AWS Certified developer associate

- Courses: Acloudguru, Udemy
- Practice exams: Braincert (DVA-C01), Udemy, Whizlabs

Table of contents

- Compute
  - EC2
  - ECS
  - Lambda
- Monitoring
  - Cloudwatch
  - X-Ray
- Networking
- Security
  - IAM
  - Cognito
  - STS
  - Encryption
  - Systems manager
  - OpsWorks
- Notifications
  - SQS
  - SNS
  - SES
  - Kinesis
- Storage and databases
  - S3
  - EBS
  - RDS
  - Aurora
  - DynamoDB
  - Elasticache
- CloudFormation
- Elastic beanstalk
- Deployment
- Amazon API Gateway
- Developer tools
  - CodePipeline
  - CodeCommit
  - CodeBuild
  - CodeDeploy

Compute

EC2

- By default, user data runs only during the boot cycle when the instance is first launched. And by default, scripts entered as user data have root user privileges for executing, so they don't need sudo command
- Burstable performance instances, such as T3, T3a and T2, are designed to provide a baseline level of CPU performance with the ability to burst to a higher level when required by the workload. If your AWS account is less than 12 months old, you can use a t2.micro instance for free within certain usage limits

For reserving instances, you can reserve it for an AZ (zonal RI) or region (regional AI). Zonal RIs provide a capacity reservation but regional don't.

EC2 only allows in-place or blue/green deployment. A placement group enables instances to interact with each other via high bandwidth, low latency connection.

For detailed monitoring, install Cloudwatch agent on the machine and then configure it to send the server's logs to a central location in Cloudwatch.

AWS Data Pipeline automates data movement and transformation. It can transfer logs or other data from EC2 instances to S3 buckets.

```
# launch an instance while enabling detaile dmonitoring
aws ec2 run-instances --image-id ami-09092360 --monitoring Enabled=true
# enable detailed monitoring for an existing instance
aws ec2 monitor-instances --instance-ids i-1234567890abcdef0
```

> Autoscaling

- When creating from the console, only basic monitoring by default. When creating through CLI/SDK, detailed monitoring by default.
- If one instance exceeds capacity, ASG creates a new one. If it's unhealthy, the AS group will terminate it first, and create a new one after
- ASGs attempt to distribute instances evenly between the AZs that are enabled for the ASG
- ASG can contain EC2 instances in multiple AZs within the same region, and cannot span multiple regions
- By querying the instance metadata, it's possible to get the private IP address of the instance.

ECS

A scalable container orchestration service supporting docker. It uses Fargate for serverless, or you can use EC2 for more control. First upload the image to ECR, elastic container registry. ECS connects to ECR and deploys the images.

To log in, run `aws ecr get-login`, use the output to login to ECR, and then pull the image with `docker pull REPOSITORY URI : TAG`

- To allow the container to access SQS, the policy must be attached to the ECS Task's execution role
- To use X-Ray with ECS, create a separate Docker image to run the X-Ray daemon, add instrumentation to the app code for X-Ray and configure and use an IAM role for tasks.
- If the container should share storage, use Fargate launch

- When you use ECS with a load balancer deployed across multiple AZs, you get a scalable and highly available REST API.

Lambda

Lambda is priced based on number of requests (first 1M free, then $0.2 per 1M requests) and duration. Lambda automatically scales out, not up.

Function

With an alias you can create many versions of Lambda functions. `$LATEST` is always the last version of code you uploaded to Lambda. Use Lambda versioning and aliases to point the apps to a specific version if you don't want to use `$LATEST`. If the app has an alias instead of `$LATEST`, it won't automatically use new code when you upload it. `$LATEST` can't be configured in an ARN, use Alias instead.

With aliases, you can route some traffic to the new Lambda version.

To reuse execution context, package only the modules the function requires and move the initialization of the RDS connection outside the handler function. To avoid hard-coding information, use environment variables.

Environment variables can't exceed 4KB, and you can use as many as you want. so if encrypted data must be passed to the function, use envelope encryption and reference the data as file within the code. Using an envelope encryption, the network load is much lower. For lower files, you can use Encryption SDK and pack the encrypted file with the Lambda function.

For temporary storage that won't be necessary after the function finishes, use /tmp directory up to 512MB and delete the files at the end of the function.

A Publish version is a snapshopt copy of a function code and config in the latest version. Config can't be changed and it has a unique ARN which can't be modified.

Limits

- Maximum execution duration for a Lambda request is 900s=15mins
- The maximum deployment package size is 50MB, and the max size of code/dependencies zipped into a deployment package is 250MB
- Lambda has a limit to the number of functions that can run simultaneously in a region. Default is 1000 per second per region, the error you get is `TooManyRequestsException`, HTTP status code 429. Avoid recursion!
- Maximum RAM is 10GB

Make sure the dependencies are included in the deployment package, along with the code, to improve startup time

Monitoring and security

With **step functions** you can visualize serverless applications, they automatically trigger and track each step so if something goes wrong you can track what went wrong where. With step functions you can track flows executed by Lambda functions, and you can control multiple Lambda functions. With Step functions state machines you can orchestrate the workflow

With Lambda authorizer, a 3rd party authorization mechanism is used to invoke the Lambda function.

