: hash or include message id

- FIFO: message group id, to ensure order within group, no guarantee between groups
  - `MessageDeduplicationId` if same ID received within 5 mins, the message will be accepted and not delivered
  - `MessageGroupId` guarantee in group message are in order

SNS (pub/sub)

- 100000 topics, 12.5 subscriptions per topic
- topic push(like mqtt) or direct push(mobile apps SDK)
- with encryption, SNS access policy
- with Json policy for filtering
- fully decoupled, no need to know who is subscriber
- allow data persistence, delay processing, retry
- works with SQS in other regions
- fan-out pattern, S3 new object -> SNS topic -> different SQS queue/ Kinesis Data Firehose/ lambda f()
- FIFO: supported, SNS FIFO topic -> SQS FIFO queue

Scaling in queue

- calculate backlog per instance metric (how long a message have to wait to be consumed)
- target tracking scaling:
  - if backlog > threshold then increase instance in ASG
- ECS auto scaling is alternative

P.463 Kinesis

- handle data streaming in real time
- Data streams
- Data Firehose
  - load data stream into data store, data streams -> firehose
- Data Analytics
  - analyze data with SQL/ Apache Flink
- Video streams
- Firehose is often more cost-effective for straightforward ingestion to AWS services
- Data Streams may be more cost-effective for complex processing or custom consumers

Kinesis Agent

- Java app run on EC2/ on-premise server
- collect data from instance and stream to Kinesis Data Stream/ Firehose

PutRecord/ PutRecords

- Put records write multiple records *without ordering guaranteed*
- if need ordering, use PutRecord with *SequenceNumberForOrdering*

Kinesis Data Streams

- retention 1-365 days, can replay, default is 1 day
- if data is inserted, cannot delete
- data shares the same partition goes to same shard
- producer: AWS SDK/ Kinesis Producer Lib/ Kinesis Agent
- consumer: Kinesis Client Lib/ AWS SDK/ lambda, firehose, analytics
- control with IAM policy, encryption
- VPC endpoint available for kinesis to access within VPC
- monitor API call? CloudTrail
- Shards
- producer: upload up to 1mb, then put into shard using "partition key"
- same partition key -> same shard
- consumer: lambda, analytics, firehose, SDK, KCL
- standard consumer: 2MB/s per shard across all customer
- enhanced: 2MB/s per shard per consumer, lower latency, expensive. 5 consumer per stream
- high input: more shards
- high output: enhanced fan out
- Kinesis Client Lib(KCL)
  - 1 shard can be read by 1 KCL
  - progress checkpoint into DynamoDB (need IAM)
  - KCL run on EC2, Beanstalk, on-premise
  - in order at shard level
- merge shard: merge two lowest, cannot merge three at once

Data Firehose

- full managed, pay for data
- destination: redshift, S3, OpenSearch, MongoDB, Elasticsearch(ES), HTTP endpoint
  - cannot send to ElastiCache
- time/ data limit
  - 60s minimum if batch is not full
  - or 1MB data at a time
- support data transform using lambda
- can forward to S3

Data Analytics

data streams/ data firehose -> data analytics -> (1) data streams, -> EC2, lambda (2) data firehose, -> S3, Redshift/ ...

P.491 Monitoring

CloudWatch

- CloudWatch metrics, get <=30 dimensions/ metric to monitor CPU, RAM, traffic use, etc (custom metric)

- default 1 min interval, can set up to 1s

- `Period` = length of time to evaluate metric

- `Evaluation Period` = number of points to determine alarm

- `Datapoints to Alarm` = number of points to trigger alarm within `Evaluation Period`

- CloudWatch Log

  - Log insights, search and analyze log data

  - support query lang

  - non real time

  - subscribe with subscription filter -> lambda, data firehose(to store), data stream -> EC2, lambda

  - subscription filter: each for specific "Error" Keyword, if found ->SNS

  - metric filter

- CloudWatch Alarm

  - watch over a single metric
  - for composite metric, use composite alarm (AND, OR supported)
  - send notification to SNS topic/ Auto Scaling
  - stop/ terminate/ reboot/ recover EC2, trigger ASG

- Alarm watch EC2 instance status check. Fail -> recover instance & alert with SNS

AWS X-Ray

- debugging in production and distributed system
- troubleshoot performance/ bottlenecks, understand dependency by create service map graph
- tracing the traffic of request(segments)
- can trace every/ sample request
  - default: 1st request each second(reservoir) + 5% request(rate)
- in applications, modify some configuration to use X-Ray SDK
- can integrate with Elastic Beanstalk
- X-Ray daemon on device
- X-Ray SDK in application code
- X-Ray subsegment: more detailed log
- trace = many segments
- segment = per service/ app level log
- segment = many subsegments

