AWS Developer Exam Cheat Sheet

Metrics to remember:

Max lambda concurrency	1000 (900 reserved, 100 unreserved)
Lambda concurrency calculation	Request per second * duration per function E.g 100rps * 5s = 500 concurrent units Kinesis: concurrent requests = #shards E.g 100 shards = need 100 concurrent units
Max lambda timeout duration	15 minutes
Max API gateway concurrency	10,000 (before throttling kicks in)
Max S3 object size	5TB
Lambda /tmp folder size	512 MB
DynamoDB RCU calculation	4KB per strongly consistent read (2x for eventually consistent reads) Scenario: Client wants to read up to 30 rows of customer data per second. Each row is about 8KB. How many RCU is needed for strongly consistent reads? RCU per row = 8KB / 4KB = 2 RCU Total RCUs needed = 2 RCU * 30 = 60 For eventual reads, divide answer by 2
DynamoDB WCU calculation	1KB per write (2x for transactional write) Scenario: Client wants to store 100 rows of customer data per second. Each row is 8KB. How many WCU is required? WCU per row = 8KB * 1 = 8 WCU Total WCUs needed = 100* 8 WCU = 800
SQS retention period	14 days

Http error codes to remember

429 (too many requests)	Happens when number of requests exceed API gateway or lambda function limits
504 (Gateway timeout)	API gateway (29 seconds) -> can be configured API gateway + lambda = Lambda taking longer than 29 seconds…

Lambda tips

- To improve cpu performance, increase the memory (yes, you read that right…)
- To reuse db connections or cached expensive operations, use execution context
- To process data locally, use /tmp folder
- To embed code in cloud formation template, use Zipfile syntax under code.
- Use traffic shifting if you want to shift certain percentage of traffic to current and new versions (weight alias)
- Use environment variables if you want to have different configurations per aliases
- Use aliases when you want to segregate your lambda function by env (e.g prod, stage, test)
- Use dead letter queue if you want to handle failed async lambda function
- You can configure async lambda function by setting invocation type to event
- Lambda layer is used for packaging libraries and other dependencies for lambda function. Using layers means you can keep your lambda footprint small.
- For non-supported programming language, use custom runtime
- For VPC access
  - Configure the subnet of the VPC to access
  - Configure the security group of the VPC
  - Make sure VPC has a NAT gateway so that lambda can access the internet

API Gateway tips

- API Cache
  - Capacity
  - Encrypted: Y/N
  - Cache TTL
  - Require authorization
  - Handle unauthroized requests
- To clear cache
  - Set header to `Cache-Control: max-age=0`
  - `Grant permission to user`

```
"Effect":
"Allow",
      "Action": [
"execute-api:InvalidateCache
"],
Resource": [

"arn:aws:execute-api:region:
account-id:api-id/stage-name
/GET/resource-path-specifier
"]
```

- To handle environment specific configuration, use stage variables
- Use HTTP integration when requests/response needs to be mapped
- Use HTTP proxy when mappings are not required
- Use lambda authorizer for request or token based authorization