Lambda works on S3 events asynchronously and for asynchronous invocations, Lambda retries function errors twice. If the retries fail, you can use dead letter queues to direct unprocessed events to an SQS queue.

> Concurrency: # requests that a Lambda function is serving at a time.

If a function is invoked while a request is still undergoing, another instance is allocated, increasing the functions' concurrency. Reaching the concurrency limit leads to latency bottlenecks. To enable the function to scale without fluctuations in latency, use provisioned concurrency. By allocationg this before an increase in invocations, you can ensure that all requests are served by initialized instances with very low latency. You can also configure application auto scaling to manage provisioned concurrency on a schedule or based on use. To increase provisioned concurrency automatically as needed, use the app auto scaling API to register a target and create a scaling policy.

You can set up reserved concurrency for the Lambda function so that it throttles if it goes above a certain concurrency limit. This is used to limit the maximum concurrency for a given Lambda function.

If Lambda is working with Kinesis data streams, it works synchronously. If it encounters errors, it doesn't proceed execution and retries until the error is resolved or the data expired.

Lambda & X-Ray, no need to install daemon in Lambda. Only the IAM role assigned to Lambda should have access to X-Ray.

Monitoring

- CloudTrail: monitors API calls in the AWS platform. Logs all authenticated API requests to IAM and STS APIs.
- AWS config records the state of your AWS env and can notify you of changes
- AWS Budgets needs approximately 5 weeks of usage data to generate budget forecasts

Cloudwatch

- host level metrics: CPU, network, disk and status check
- Custom metrics: can be used for RAM, # logged-in users. Min 1min granularity
- By default, Cloudwatch stores the log data indefinitely, but you can change the retention
- CloudWatch metrics: they tell the rate at which the function/code is executing, but can't help with debugging the error
- Cloudwatch logs: Lambda pushes all logs to CW Logs
- You can retrieve data from any terminated instance after its termination
- For alarms that should be triggered only if it breaches 2 evaluation periods, set data points to alarm as 2 while creating the alarm
- CloudWatch integration feature with S3 allows to export log data from CW log groups to S3
- *Cloudwatch events*: when the AWS environment changes, AWS resources can be triggered. You can create custom events or events generated periodically.
- Log group data is always encrypted in CloudWatch logs, with the optional choice of using KMS for this encryption with the customer master key with the CLI command `associate-kms-key`

X-Ray

Prerequisites: install X-Ray daemon in the instance, create IAM role, give X-Ray permission: xray:PutTraceSegments and xray:PutTelemetryRecords and instrument the app with daemon and SDK.

Analyzes and debugs **distributed** apps' performance, identify and troubleshoot performance issues and errors. Provides an end-to-end view of requests as they travel through the app, and shows a map of the app's underlying components.

X-Ray can be used to collect data **across AWS** accounts. For that, create a role in the target unified account and allow roles in each sub-acc to assume the role. Configure the X-Ray daemon to use an IAM instance role.

X-Ray sampling allows to control the amount of data to be recorded, and modify sampling behavior on the fly without modifying or redeploying your code. By default, X-Ray records one request/s and 5% of any additional request per host.

> Requests sampled per second = reservoir_rate + fixed_rate/100 * (total_req/s - reservoir_rate)

For reservoir rate of 60, fixed rate of 20% and total requests of 200, `60 + 0.2*(200-60) = 60 + 28 = 88`

X-Ray integrates with ELBs, Lambda and API gateway. Provides:

- Interceptors to add to the code to trace incoming HTTP requests
- Client handlers to instrument AWS SDK clients that the app uses to call other AWS services
- An HTTP client to instrument calls to other internal and external HTTP web services

To ensure that X-Ray daemon is correctly discovered in ECS, use the SDK AWS_XRAY_DAEMON_ADDRESS env variable.

Networking

4xx errors are due to missing required parameters in a request, or exceeding a DB table's provisioned throughput. 5xx errors are due to unavailable services, or network issues.

> Elastic load balancer, ELB

- **ALB**, application load balancer: proxies HTTP and HTTPS requests. Slightly slower than NLB, and doesn't scale quickly. with host/path based routing, it can handle multiple domains. It doesn't support TCP passthrough
    - ALB access logs: detailed info about requests sent to the LB. Each log contains the time the request was received, the client's IP address, latencies, request paths, server responses and API connections
    - ALB request tracing: tracks HTTP requests. The LB adds a header with a trace identifier to each request it receives.
    - If the target groups have no registered targets, the error shown is HTTP 503.
    - THe ALB only sees the LB IP address, not the user's public IP. But it can obtain this by X-Forwarded-For header.
    - Sticky sessions route requests to the same target in a target group. The clients must support cookies. Stickiness restricts app elasticity, it's not so scalable.
    - If the app has no cookie management, let the LB generate a cookie for a specified duration
- **NLB**, network load balancer: routes network packets, balance TCP traffic where extreme performance is required, handle millions of requests per second. Allows TCP passthrough. Cannot handle path based routing. They can capture the user's source IP address and source port without using X-Forwarded-For
- Classic LB: load balance HTTP/HTTPS apps and user layer7 specific features, and also use strict layer4 load balancing for apps that rely purely on TCP protocol. If it returns 504, it means the gateway has

timed out. The app might be failing in the web server or db server.