X-Ray sampling

- Reservoir limit(at most how many sample per second) + rate limit (% after exceed reservoir limit)

X-Ray on different case

- on EC2, X-Ray daemon
- on lambda, import SDK in code, select x-ray integration in configuration
- on Elastic Beanstalk, use `.ebextensions/xray-daemon.config` or in console
- on ECS/EKS/Fargate, use docker image run daemon and set up port mapping
- also need proper IAM roles
- use UDP with port 2000

-`namespace` = aws for AWS SDK call, remote for downstream call

- `GetTraceSummaries` get list of trace id & annotations
- `BatchGetTraces` get list of trace segment

Filtering in X-Ray

- annotations <K,V>, indexed for trace, for search

- metadata <K,V>, for record custom data

X-Ray env variable

- `_X_AMZN_TRACE_ID`
- `AWS_XRAY_CONTEXT_MISSING`, behavior is tracing header is missing
- `AWS_XRAY_DAEMON_ADDRESS`, send data to X-Ray daemon

CloudTrail

- audit API calls made by user/service/AWS console
- CloudTrail Insight: create baseline measures, finding unusual calls
- api call -> send event to EventBridge -> SNS email alert

P.549 Serverless

AWS Lambda

```python
def handler_name(event, context):
    ...
    return some_value
```

- event = input data
- context = runtime info, e.g.

```
- lambda name
- invoked_function_arn
- get_remaining_time_in_millis
- log_stream_name
- cognito_identity_id/ cognito_identity_pool_id
```

- functions, short executions, on-demand, auto scaling
- support many languages, or use custom runtime API for others
- trigger by some event(eg network call/ EventBridge)
- alias can point to versions only
  - canary: configure the alias to send 10% traffic to new version. if fail then reset
- lambda permission: execution role, not resource-based policy

Lambda Execution Role, Resource Policy

- execution role: permission to access other AWS service from lambda
- resource policy: who can invoke lambda function

Lambda Best Practice

– separate the Lambda handler (entry point) from core logic – use Execution Context reuse to improve performance – use AWS Lambda Environment Variables to pass operational parameters – control dependencies in function's deployment package.

Price

- per request + per duration
- charged at (memory) GB-seconds in 1ms measurement.
- *more memory configuration* -> *more cpu*, cannot directly allocate CPU
- 1792 MB ram = one full vCPU, more than it can use multi-thread
- default 3s timeout, max 900s

How to trigger Lambda

- ELB/ALB
- API Gateway
- CloudFront
- services, e.g. Amazon Cognito, AWS Step functions

Lambda error

- 504 INTEGRATION_FAILURE/ INTEGRATION_TIMEOUT
- 429 502 Throttling
- 502 no function associated with it
- 403 Unauthorized

Lambda Invocation Type (Sync/ Async)

- RequestResponse, sync, wait for response
- Event, async
- `aws lambda invoke` *NO* `invokeAsync`

Internet

- by default lambda has access to internet + AWS services
- deployed to VPC, place lambda in private subnet, need NAT instance/ gateway in public subnet, then configure route table

Lambda Authorizer

- input: caller identity, support OAuth, SAML
- output: IAM policy, to control which client have access to resource

ALB/ API Gateway(Sync)

- pass HTTP request to JSON, contain path, header, body, GET/POST
- support multiple value in header(one key many value)
- register lambda in target group
- sync function, JSON->HTTP to user
- need to set up permission = resource policy for api to invoke lambda

Event->Lambda(Async)

- S3/ SNS/ Cloudwatch event->event queue(allow retry) ->trigger lambda.
- Set up SQS/SNS if process is fail.
- useful EventBridge rule: CRON or rate(some time schedule)

Event Source Mapping

- connect data source(SQS queue/ Kinesis stream/ DynamoDB stream) to lambda
- periodic polls event source and consumed by lambda
- used where events are generated async, processed in serverless
- allow batch operation and auto scaling
  - kinesis data stream/ DynamoDB stream: on lambda invocation per shard
    - batch processing: 10
  - SQS standard: 1000 batches of message at most
  - SQS FIFO: lambda scales with # of active message groups(GroupID)
- if using kinesis batch, in-order within each shard(partition key)
- if fail, entire batch is reprocessed,
  - discard old events/ restrict retry/ split batch
  - then send to Destination(DLQ or trigger another lambda)
  - DLQ set up on SQS queue, not lambda