X-Ray tips

- To enable X-ray:
    - EC2: user data script
    - ECS: create docker instance with X-ray
    - Beanstalk: enable x-ray inside .ebexensions/xray-daemon.config

```
option_settings:
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```

- Annotations: key-value pairs data that can be queried via filter expression in X-ray
- Metadata: Similar to annotations without the query/searchable functionality
- Segments/Subsegments:
    - Allows call graph to be more detailed by exposing inner call graphs
- GetTraceSummaries: Fetch trace summaries that contains trace ids and annotations
- BatchGetTraces: Fetch trace details from the trace id. Contains segment documents.
- Remember:
    - X-ray is used for troubleshooting and not logging.

DynamoDB tips

- Serverless NOSQL database
- Local Secondary Index
    - Same key as primary but with different sort key
    - Can only be defined when table is created
- Global Secondary Index
    - Different key, sory key can be anything
    - Query supports eventual consistency only
    - Read and Write capacity should be >= table, otherwise you will hit the provisioned throughput exception
- Use DAX to improve read performance to microseconds. Note that DAX costs $ and lookout for specific keywords in the exam such as cost-effective (which means DAX should be ruled out in favor of something else)

DynamoDB…

- DynamoDB is serverless and is per region
- Use BatchWriteItems and BatchReadItems when batching operation is required for performance reason
- Use TransactWriteItems when ACID is required (All or nothing)
- Use atomic counters when you want to implement auto increment
- Use conditional write when you want to update database only when certain conditions are met
- Projection expression allows scan/query to return desired attributes instead of whole table
- Use DynamoDB streams for you need to react to CUD operations, such as auditing, writing data to another dynamo db, etc..
- Use parallel scan to improve scanning performance if there is enough read capacity
- Use page size and max items to control the number of rows that are read at a time (and to avoid timeouts in the AWS CLI)
- Use TTL to save space/costs when the objects can be deleted after certain period of time.
- DynamoDB supports provisioned and on demand throughput.
- ReturnConsumedCapacity can be passed in the write request to get number of WCU used.
    - INDEXES: Include indexes
    - TOTALS: Just the main table

Elastic Cache

- Both supports data partitioning
    - Distribute data among multiple nodes to improve scalability
- MemCache
    - Multi-threaded: Can leverage multi-cores to handle more requests
    - Should be preferred as it is simpler unless availability is a factor.
- Redis
    - Supports replication-> highly available

Elastic Cache…

- Always read the question carefully to determine whether the question contains ‘highly available’ or ‘multi-threaded’
- In general, Elastic Cache is an easy way to improve performance when workload reads a lot of data from database and query is expensive.
- Remember the followings:
    - Write through cache: Whenever data is updated, the cache is updated
    - Lazy loading: Only cache on demand

AWS KMS

- Key management system (crypto)
- Types
    - AWS Keys
        - Managed by AWS for AWS services.
        - Rotated every 3 years
    - Customer managed Keys
        - KMS keys in AWS managed by user
        - Rotated every 1 year and can be configured
- Symmetric
    - 256 bit encryption key that never leaves AWS unencrypted.
    - To use, call AWS KMS
    - The same key is used for encryption and decryption.
- Asymmetric:
    - Pair of public/private keys
    - Public is used for decryption outside of AWS (also within)
    - Private is used for encryption inside AWS
- CloudHSM: hardware-based security module that allows user to generate and use keys on the AWS Cloud. If the questions mentions ‘hardware’, the answer will be this.

AWS KMS…

- GenerateDataKey
    - Returns both plaintext and encrypted copy of data key
- GenerateDataKeyPlainText
    - Returns plaintext copy of data key
- Use client-side encryption library
- (AWS Encryption SDK) to encrypt data using plaintext key. The output is encrypted data key and encrypted output. Remove the plain text key from memory.
- Use AWS SDK to decrypt the data key. The data key becomes a plain text file. Then use client side encryption library to decrypt the encrypted data using the plain text data key. Remove the plain text key from memory.

Others

- Encrypt: Encrypt data smaller than 4kb. For anything larger, use envelope encryption.

Cognito

User pools

- Use for sign in and sign out functionality
- Can integrate with social identity provider (idP) such as Facebook
- Default login page
    - Only logo can be changed

Identity pools (federated identities)

- Provides temp credentials for accessing AWS services

Cognito sync:

- Sync user profiles across multiple devices

Cloudformation

- StackSets: Share stack across multiple accounts
- To prevent accidental deletion, use termination protection
- To import value exported by another stack in outputs, use fn::ImportValue
- To package and deploy,
    - aws coudformation package
    - aws cloudformation deploy
- Cfn-init: Run scripts, install packages, start services, etc..

SAM

- Framework for building serverless application
- Consists of
    - SAM template specificaiton
    - AWS SAM CLI
- Resources
    - AWS::Serverless::Api
    - AWS::Serverless::Application
    - AWS::Serverless::Function
    - AWS::Serverless::HttpApi
    - AWS::Serverless::LayerVersion
    - AWS::Serverless::SimpleTable
    - AWS::Serverless::StateMachine
- Commands
    - AWS SAM init
        - Init a SAM project
    - AWS SAM build