If users lose session or need to re-authenticate often, use ElastiCache Cluster, which provides a shared data storage for sessions that can be accessed from any individual web server.

> Route 53

Amazon's DNS service, maps domain names to IP addresses (EC2 instances, load balancers or S3 buckets). For a specific domain name, specify CNAME.

Weighted routing can associate multiple resources with a single (sub)domain name and choose *how much traffic* is routed to each resource. This can be used for load balancing and testing new versions of software.

> Internet Gateway

For internet connectivity to be established in an EC2 instance:

- The network ACL associated with the subnet must have rules to allow in/outbound traffic on port 80 and 443.
- The route table in the instance's subnet must have a route to an Internet Gateway
- The instance's subnet is always associated with a route table, and can only be associated with one route table at a time.

A VPC doesn't connect directly to an Internet gateway, it connects to a Nat Gateway instead for internet access.

> ACM, certificate manager

Used to configure SSL/TLS certificates on an ALB; use IAM when the region is not supported by ACM.

Security

- Key pairs: public and private keys, used to access Linux servers
  - Need root access for CloudFront key pairs
- Access keys: AWS equivalent of username+password, but for API calls.

To avoid data leaks and identify security weaknesses, try SQL injections, penetration tests (with AWS approval) and hardening tests. Code checks only check performance issues.

IAM

With policies, all requests are denied by default. An explicit allow overrides a default deny and an explicit deny overrides an explicit allow.

IAM variables: policy variables can be used to create a single policy that applies to multiple users, for example to allow each user in the group full programmatic access to a user-specific object (their own "home directory") in S3.

For Billing and Cost management console, users need to have IAM activated. When using this console, there's a paying account and a linked account.

> Cross-account access

Used to delegate access to resources that are in different AWS accounts by using roles, without needing to create individual IAM users in each account. For an AWS resource from account A to access resources of account B:

1. Establish trust between accounts A and B: create role in A and specify policy to access a certain resource.
2. Create role in B so that the resource from A can gain access to the cross-account resource
3. Modify the trust policy of the role in B to allow the role from A to assume this role
4. Update the resource from A to add the AssumeRole API call

> Policies

- Managed policy: default, AWS-managed. It can be assigned to multiple users, groups or roles and it is available for use by any AWS account. You can't change the default permissions defined in the policy
- Customer managed policy. It can be assigned to multiple users, groups or roles in your account
- Inline policy: managed by the customer and embedded in a single user, group or role. useful if you want to maintain a strict one-to-one relationship between a policy and the identity that it's applied to. The policy will be deleted if you delete the user, group or role it is associated with
- Resource based policy, trust policy: defines which entities can assume the role. An IAM role needs both a trust policy and an identity-based policy attached to it.

> IAM services

- To check and test a profile's permissions, use the CLI `--dry-run` option, which checks the permissions but doesn't make the request. Useful to test if a profile can do a certain action
- To check custom policies, test out the permissions by getting the context keys, and use the `aws iam simulate-custom-policy` command
- To check unused IAM roles, use Access advisor feature on IAM console
- *IAM access analyzer* lets you see AWS resources that are shared with an external entity, it lets you identify unintended access to your resources and data.

Cognito

Web identity federation (WIF) allows users to authenticate with a WI provider (WIP): Google, Facebook, Amazon. Cognito works with MFA authentication.

- Cognito **user pools**: sign-up and sign-in webpages for your app, access and manage user data. It can track user device, location, and IP address.
- Cognito **identity pools**: gives users access to AWS resources, like S3 or DynamoDB. Generates temporary credentials for unauthenticated, unique and trusted users
    - The user authenticates first with the Web ID provider and receives an authentication token, which is exchanged for tmp AWS credentials allowing them to assume an IAM role.
- Cognito **Sync**: cross device data sync without requiring your own backend
- Cognito **streams**: control and insight into the data stored in Cognito. You can configure a kinesis stream to receive events as data is updated and synchronized. Cognito can push each dataset change to a Kinesis stream in real-time

STS

Security token service: used to request *temporary* (expire after 1h), limited-privilege credentials for IAM users or for users you authenticate. Not supported by API gateway for authentication.

If an encoded authorization message is received when accessing an AWS resource, use STS decode-authorization-message

- `AssumeRoleWithWebIdentity` allows users who have authenticated with a WIP to access AWS resources. If successful, STS returns temporary credentials
- `AssumedRoleUser` ARN and `AssumedRoleID` are used to programatically retrieve the identifiers for the temporary security credentials.