Event object

- data(input arguments) from invoking service, e.g EventBridge
- source, region, IP, input

Context object

- property of invocation, runtime env, e.g id, function_name, memory

Destination

- async event: SQS/ SNS/ Lambda/ EventBridge bus
- event source mapping(batches): SQS/ SNS

IAM Roles

- LambdaBasicExecutionRule - upload log to CloudWatch, read event data in event source mapping

  - use CloudWatch Metrics to check invocations, durations, error/success
- Lambda Kineses/SQS/VPC/... read from Kinesis/ deploy to VPC ...
- resource-based policy to give AWS service to access to lambda
- Enable X-ray in lambda configuration(active tracing) and use in code
  - need IAM role, AWSXRayDaemonWriteAccess

Lambda env var

- store secrets(encrypted by KMS, Lambda service key...)

CloudFront Functions vs Lambda@Edge

- CloudFront Fun, modify headers, redirecting, customizing viewer request/response, validate JWT
- use JS, sub-ms startup, 5s limit, small memory
- handle millions request/s, 1/6x cost of Lambda@Edge
- Lambda@Edge custom response in viewer & *origin* request/response
  - CloudFront Function for viewer request/response only
- NodeJS, Python, more flexibility
- 30s limit, 1000 request/s
- author functions on us-east-1, then AWS replicates to others
- consider CloudFront Function first
  - cache key normalization
  - header manipulation
  - URL rewrite/ redirect
  - request authentication, authorization, eg create/validate JWT
- use Lambda@Edge for
  - depends on 3rd lib, e.g. other AWS service
  - network access to external, or file system

Lambda with VPC

- default outside VPC, cannot access resources
- define VPC ID, subnet, security group -> lambda create ENI(Elastic Network Interface) in your subnet (with AWSLambdaVPCAccessExecutionRole)
- no internet, unless have NAT Gateway(allow private subnet to access outside)/ use VPC endpoint to access other AWS service(e.g. DynamoDB)

- CloudWatch Logs still works without endpoint/NAT

Lambda Execution Context

- temporary runtime env, good for db, HTTP client
- maintained for some time if there is another lambda call
- can reuse resource
- use lambda layers to manage/reuse code
- put db connection, http client out of the inner function
- use /tmp space, up to 10GB, remains when execution context is frozen

Storage Comparison

- ephemeral storage /tmp, 10GB, inside single invocations, Permission: Function
- lambda layer, 250MB, static, share across invocations, Permission: IAM
- S3, Permission: IAM
- EFS, Permission: IAM + NFS

Concurrency/ Throttling

- 1000 concurrent session, can set "reserved concurrency" level
  - set some capacity for own AWS service to avoid external users use up all executions
- throttle
  - sync: throttle error 429
  - async: retry, then DLQ
  - retry with exponential

Reserved Concurrency

- max number of concurrent sessions, set in lambda function

Provisioned Concurrency

- cold start takes some time to init resources
- with provisioned concurrency($$$), resources is allocated.

3rd party lib

- install packages alongside code and zip together
- <50 MB then upload to lambda, else S3
- aws sdk comes with lambda function

with CloudFormation

- inline function in CloudFormation
  - put in [Code.ZipFile] but no extra dependency

- S3
    - need to update S3 Bucket/Key/Object Version so that CloudFormation will update

Container

- deploy lambda as container in ECR, implement Lambda Runtime API
- container image size max 10GB
- test containers locally using the Lambda Runtime Interface Emulator
- use AWS-provided Base image, multi-stage build, single repo for function with large layer to avoid uploading and duplicates.

Version

- Tag **$LATEST**
- each version has own ARN(resource name)
- Aliases: point to specific version. cannot reference aliases.
    - use in Canary development, or splitting env eg dev, test, prod

CodeDeploy(within SAM)

- linear: Linear10PercentEvery3Minutes, shift traffic gradually
- canary: Canary10Percent5Minutes, then 100% traffic shift
- all at once
- define in AppSpec.yml, specify name, alias, currentVer and targetVer

URL

- access through URL in public network only
- can apply to alias or $LATEST (not point to versions)
- set resource-based policy, CORS to allow different domain access
- for public access, AuthType=NONE
- AuthType=AWS_IAM for identity-based & resource-based policy

Runtime Performance

- CodeGuru Profiler, activate from lambda console

Deployment

- `CodeDeployDefault.LambdaAllAtOnce`, all at once
- `CodeDeployDefault.LambdaCanary10Percent5Minutes`
- `CodeDeployDefault.LambdaLinear10PercentEvery1Minutes`