        - Build a SAM project
    - AWS SAM deploy
        - Deploy a SAM project
- Cloudformation can use transform command to take template written in AWS SAM Syntax and transform it to cloud formation compliant template.

SQS

- Always use long polling to save costs
- Use message deduplication id in the provider to avoid duplicates. Any duplicates sent within 5 minute window will be ignored
- If a function execution time is longer than the message visibility timeout, the message that is currently being consumed will be visible to other consumers. To avoid this situation, set visibility timeout to longer period
- Use SQL delay to delay the message being visible to consumers after it is created. This is useful for situation where you need to delay sending message so that something happened before the message is sent (e.g payment is processed in another system)
- SQS FIFO
    - Exactly once processing
    - Preserves order
- SQS Standard
    - At least once processing

CloudWatch

- Default time period: 5 min
- To increase interval to every 1 min, use detailed monitoring
- To increase interval to under minutes, use high-resolution metric
- Ops:
    - PutMetricData
        - Creates or updates metrics
    - PutMetricAlarm:
        - Creates or updates alarm
- Alarms:
    - Period: Granularity of period
    - Evaluation period: last number of periods to evaluate
    - Datapoints to alarm: Number of data points within evaluation period to trigger the alarm.
- Dashboard:
    - Allows user to create dashboard for monitoring resources across region

CloudWatch...

- Namespace: Container for cloudwatch metrics. eg . AWS/<services>
- Dimensions:
    - name/value pair that identifies a metric
    - You can find metrics to a specific ec2 instance using InstanceId dimension.
- Useful metrics to remember:
    - Latency: How slow is API overall
    - IntegrationLatency: How slow is the backend (e.g lambda)
    - CacheHitCount
    - CacheMissCount

Beanstalk

- Best for deploying scalable web apps written in Java,Go, .Net, PHP, NodeJS, Ruby and Docker and resources are automatically managed (deployment, capacity provisioning, load balancing, auto scaling and health monitoring)
- Configs are stored in .ebextension folder
- Beanstalk worker environments are great for handling background tasks. Can be configured to fire via SQS using daemon. Alternatively, it can be triggered using CRON via cron.yaml
- Deployment:

Types	Capacity reduction	Rollback impact	Speed
All at once	Full	Full	Fastest
Rolling	Minor	Minor	Fast
Rolling with batches	None	Same as rolling	Slow
Immutable	None	None	Slowest

CodeDeploy

- On premise
    - In place deployment

- EC2/ECS/Lambda
    - In place deployment
    - Blue/Green
        - Canary
        - Linear
        - All at once

Code Commit

- Similar to GitHub. If you understand GitHub, you will understand GitCommit
- Can be accessed
    - using IAM User with the right permissions. User will need to associate public key with their IAM User. In addition, user will need to setup GitCredentialHelper to use AWS Creds.
    - Git credentials: HTTPS connection using username and password

CodeBuild

- a fully managed continuous integration se that compiles source code, runs tests, and produces software packages that are ready to deploy.
- Similar to Bamboo build or Jenkins
- Config is named Buildspec.yml

ECS

- Container orchestration service that run dockers applications
- Task definition (configures docker)
- To use X-ray, create docker container with x-ray SDK
- Cluster query language: group containers by expressions
- Tasks Placement strategy:
    - Random: Tasks placed randomly while still respecting the constraints.
    - Binpack: Reduce #instances
    - Spread: Spread evenly

Kinesis Stream

- Shards
    - Cold shards:
        - Underutilized
        - Suggest to merge (decrease capacity)
    - Hot shards:
        - Overutilized
        - Suggest to split (increase capacity)
- Performance:
    - #shards = #compute instances
- Data is kept in the shards for only 24 hours

S3/Cloudfront tips

- To encrypt data at rest, use default encryption
- To ensure data is encrypted at rest when uploading,
    - Appropriate bucket policy to deny s3 action when condition x-amz-server-side-encryption is neither true nor AES256
    - make sure x-amz-server-side-encryption is set to true in header
- Transferring large data in S3
    - Use S3 content transfer acceleration where possible when name is DNS compliant
    - If name is not DNS compliant (contains dots), use Multi-Upload
- If uploading encrypted file via multi-upload
    - Make sure user have aws-decrypt permission, otherwise you will get error
- To allow only certain users from accessing url (members, eg.), use pre-signed url
- To improve performance for global audience, use cloudfront
- To enable SSL for Cloudfront
    - Viewer-protocol-policy
        - Https only
        - Http to https
    - Origin protocol policy
        - Https only
- To control access at object level, use ACL

I AM

- Role: An IAM identity that has specific permissions. Similar the user but can be attached to service/person/applications
- Principal: Person or application that can make requests for an action on AWS resource
- Policies: Permission(s) for an action. Eg S3:PutObject
    - Identify: Allow/deny actions on resource X. Attaches to an identity
    - Resource: Attaches to a resource (eg. s3, SQS, VPC endpoints, etc.)

- Troubleshooting
  To diagnose policy/permission issue, use IAM Policy Simulator
- How does assume role work?

Define trust policy allows a principal to assume a role.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": { "AWS":
"arn:aws:iam::123456789012:root" },
        "Action": "sts:AssumeRole"
    }
}
```