Encryption

- **KMS** (key management service) is used to create and store encryption keys. KMS encryption keys are regional and can encrypt up to 4KB of data. For bigger keys, use envelope encryption.
- CMK (customer master key) is used for data keys
- Envelope encryption
    - First the data is encrypted using a plaintext Data key
    - This key is then further encrypted with a plaintext Master key
    - This way, only the data key goes over the network, not the data itself
    - This method requires code changes

Systems manager

SSM Parameter Store is storage for config data management and secrets management, to manage configs outside of EC2, or EBS. They can be loaded dynamically into the app at runtime

OpsWorks

OpsWorks is a config management service providing managed instances of Chef and Puppet, which are automation platforms allowing the use of code to automate the configs of servers.

Notifications

SQS

Highly-durable pull-based queue, useful for persist in-flight transactions. It has no strict ordering and might contain duplicates. To delete messages, a command is necessary, it's not automatic.

Visibility timeout: amount of time that the message is invisible in the SQS after a reader picks up the message. Makes sure that the message isn't read by any other consumer while it's being processed by one. If the job isn't processed in that time, the msg becomse visible again and another reader will process it. Default timeout is 30s, can be increased up to 12h.

- Message size limit is 256kb. For larger messages, store the messages in S3 and use Amazon SQS extended client library for Java to manage them, and also the AWS SDK for Java.
- No message limits for storing in SQS, but max 120k 'in-flight messages' (received from a queue by a consumer, but not yet deleted from queue)
- When retrieving messages from a queue, you can retrieve max 10 messages

- Retention period default is 4 days, but you can increase the queue message retention up to 14 days with Set QueueAttributes action

> Types of queues

- **FIFO queue**: strict ordering, one-time processing and no duplicates. But limit of 300 messages/second. To ensure messages arrive in order, use the sequence info in the messages with Standard queues
- **Delay queue**: postpone delivery of new messages when they're first added to the queue, default delay is 0s, max 900s. In this time, messages are invisible. For delaying individual messages rather than the entire queue, use message timers.
- **Dead letter queue**: to prevent data loss, they sideline, isolate and alayze the unsucessfully procesed messages. Useful when the lambda function invocation is asynchronous and it fails all retry attempts, in which case the message sends it to the DLQ.

If there are multiple senders, each sender's messages must be processed in order, by configuring each sender with a unique MessageGroupId, this is a flag that specifies that a message belongs to a specific message group. Messages that belong to a group are always processed one by one, in a strict order relative to the message group. For orders with different priorities, use 2 SQS queues

> Fetching from queue

- Short polling: returns a response immediately. Some messages might not get received, because short polling doesn't return all messages. It has additional cost
- Long polling: periodically poll the queue, response is returned only when a msg arrives or the long poll times out. Retrieves all messages. Can save money. To have the shortest delay, use `ReceiveMessageWaitTimeSeconds`. To reduce costs even more, group the SQS API operations in batches.

SNS

- Push-based asynchronous simple notification service with durable storage to send notifications from the cloud
- It can deliver push notifications, SMS and emails to any HTTP endpoint, and it can trigger a Lambda function. SES is not an endpoint. Can't receive anything
- Pub-sub model (publish and subscribe). Apps can push msgs to a topic, and subscribers receive these from the topic. Consumers must subscribe to a topic to receive the notifications
- Useful when sending a message and its metadata at the same time
- SNS can be used in conjunction with SQS to fan a single message out to multiple SQS queues

SES

Scalable and highly available email service. Pay as you go model, it send and receive emails. It can trigger SNS and Lambda. It can be used for automated emails, online purchases, marketing emails. SES is not a valid target for CloudWatch Events.

Kinesis

Family of services to analyze streaming data in real time.

> Shards

- Streams are made of shards, each shard is a sequence of 1+ data records and provides a fixed unit of capacity
- Default is 5reads/s, max 2MB. 1k writes/s, max is 1MB/s
- Records can be consumed according to a sequence number applied when data is written to the Kinesis shard
- For X shards, max X instances are allowed
- To scale up processing in your app, increase instance size, increase # instances to max # open shards, and increase # shards. The first two steps improve instances while shards run in parallel, the third increases the level of parallelism
- One worker can process multiple shards. Resharding, increasing the number of shards, doesn't mean you need more instances
- Order of data within a shard is guaranteed, but not across multiple shards

The partition key is used by KDS to distribute data across shards, and it's used to determine the shards to which a given data record belongs. If this key is not distributed enough, all data is getting sent to a few shards and not leveraging the entire cluster of shards. Shards can get hot or cold, if they're receiving much data or too little.