CodeDeploy Hooks

1. ApplicationStop
2. DownloadBundle (download new app version)
3. BeforeInstall
4. Install

5. AfterInstall (configuration or change file permissions here)
6. ApplicationStart
7. ValidateService

P.625 DynamoDB

- Relational DB support join and aggregations scale vertically(more powerful CPU), horizontal(more reading capability by adding EC2, RDS read replica)
- NoSQL eg MongoDB, DynamoDB, does not support query join(limited join), no aggregation, scales horizontally.
- full managed, replication across multiple AZ, auto-scaling
- standard/ Infrequent access IA class
- made up of tables, structure, key: value(small document, <400KB)
- infinite items, each item has attribute
- auto delete data with TTL

Keys

- Partition Key(Hash), e.g. random UserID
- Partition Key + Sort Key, combination is unique
- different partition key -> store in different disk (consistent hashing)
- sort key(range key) -> support range query (less than, between)
- use Partition Key + Sort Key for query
- filter on other attributes = scan entire table
- only 1 partition key, 1 sort key -> use LSI

https://www.youtube.com/watch?v=Y8gMoZOMYyg 5:45

Capacity

- provisioned capacity, specify R/W and pay for it and set "burst capacity"
- on demand: scale automatically, more $ if the usage is stable and known
  - 2.5 more expensive than provisioned

DynamoDB backup

- on demand/ point-in-time recovery
- backup to S3 internally but not accessible

Read/ Write Capacity Unit

- RCU/WCU applied to single table
- one write per second for item up to 1KB, e.g. 2x item size 4.5KB /s = 10 WCU
- RCU: one strong read/ two eventually consistent read, 4KB
- default to eventually consistent read, and change to strong but 2x RCU required
    - "ConsistentRead" : True
- data is partitioned, # of partitions depends on capacity and size

Throttling

- reason: hot key, partition, large item
- solution:
    - distribute keys as much as possible
    - exponential backoff
    - RCU issue -> ElastiCache/ DynamoDB Accelerator(in memory cache), providing ms latency

W/R data

- PutItem(), UpdateItem(), can make Atomic Counters, Conditional Write
    - put item: create or replace an item by new item
    - update item: create or update an existing item
- GetItem() using primary key(hash/ hash+range)
    - use Projection Expression to retrieve only certain attributes
- BatchGetItem(), retrieve at most 16 MB of data from up to 100 items
    - return `UnprocessedKeys` if exceed limit
- Filter Expression, which item
- Projection Expression, which attribute, eg `value.prop1`
- Condition Expression, which to update
- Expression attribute names, placeholder in projection expression
- Query based on Hash Key + Option sort key value(=, >= between)
- return up to 1MB data
- Scan entire table, consume many RCU
    - use parallel scan to reduce time
    - use limit to limit impact
    - can use Projection Expression/ Filter Expression
- local secondary index? global secondary index

- DeleteItem, can perform conditional delete
- DeleteTable, faster than DeleteItem
- `Scan` default is sequential, 1MB at a time, but can set to parallel (can have rate control)

WCU consumed in `PutItem`, `UpdateItem`, `DeleteItem`

- `ReturnConsumedCapacity:TOTAL` to return WCU consumed
- `ReturnConsumedCapacity:INDEXES` to return WCU consumed and subtotals for the table and any secondary indexes

Access Control with IAM Policy

- `dynamodb:LeadingKeys` limit on partition key
- `dynamodb:Select` limit on scanning
- `dynamodb:Attributes` limit on columns

Conditional Expression(Write)

- attribute exist/ not exist/ type
- contains
- begins with
- size
- (with version number for optimistic locking)
- common use: `attribute_not_exists(partition_key)`, `attribute_not_exists(partition_key) and attribute_not_exists(sort_key)`
- filter expression for read
- Batch Operation
  - Batch WriteItem(PutItem/ DeleteItem), no update
  - Batch GetItem

PartiQL

- support structured/relational and semi-structured data
- support DynamoDB and MongoDB
- syntax like SQL
- support INSERT, UPDATE, SELECT, DELETE

Global Secondary Index/ Local Secondary Index

- LSI
  - defined on creation time only(create new table, then migrate)

  - no extra cost
  - act as an extra sort key for query

- GSI

  - can define after creation
  - clone primary using GSI as new partition key
  - keep two tables in sync if there is any changes
  - GSI needs uniform data distribution
  - define RCU/WCU separately and throttling, better to have WCU on GSI > WCU on main
  - write to main table -> N+1 x cost (update two tables if N GSIs)
  - eventually consistent (consistency/ availability trade off)
  - with separate metrics https://www.youtube.com/watch?v=ihMOlb8EZKE