Create role and specify trust policy:

```
aws iam create-role
--role-name example-role
--assume-role-policy-document
file://example-role-trust-poli
cy.json
```

You can attach a policy to a role using attach-role-policy

```
aws iam attach-role-policy
--role-name example-role
--policy-arn
"arn:aws:iam::aws:policy/Amazo
nRDSReadOnlyAccess"
```

I AM…

You then call `aws sts assume-role` to assume the role

```
aws sts assume-role --role-arn
"arn:aws:iam::123456789012:role/examp
le-role" --role-session-name
AWSCLI-Session
```

AWS STS (Security token service)

- AssumeRole
    - Used for AWS users or roles with existing creds.
- AssumeRoleSAML
    - Used with SAML
- AssumeRoleWithWebIdentity
- GetSessionToken:
    - Used with MFA
- GetFederationToken
    - Used for custom integration
    -

Delegating access between 2 accounts. Eg user in account dev access s3 bucket in prod

- On prod account, create I AM role and specify dev account as trusted entity
- Set policy that will grant access to S3 for the I AM role above

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
"Principal": {"AWS":DEV-ACCOUNT-ID},
      "Action": [
        "s3:*"
      ],
      "Resource":
"arn:aws:s3:::productionapp"
    }
}
```

- On dev account, create policy to assume IAM role in prod. Attach that policy to the user.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": "sts:AssumeRole",
    "Resource":
"arn:aws:iam::PRODUCTION-ACCOUNT-ID:role/Up
dateApp"
  }
}
```

Others…

- AWS SWF
    - Orchestrate workflow. Eg. business process
    - Markers can be used to record events for application specific purposes.
- AWS StepFunctions
    - Runs serverless workflows
    - To pass output as input to the next step, use ResultPath
- Parameter store vs Secrets manager
    - If the question is about RDS and rotations of secrets are required, the answer is always Secrets manager
    - Keys can be in hierarchical form:
        - foo/bar, foo/hello, foo
- Question about customers with multiple projects in AWS.
    - Solution: Splitting projects by accounts and then use consolidated billing
- Lambda@Edge is good for use case where you have global users and you want to run lambda function closer to the user for performance reason. E.g authentication workflow
- AWS Cloud9 is an integrated IDE that allows you to write, run and debug your code in browser
- Question about ECS cluster where instances are intermittently failing health check
    - Solution: Increase the healthcheck period
- Question about ECS cluster where many instances are launched too soon and then scaled back due to over-provisioning
    - Solution: Use cool down (for simple scaling policies), otherwise warm up can also be used since instance that is warming up is not added to the auto-scaling group.
- EC2
    - Metadata
        - Allows user to retrieve private and public IP
    - User data: Run script

General tips

- Always pay attention to the following keywords as that will dictate different solutions
  - Availability
  - Costs
  - Performance
  - E.g DAX is the fastest performing option but may be more costly than changing code to use query instead of scan. etc..
- For hard questions, sometimes AWS will intentionally give misleading answers
  - Rule out the least likely by scrutinizing each one
  - Whatever remains is probably the answer

Best practices

- Development with AWS services
  - In general, Serverless > Beanstalk > ECS > EC2
  - In AWS's world, DynamoDB > RDS
- Deployment
  - Always prefer blue/Green deployment to avoid capacity reduction and impact to customers
- Monitoring / Troubleshooting
  - X-ray SDK for troubleshooting your AWS applications, especially if you rely on many AWS services
  - AWS CloudTrail for logging/auditing
  - AWS CloudWatch for monitoring system metrics
- Refactoring:
  - RDS: Use elastic cache or read replicas to improve read performance.
  - DynamoDB: Always use query (with global or secondary indexes). Use pagination to limit amount of data being transferred. DAX should be used if costs is not a factor.
  - Use retry with exponential backoffs to avoid overloading a server that is under load.
  - Always use optimistic locking on situation where there are multiple updates on the same row

Best practices….

  - To optimize performance for workload involving lambda functions and an immediate response is not required, consider using async lambda functions to leverage parallelism
- Security:
  - Principles of least privilege
  - Never ever use root account access keys for every-day tasks (better if its deleted)
  - Https please