> Kinesis components

- **Data streams**: stream data/video
  - Strict ordering and duplicates, unlimited # consumers
  - Allows KMS encryption for data at rest, and encryption in flight with HTTPS endpoint
  - With enhanced fanout, multiple users can retrieve data from a stream in parallel. Stream consumers can be registered to receive their own 2MB/s pipe of read throughput per shard, with an average message propagation delay of 70ms for all consumers
- **Data firehose**: capture, transform, load streams into AWS data stores, when streamling directly into S3 for example. no analysis
  - To encrypt data, enable encryption on firehose and ensure that kinesis streams are used to transfer data from the producers
  - Firehose allows Elasticsearch, Redshift and S3 as sink types. Not ElastiCache, since it's not a storage type
- **Data analytics**: analyze, query and transform streamed data in real-time using standard SQL and save in an AWS data store

Storage and databases

S3

- Serverless storage service, best suited for objects and BLOB (binary large object) data
- Pricing is for every TB for month
- When using CloudFormation to delete buckets, make sure that all objects are deleted before deleting the bucket

> Uploading

- Max file size is 5GB. To upload larger files, use multi-part upload
- Object locking for concurrent updates not supported

- PUT is eventually consistent: when immediately listing the keys within a bucket after uploading a file, the object might not appear in the list
- DELETE is eventually consistent: when trying to read/list a recently deleted bucket, S3 might return the deleted data
- To improve performance when many files are uploaded/downloaded, add a hash prefix to the folder or if all files are in the same folder, add the key prefix to the object
- A sucessful upload results in a HTTP 200 result code and MD5 checksum

Downloading

If a bucket is private but object should be accessed, use a *pre-signed URL link* to access the file only. This access can have an expiration time.

To improve download performance, enable CloudFront to cache it. After, to minimize download speed, enable S3 transfer acceleration. For customizing content distributed via CloudFront, use Lambda@Edge. CloudFront can use HTTPS between clients and CF, and between CF and backend.

Objects

- *S3 versioning*: creates a new version whenever a file is overwritten or deleted
  - Protects from accidental deletes
  - Previous, unversioned files get the 'null' version
- *S3 Select*: SQL expressions to select a subset of the data
- *S3 inventory*: to audit and report on the replication & encryption status of the objects. Use this when getting an HTTP 503 error after many PUT operations
- *S3 analytics*: analyze storage access patterns, to change storage class for example

Security

Data can be secured with access control lists in file level (ACL) and bucket policies in bucket level. S3 access logs is security and access auditing of the requests that are made to the bucket.

S3 uses a log delivery account, Log Delivery group, to write access logs. This group will need group write permissions on the target bucket by adding a grant entry in the bucket's ACL. If the logging is redirected to the bucket itself, it will grow exponentially.

Encryption:

- Server side encryption (SSE), protects data at rest. For example, a bucket policy that denies S3 PUT requests that don't include the SSE parameter in the request header
  - S3 managed keys (SSE-S3): a unique key encrypts each object, and the key itself is encrypted with a master key that is regularly rotated. It uses AES-256 for encryption. The header for a request with this encryption is: `s3:x-amz-server-side-encryption": "AES256`
  - AWS KMS managed keys (SSE-KMS), KMS manages *data* key, the customer *master* key can be customer managed or created by KMS directly. `s3:x-amz-server-side-encryption": "aws-kms`
  - Server side encryption with customer provided keys (SSE-C): customer must manage the encryption key. No code necessary to encrypt or decrypt, only necessary to manage the encryption key you provide. In each API call, the customer needs to send the keys and encryption algorithm. For this situation, if the request is made over HTTP, S3 rejects it.

- Client side encryption: used for encryption in transit via SSL/TLS

Static web hosting

- When creating a static website hosting, you can configure index document, error document and conditional redirection on object name. It's not possible to configure conditional error on object name.

Cross-region

To replicate a bucket in another region:

1. Enable S3 bucket versioning
2. Enable cross-origin resource sharing (CORS) on bucket2: a bucket1 from region A can access files from bucket2
3. Bucket1 needs permission to replicate objects from bucket2

CORS allows cross-origin access to S3 resources. To do this, create a CROS configuration with an XML doc with rules that identify the origins that you allow to access the bucket, the operations (HTTP methods) that will support for each origin.

EBS

Highly available (automatically replicated within a single AZ but *AZ locked*) and scalable storage volume that can be attached to the EC2 instance, but can't share data between instances. Upon launch of an instance, at least one EBS volume is attached to it.

Snapshots are incremental and they span the region, which means they aren't in the same AZ as the volume.