Transactions

- all-or-nothing across tables
- Read mode: eventually consistent
- Write mode: standard, transactional, 2x WCU, RCU used

DAX

- in-memory cache for DynamoDB, 5 mins TTL, multi-AZ
- compatible wth DynamoDB API
- use ElastiCache for aggregation result, DAX for original data

DynamoDB streams

```
- changed items-> streams-> Kineses data streams/ Lambda/ Kinesis
- can stream old/new image/both, or keys
- EventBridge cannot detect table-level changes in DynamoDB
- need `AWSLambdaDynamoDBExecutionRole` permission in lambda
```

Time to Live

- define a "number" with unix epoch, no extra cost at deletion, delete within 48 hrs from both LSIs and GSIs

Write types

- concurrent/ atomic/ conditional/ batch(twice record)

DynamoDB Global table

- use "last writer wins", no locking strategy
- in normal table, use optimistic locking to prevent concurrent writes

App Design

- client login with Cognito/SAML/...
- client get temporary AWS credentials, obtain IAM role and access table
- fine-access control: Cognito, assign IAM role with condition
- limit in row-level access, limit to specific attribute

P.679 API Gateway

- invoke lambda
- expose HTTP endpoint in backend
- expose any AWS api

Endpoint Type

- edge-optimized(default)
  - routed in edge locations, lives only in one region
- regional
- private, access from VPC using ENI
- user auth using IAM roles/ Cognito
- if using custom domain name HTTPS, cert must be in us-east-1 for edge-optimized
- deploy in stage variables("v1", "v2", "dev"), indicate to different version of a lambda
- passed into "context" object in lambda, ${stageVariables.variableName}
- canary deployment: separate some traffic, with separate logs & metrics(blue/ green)
- use mapping templates for request/response

Integration Types

- MOCK
  - return response without sending the backend
- HTTP/AWS (lambda/ aws services)
  - need to set *mapping templates*
  - can rename/ modify query string params, add headers
  - modify body content
  - method request: before sending to backend
  - integration request: after receiving from backend
  - integration response: before sending to client

    - method response: after receiving from client

- HTTP_PROXY

    - no mapping

- AWS_PROXY

    - no mapping template
    - request pass to backend, response forward by API gateway
    - add API key to HTTP header if needed

API Gateway Usage Plan

- control rate/ stages/ methods of API access with different API keys
- can configure for specific key
- support lambda authorizer, IAM role, Cognito
- 1 API key many usage plan, 1 usage plan many stage.

API Gateway Authentication

- resource policy
- IAM roles & policy/ IAM tag
- lambda authorizer
    - support many methods but not STS
- Cognito user pool
- cannot integrate with STS directly

Cache Control

- enable/disable cache in console
- client add `Cache-Control: max-age=0` header
- in per-key cache invalidation, set `Require Authorization` to prevent unintended invalidation

API Gateway -> CloudWatch

- call STS to assume IAM role, then log data to CloudWatch

P.724 CICD

AWS CI/CD series - AWS CodeCommit – storing our code - AWS CodePipeline – automating our pipeline from code to Elastic Beanstalk - AWS CodeBuild – building and testing our code - AWS CodeDeploy – deploying the code to EC2 instances (not Elastic Beanstalk) - AWS CodeStar – manage software development activities in one place - AWS CodeArtifact – store, publish, and share software packages - AWS CodeGuru – automated code reviews using Machine Learning

CodeCommit

- private git, no size limit, use IAM policy(user/role)

- Supported credential
    - Git credential (IAM generated)
    - SSH key
    - AWS access key

CodePipeline

- visual workflow to orchestrate CI/CD
- automate the build, test and deploy process
- can add manual approval step(action) as a stage (add a manual step + SNS)
- artifacts
    - files/ data produced in/ used in pipeline stages
    - store in S3, pass information in pipeline actions
    - with versions
    - can include source code, build output, deployment files
- use CloudWatch events for execution state changes
    - fail -> give information in console
- use CloudTrail for audit API calls

CodeBuild

- CI, with `buildspec.yaml` for setup
- `$env`
    - variables
    - parameter-store, from SSM parameter store
    - secrets-manager, from AWS secret manager
- phases
    - install/ pre_build/ build/ post_build
- artifacts
    - upload to S3 (need S3 permission)
    - if need to encrypt, use `CODEBUILD_KMS_KEY_ID` with KMS
- cache
- testing: run CodeBuild locally with CodeBuild Agent(container image)