Volume types

- **gp2**: general purpose SSD, boot disks and general applications. the only option that can be a boot volume. up to 16k IOPS per volume
- **io2**: provisioned IOPS SSD: higher IOPS, many read/writes per second. For large dbs, latency-sensitive workloads. highest performance option, most expensive
    - The maximum ratio of provisioned IOPS to the requested volume size (in GB) is 50:1. For example, a 100GB volume can be provisioned witih up to 5000 IOPS
- **st1**: throughput optimized HDD: for read-intensive workloads, for frequently accessed workloads that need to store mountains of data, big data, data warehouses
- **sc1**: cold HDD: lowest cost option, workloads where performance isn't a factor

Encrpytion

- EBS supports in-flight encryption and also at rest encryption using KMS
- To encrypt a volume: create an encrypted snapshot of the volume, restore the volume from the encrypted snapshot, mount it
- An encrypted EBS volume always creates an encrypted snapshot, which always creates an encrypted EBS volume
- Encryption by default is a region-specific setting. If enabled for a region, it can't be disabled for individual volumes or snapshots in that region

RDS

Highly available, fault tolereant SQL-type database solution. Primary host in one AZ handles all traffic, and replicates to the secondary host in another AZ in a *synchronous* manner: a write to the primary host is written to the secondary host. The secondary host doesn't handle traffic.

For backup that should be retained for long time, create a cron event in CloudWatch, which triggers a Lambda function that triggers the db snapshot.

To make all-or-nothing operations, use RDS MySQL to make both operations in a single transaction block.

RDS are monolithic, they have very coupled layers, if one is slow they are all slow. if one fails, they all fail. can't operate or scale independently.

> Read replica

- Used for read intensive databases
- It's a read-only db instance, updates to source are *asynchronously* replicated to read replica (eventual consistency)
    - Querying a read replica might return stale or old data
- Up to 5 read replicas can be created
- To access it, use the DNS endpoint

Aurora

Aurora is a self-healing cloud optimized relational db, distributed and elastic: it breaks apart the monolithic stack to enable a db that can scale out.

Aurora moves out the logging and storage layers of the db into a multi tented, scale out db optimized storage service. In Aurora global, the primary region allows read & write, while the secondary region is read only.

Aurora is not serverless by default, but can be made serverless.

DynamoDB

Non-relational serverless db stored on SSD, spread across 3 geographically distinct data centers, immediately consistent and highly durable. Can be used for web sessions, JSON docs, and metadata for S3 objects, scalable session handling.

> Consistency and throughput

Consistency only applies to read operations, which can be *query*, *scan*, *GetItem* or *BatchGetItem* for multiple items. These return all attributes by default: to get just some, use a projection expression.

1 RCU = 1 strongly consistent read/s for an item of 4kb in size. For larger files, more RCU units. Transactional reads' RCU consumption = double consistent reads = double eventual reads. WCU is 1 per 1KB/s. RCU and WCUs are specific to one table.

> How many RCUs does a file of 15KB need for 100 strongly consistent reads? 15 / 4 = 4, 100*4 = 400 RCUs. For eventual consistency, 200 RCUs.

For the highest throughput questions, choose eventual consistency, maximum reads capacity * max item size. For high latency issues, use eventually consistent reads instead of strongly consistent reads.

Security

- By default, DynamoDB tables are encrypted at rest with an AWS owned key
- Access is controlled using IAM policies: conditions can be specified when granting permissions: read-only access to certain items/attributes, or write-only based upon the identity of the user
    - With `dynamodb:LeadingKeys`, users can only access the items where the partition key value matches their user ID

Keys and indexs

An item can have more than one attributes. The primary key can be single key (partition key) or composite key (partition key + sort key). The partition key and sort key can be different from the base table.

A local secondary index can only be created at the time of table creation, but a global secondary index can be created at at any time. Local indexes are immediately consistent but once you create them, all records sharing the same partition key need to fit in 10GB. Global indexes don't constrain your table size in any way, but reading from them is eventually consistent.

LSIs and GSIs can coexist. LSIs use the RCU and WCU of the main table, which can't be changed. GSIs are stored in their own partition space away from the base table, and scale separately from the base table.

Queries

Query results are more efficient than a scan, and they're always sorted in ascending order by the sort key if there is one. If querying in an attribute *not* part of partition/sort key, querying that is the same as scanning. Parallel scans while limiting the rate minimize the execution time of a table lookup.

The number of tables per account and number of provisioned throughput units per account can be changed by raising a request. Use global tables if the app is accessed by globally distributed users.

A conditional action is an action only to be taken if certain attributes of the item still have the values you expect. If requests are being throttled because of exceeded throughput, use exponential backoff in the requests.

When read and write are distributed unevenly, a "hot" partition can receive a higher volume of read and write traffic compared to other partitions. DynamoDB adaptive capacity enables the app to continue reading/writing to hot partitions without being throttled, provided that traffic doesn't exceed the table's or partition's max capacity.

- **DynamoDB TTL** is a time to leave attribute to *remove* data after certain time
- **DynamoDB streams** is a time-ordered sequence of item level modifications in the DynamoDB tables, can be used as an event source for Lambda to take action based on events in the table. You can use it when a record is inserted to table2 whenever items are updated in table1
- **DynamoDB transactions**: to do coordinated, all-or-nothing changes to multiple items within and across tables.

Caching

DynamoDB accelerator (DAX) is a fully managed, clustered in-memory cache for DynamoDB. Only use it for eventually consistent reads. API calls are pointed to the DAX cluster instead of to the table, and it passes all requests to DynamoDB and doesn't cache for these requests.

> Backup

On-demand and point-in-time recovery, which provides continuous backups of the table data. When enabled, DynamoDB maintains incremental backups of your table for the last 35 days until you explicitly turn it off.

Elasticache

In-memory cache in the cloud, used to improve latency and throughput for many read-heavy or *compute-intensive* workloads, or to store session states. If the db is running many OLAP transactions, use Redshift. Redshift is not suitable for workloads that need to capture data, but it supports join operations.

> Types of Elasticache

- *Memcached*: *simple* object caching system to offload a db, supports multithreaded performance using multiple cores. No multi-AZ capability, could have downtime
- *Redis*: more complete, in-memory key-value store used for high availability, multi-AZ redundancy, but it's not highly durable. AWS treats this more like a RDS, with sorting and ranking datasets in memory. Redis cluster provide high availability and durability

> Strategies for caching

- **Lazy loading**: loads data into the cache only when data is requested. Good option when there's not much space. It can have cache miss penalty, and the data can get stale. If the data in the db changes, the cache doesn't automatically get updated
- **Write through**: adds/updates data to the cache whenever data is written to the db. Data in the cache is never stale, but write penalty: every write involves a write to the cache and resources are wasted if the data is never read. Elasticache node failure means that data is missing until added or updated in the database

For backend caching, similar to write-through: write to backend, and then invalidate the cache. Then the caching engine fetches the latest value from the backend, so the backend and cache are in sync always.

CloudFormation

AWS resources defined in a YAML script. CF should be used for VP configs, security groups, LBs, deployment pipelines, IAM roles. Not to be used for DynamoDB tables, Kinesis streams, AutoScaling settings or S3 buckets. AWS SAM and Elastic Beanstalk rely on CF to provision resources.

- **Transforms** required field. if this is used, it means the document is a SAM template
- **Resources** required field. defines the resources and their properties
    - Can reference a nested stack, which must be previously saved in S3
    - `!GetAtt` returns the value of an attribute from a resource in the template, can't receive inputs
- **Conditions** section includes conditions that control whether certain resource properties are assigned a value during stack creation or update
- **Parameters** to *pass values* such as passwords to the template at runtime. Does not allow conditions. Allows the data types of string, number and list.
- **Mappings** of keys and values that can specify conditional parameter values
    - `!FindInMap` returns the value corresponding to keys in a two-level map that is declared in this section

- **Outputs** declares output values that you can import into other stacks, return in response or view on CF console. Use Export field for this
    - With a cross-stack reference, owners of the web app stacks don't need to create or maintain networking rules or assets

Intrinsic functions in templates are used to assign values to properties that aren't available until runtime. `Ref` returns the value of the specified parameter or resource, but cannot import values. It can accept input value from the user.

If part of the CF deployment fails due to a misconfiguration, CF rollbacks the entire stack. If one of the resources in a stack can't be created, previously created resources get deleted and the stack creation terminates. Termination Protection stack option prevents accidental deletion of an entire CloudFormation stack.

StackSets extend the functionality of stacks by enabling you to create, update or delete stacks across many accounts and regions.

If Stack B and Stack C depend on Stack A, stack A must be deleted last. All imports must be removed before deleting the exporting stack. You can't delete Stack A first because it's being referenced in the other Stacks.

To declare a Lambda function in CF, either upload the code as a zip to S3 and refer the object in AWS::Lambda::Function, or if the code is small and has no third-party dependencies, write the code inline in CF in the AWS::Lambda::Function block.

CF change sets feature: helps check how changes to a CF template affect AWS resources before implementing the updated

> CLI commands:

- `cloudformation package`: package the local artifacts that the CF template references. Uploads the local artifacts, like source code to Lambda
- `cloudformation deploy`: deploys the specified CF template

> Serverless

SAM is the serverless application model, to define and provision serverless apps using CloudFormation. To deploy using SAM CLI, develop+test the app locally, use `sam deploy`. This zips the app, uploads them to S3 and deploys the app to AWS.

- AWS::Serverless::Application: to embed nested apps from S3 buckets
- AWS::Serverless::API: to create API gateway resources and methods that can be invoked through HTTPS endpoints
- AWS::Serverless::Function: configuration to create a Lambda function
- AWS::Serverless::LayerVersion: to create Lambda layered function

Elastic beanstalk

Deploy and scale web apps, without provisioning underlying servers, LBs, security groups etc.. It supports the deployment of web apps from Docker. If the on-premise application doesn't use Docker and can't seem to find a relevant environment in Beanstalk, use Packer to create a custom platform.