CodeDeploy

- CD service, deploy to EC2, on-premise server, lambda, ECS
- configure with appspec.yaml
- auto rollback if fail
    - rollback = redeploy previous working version as new deployment
- many options over deployment steps

On EC2 instance/ On-premise Server

- Pre-requisites: run CodeDeploy Agent on target instance, and permission granted
- AllAtOnce/Half/One (In-place, may have downtime)
- Blue-Green
    - create new auto-scaling group (ASG)
    - Generate new instance, when all ready, move whole system to new version
    - must be using an ELB (to route traffic and zero downtime and monitor health)
- CodeDeploy agent = bridge between the CodeDeploy service and the instance, can do many stuff
- CodeDeploy agent use HTTPS on 443

On Lambda

- automate traffic shift for lambda alias (All, Canary, Linear)

On ECS

- only Blue(old)/Green(new) deployment (All, Canary, Linear)

CodeStar

- integrated solution of CI/CD series
- quickly create CI/CD projects for EC2, lambda, Beanstalk
- limited customization

CodeStar vs CodeBuild+CodeDeploy+zCodePipeline

- Codestar setup the toolchain more rapid, suitable for small project.
- use separate tools if need more granular control

CodeArtifact

- managed artifact repository to store/publish/share software packages (like npm maven)
- use proxy to access AWS CodeArtifact and build and CodeBuild
- when package version changes, send event to EventBridge
- use resource policy to access permission

CodeGuru

- ML-powered service for auto code review and application performance recommendation

CodeGuru Reviewer

- identify critical issue, security vulnerabilities, bugs
- language supported:
    - Java
    - Python
    - JS/TS

- C# etc

CodeGuru Profiler

- identify code inefficiency, decrease compute cost, provide heap summary, anomaly detection
- use Agent Configuration to customize stack depth(method A call B call C...), report time, sample interval

P.762 SAM

- framework for develop/deploy "serverless" app
- configured in YAML, support all operation in CloudFormation
- use CodeDeploy for lambda
- need to add permission to lambda function
- SAM framework natively use CodeDeploy to update lambda (alias)
  - In CloudFormation template, add transformer header to indicate SAM `Transform: 'AWS::Serverless-2016-10-31'`

  SAM Template(YAML) -> CloudFormation Template -> upload to S3 -> CloudFormation (CF Stack)

  - Code `AWS::Serverless::Function / Api / SimpleTable`
    - Function: Lambda
    - Api: API Gateway
    - SimpleTable: DynamoDB
    - Application: (nested app)
  - Deploy -`aws cloudformation validate-template`
    - check is valid json/yaml `aws cloudformation package, aws cloudformation deploy`
    - package = upload local artifact to S3
- Pros
  - Build app locally `sam build` and debug (CloudFormation does not)
  - locally start AWS lambda, invoke lambda, start API Gateway endpoint and generate AWS event for lambda
    - not replicating AWS execution environment, need to set up local profile and roles
    - use `aws configure` command with `--profile` to add profile
    - in lambda use `sam local invoke` with the `--profile`
  - provide lambda-like env and support many IDE (using AWS Toolkit)
- `sam build`, resolve dependencies, build artifacts
- `sam deploy`, deploy app with specific CloudFormation stack
- `sam sync`, sync local changes to aws, faster for developing
- `sam local invoke` test lambda

- `sam local start-lambda` start local endpoint to test lambda

SAM resource types:

- `AWS::Serverless::Api Application Function HttpApi`
- `AWS::Serverless::LayerVersion SimpleTable StateMachine`

P.774 CDK

- define cloud infrastructure(vs serverless in SAM) using programming language(vs json/yaml in SAM)
- code is compiled into CloudFormation template(transformed in SAM)
- deploy infrastructure and application runtime code together (good for lambda, docker in ECS)
- create template in CDK -> create stack -> (build) -> synthesize stacks -> deploy stacks
- use SAM cli to test CDK apps(lambda) locally through `cdk synth`
- or using CDK assertions module
  - fine-grained test: test for specific resources, variables, rule, conditions, permissions
  - snapshot test
- `cdk synth`, synthesizes a CF template from CDK

CDK Constructs

- can represent single AWS resources(eg S3), multiple resources, or even applications
- three level hierarchical structure, construct can include other constructs
  - L1 = low-level, direct CloudFormation resources
  - L2 = mid, usually boilerplate code
  - L3 = high, patterns, eg API gateway backed by lambda
- can reuse for making infrastructure patterns

CDK Bootstrapping

- create CDKToolKit (Cloudformation Stack), contain S3 bucket and IAM roles
- one-time operation to set up necessary infrastructure for CDK to manage resources
- use before deploying to cloud (set up necessary resources)

P.788 Cognito

- fully managed serverless identity service

User Pools = Authentication

- sign-up, with email/SMS confirmation (or with social identity Facebook/Google/SAML/OpenID)
- sign-in, verify email/pw (or 2FA) and issue JWT token

- User Pool will trigger lambda on certain actions
- service provide can use the JWT token, pass to Cognito User Pols and evaluate
- integrate with API gateway/ ALB
- allow custom UI
- provide adaptive authentication to classify high risk login and require MFA
- JWT token = header + payload + signature(verify signature by asking user pool)
  - contain `cognito:username`, groups and roles etc

Integrate with ALB

- In ALB, add a listener rule to authenticate with Cognito (perform authentication before forwarding request)
- if authenticated, then forward to target group
- otherwise redirect to login
- user login with User pools
- user exchange JWT token with temporary AWS credentials in Identity Pool
- Identity pool validate with User Pool/ SAML/ OpenID, and generate temporary AWS credentials with Security Token Service (STS)
- user access AWS resources directly with temporary credentials

Identity Pools = Authorization (federated identity)

- provide aws credentials (usually temp credentials) to user to access AWS resource directly
- integrate with Cognito User Pool
- user policy variables to control user access
  - policy variable = dynamic calculated resources
- allow unauthenticated/ guest access

P.811-915 others

P.811 Step Function

- state machine written in JSON to manage workflow
- each step is a state (invoke AWS service/ run activity)
- transition: move state to another state

Task State

- InputPath -> Parameters -> Process -> ResultPath -> OutputPath

Usage

- data processing (multi step)
- DevOps/ automation
- microservices orchestration

States

- result of one state is passed to next state in array with `ResultPath`
- Choice State -Test for a condition to send to a branch (or default branch)
- Fail or Succeed State - Stop execution with failure or success
- Task: map `x` to `f(x)` (dynamic)
- Pass State - Simply pass its input to its output or inject some fixed data (return static value)
- Wait State - Provide a delay for a certain amount of time or until a specified time/date.
    - wait time can be dynamic but need to specify when runs
- Map State - Dynamically iterate steps.'
- Parallel State - Begin parallel branches of execution. When all parallel state finish, move to next state.
- *Retry* mechanism is inside a state, but not an individual state
- `WaitForTaskToken`, pause until task token is returned (usually from AWS services)

Error Handling

- retry failed state(default 3 times) / transition to fail path
- catch: if `ErrorEquals` matches the error, move to a specific state
- with error code: timeout/ task fail/ permission/ all

Step function activity task

- use activity task to integrate with external workers/process (/long running)
    - e.g. run in EC2/ lambda/ mobile
    - paused execution
    - wait for external worker to poll `GetActivityTask` and complete
    - when worker is running, send `SendTaskHeartBeat` to keep task active
    - worker complete task and send result (SendTaskSuccess/Failure)
    - continue state machine with received result

Standard/ Express(Async/ sync) Workflow

- standard run exact 1 execution only, at most 1 year, price per state transition
    - standard support activity (external worker, human approval)
- express limit to 5 mins, many executions, high throughput. Price by executions, duration, memory.

P.825 AppSync

- use GraphQL to combine different data sources(Dynamo, Aurora, Lambda, HTTP)
- use CloudFront in front of AppSync for HTTPS
- retrieve data in real-time with WS/MQTT on WS
- use API_KEY/ AWS_IAM/ openid/ cognito user pool

P.829 Amplify

- quick tools to create mobile/web app with AWS services
- Authentication/ Datastore/ Host webapp
- run E2E test before pushing code to prod

P.835 Security Token Service(STS)

- enable limited, temporary limited-privilege credentials for AWS IAM users or for users that you authenticate (federated users)
- cannot integrate with API gateway
- default timeout is 3600s, range 900-3600s

P.857 KMS, Encryption SDK, SSM param store, Secret Manager

Key Management Service(KMS)

- manage encryption keys (in many AWS services)
- auto rotate backing keys for CMK annually
- environment variable: max 4KB

KMS Direct Encryption

- encrypt at most 4KB data directly with KMS key within KMS per API call

KMS Envelope Encryption

- KMS generate data key -> encrypt large data (usually on client side)
- data key is then encrypted with KMS master key(= customer master key, CMK)

KMS API

- `Encrypt` encrypt small data using CMK
- `GenerateDataKey` return plaintext & encrypted, use plaintext key to encrypt data (!this is envelope encryption)
- `GenerateDataKeyWithoutPlaintext`, generate and store key for future use