Launch

- User can customize an AMI instead of the standard one. It can make launch faster if dependencies are preloaded and prepackaged in the AMI
    - If a lot of server configs is needed, just make a custom AMI instead of configuring and customizing environment
- If Beanstalk performs tasks that take a long time: offload the tasks to a dedicated worker environment
- Traffic splitting / canary testing: useful if you want to test the health of your new application version using a portion of incoming traffic, while keeping the rest of the traffic served by the old application version

Config

- `.ebextensions/`, environment customization with configuration files
    - Files must be YAML/JSON and have `.config` extension
    - Define Load Balancers, Elasticache
    - Docker containers are supported
    - If the env must include custom software, create YAML with the required package names
    - For RDS, reference externally and load with env variables

Other configurations

- For cron jobs, set up a worker environment and include `cron.yaml`
- Access logs without logging into the app servers by enabling log file rotation to S3 within the ELB configuration
- To export a Beanstalk configuration to another account, create a saved configuration from acc1, download it to local. In acc2, upload it to S3 bucket and from Beanstalk console create the app from 'saved configurations'.

Whenever a new version is uploaded to Beanstalk, it creates an app version. If the older ones aren't deleted, eventually the app version limit is reached. This can be avoided by applying an application version lifecycle to the apps. This tells Beanstalk to delete old app versions or to delete app versions when the total # versions for an app exceeds a number

Deployment

Deployment type	Speed	Downtime	Availability reduction	Deploy to	Rollback	Cost
All at once	Fastest	Yes	Yes	Existing instances	Manual	Low
Rolling	Fast	No	Yes	Existing instances	Manual	Low
Rolling with additional batch	Slow	No	No	Existing instances	Manual	Higher
Blue/green: new env	Slower	No	No	New instances	Automatic, point LB to older versions	Highest

Deployment type	Speed	Downtime	Availability reduction	Deploy to	Rollback	Cost
Immutable: same env	Slower	No	No	New instances	Automatic, terminate new instances	Highest

- Blue/green: new environment, new LB but using the same autoscaling group (or weighted routing policy in Route 53). The switch is performed at DNS level routing the traffic from the old env to the new when the new env is ready and healthy. Instant complete switch from old to new version
- Immutable: same environment, same LB but new autoscaling group. From the moment a new instance is created, it serves traffic. When the new instances are all healthy, switch off the old ones.

Amazon API Gateway

It's a service to manage APIs at any scale. Users make a request to the API gateway, and this redirects the request to EC2, Lambda, etc. First of all a deployment must be created in API Gateway. When changing the API, redeploy it to an existing stage or to a new stage.

- API caching can be enabled for popular requests, it caches responses from the endpoint for a specified TTL period. Default is 300s, max can be 3600. If it's 0, caching is disabled
    - To invalidate caching, send header with Cache-Control: max-age=0
- Frontend: method, backend: integration
- Docker is not supported
- If a request is comming like an XML, the request and response data mapping template will map it to JSON

```
# passing a stage variable to an HTTL URL
http://${stageVariables.<variable_name>}.example.com/dev/operation
http://example.com/${stageVariables.<variable_name>}/prod
```

> Security

- For access from different domains, use CORS: Access-Control-Allow-Methods, Access-Control-Allow-Headers, Access-Control-Allow-Origin. It can also be used to deny cross-domain access.
- To deny specific IP addresses from accessing API Gateway, use WAF or resource policies
- Lambda authorizer is a Lambda function controlling access to the API
- API gateway can be throttled to prevent attacks

Developer tools

CodeStar handles all aspects of development and deployment on AWS.

CodePipeline

- Manages the whole workflow whenever a change is detected in the source code
- If the pipeline needs needs approval, add a manual approval step at the end of the flow
- If one stage of the pipeline fails, the entire process stops running

CodeCommit

- Source and version control
- Allows access through git credentials, SSH keys and AWS access keys, not through IAM username and password. Create Git credentials for IAM users and allow the devs to connect via HTTPS using these credentials
- Repositories are automatically encrypted at rest

CodeBuild

Automated build, runs tests, produce packages. The `buildspec.yml` file specifies the build configurations, should be in the root folder.

- It scales automatically to meet peak build requests
- If the build fails, run it locally using CodeBuild Agent
- For very big dependencies, bundle them all in the source coude during the last stage of CodeBuild, thus reducing the build time
- Logs are in CloudWatch, they can be exported to S3

To override build commands without touching the code or editing the project, run the start build CLI command with buildspecOverride property to set the new buildspec.yml file.

CodeDeploy

Automated deployments to EC2, Lambda, or to on-premises. With a CodeDeploy agent in an EC2 instance, the instances can be used in Codedeploy deployments. The agent cleans up log files to conserve disk space. With Codedeploy deployment groups, EC2 instances are a set of individual instances targeted for deployment.

> Steps: ApplicationStop --> BeforeInstall --> DownloadBundle --> AfterInstall -> ApplicationStart --> ValidateService (to verify success)

- `appspec.yml` is the deployment config file, it must be in the root folder
    - When working with Lambda, specify the version in appspec.yml
    - the hooks section determines the scripts that are run in the lifecycle event hooks

> Rollback

If the deployment fails, CodeDeploy first deploys to the failed instances. A new deployment of the last known working version is deployed with a new deployment ID.

If the previous version's files are unreachable, they have to be manually added to the instance, or a new app revision must be created.