- `CreateKey`, generate a CMK, only for 4KB data, costly

Encryption SDK

- client-side encryption

SSM Parameter Store

- free in standard tier
- store config data, secret, password, API key (general, static variables, config values, password)
- no resource based policy, control with IAM policy
- standard param
  - no Expiration, ExpirationNotification policy
- advanced param
  - with NoChangeNotification, ExpirationNotification policy
  - emit notification to EventBridge

Secret Manager

- have monthly cost
- store secret, password, API key
- rotate automatically (used in db, api keys, OAuth)
- use resource-based policy attached to each secret
- IAM policy with specific ARN to individual

Miscellaneous Knowledge

AWS Trusted Advisor

- real-time guidance to help you provision your resources, optimize AWS infrastructure, improve security and performance, reduce costs

AWS Security Hub

- centralizes security alerts and compliance status across AWS accounts

Amazon Inspector

- automated security assessment service to improve security and compliance

AWS Serverless Application Repository (SAR)

- managed repository for serverless app
- can be public/ private/ shared across account
- contains pre-built app

AWS Marketplace

- general online store for 3rd party apps

AWS service catalog

- allow large company to manage approved IT (AWS) services

AWS System Manager

- management service to automatically collect software inventory, apply OS patches, create system images, and configure Windows and Linux operating systems

IAM Access Analyzer

- identify resources (eg S3, IAM roles) shared with external entity, with custom zone

Access Advisor feature on IAM console

- identify unused roles and help remove
- principle of least privilege

Billing and Cost Management

- IAM have no access to cost management, except administrator activate IAM user access

AppSync/ Cognito Sync

-Cognito Sync - *single* user sync data (1MB for single response) between devices and the cloud (serverless)

- AppSync
  - *multiple* users sync data in real time on shared data

Macie

- detect sensitive data in S3 objects
- `SensitiveData:Credentials/CustomIdentifier/Financial/Personal/Multiple`

EventBridge / SNS for event-driven

- SNS for high throughput/ low latency message/ high fan out
- EventBridge for reacting to SaaS/ AWS service

Lambda, Version, alias, stage, Traffic Shift

- version: snapshot of lambda

- alias: point to specific version

- stage in API gateway: point to alias
- Traffic Shift for alias
  - alias point to additional version
  - gradual rollout
  - cut over to new version
- Traffic Shift for stage
  - point to different alias
  - canary version (!not lambda)
    - point to different alias
    - split traffic from main/ canary version

Periodic Jobs

- Cloudwatch Events(EventBridge), schedule cron jobs/ fixed interval -> trigger lambda (easy!)
- or step function
- or ec2 instance with cron

Pseudo parameters and regular parameters (in CloudFormation)

- regular
  - user-defined, provided when creating/ updating stack
  - defined in "Parameters" section in template
  - string, number, list
  - can have constraints, default value
- pseudo
  - system-defined
  - e.g. Region, StackName, StackId, NotificationARNs

Global accelerator

- provide global static IP, route traffic to nearest endpoint

Cross account access

- Account A make an IAM role to access the resource
- Account A attache a trust policy to the role that identifies account B as the principal who can assume the role
- Account B administrator delegate permission to assume the role to other account in B

MFA

All of them give temporary security credentials

- `GetSessionToken` support MFA, for IAM users
- `GetFederationToken` does not support MFA, for a federated user

- `AssumeRoleWithWebIdentity`, give temporary security credentials for federated user using public identity providers, eg Facebook
- `AssumeRoleWithSAML`, give temporary security credentials for federated user using SAML

Interface /Gateway Endpoint

- Interface endpoints use private IP and create ENIs in your subnets
- Gateway endpoints use route tables
- Gateway endpoint for S3/ DynamoDB only

EC2 Service Role(Instance Profile)

- special kind of IAM role, attached when instance is launched
- better than using IAM role.

Kinesis Data Stream + Lambda

- lambda is triggered in sequence, one shard only one lambda.

CloudWatch Agent

- collect log files, stream data to CloudWatch Logs in near real time

`awslogs` log driver

- allow container to send log to CloudWatch Logs

S3 Multipart upload with SSE:KMS

- each part is encrypted then upload to s3
- s3 need to decrypt, combine and encrypt again
- need `kms:Decrypt` permission

AWS Distro for OpenTelemetry

- tracing for distributed system (like X-Ray)

Web Application Firewall(WAF)

- return 403 forbidden error or redirect to custom page
- can filter by parameter value, IP,...

Amazon Pinpoint

- push notification, SMS, email, voice message to customer

AWS IAM Identity Center

- support SAML 2.0 for external authentication to login to AWS