Domains:

1. Development with AWS Services
   Serverless, API, SDK, CLI
2. Deployment
   CICD, Beanstalk, Serverless
3. Security
   IAM, KMS, each service access policies and encryption methods
4. Monitoring and troubleshooting
   CloudWatch, CloudTrail, X-Ray
5. Refactoring
   AWS Service for best migration

IAM and EC2:

EC2 is region specific service, while IAM is a global service

IAM: Identity and Access Management
AWS Security: Users, Groups, Roles
Root account should never be used
All AWS services are connected to IAM for security
Policies are written in JSON

IAM controls: Who (authentication) can do What (authorization) in your AWS account.
Authentication(who) with IAM is done with users/groups and roles whereas authorization(what) is done by policies.
User - End user, think about people
Groups - a set of users under one set of permission(policies)
Roles - are used to grant specific permission to specific actors for a set of duration of time. These actors can be authenticated by AWS or some trusted external system.

Can have a password? user->Yes , role->No
Can have an access key? user->Yes , role->No
Can belong to a group? user->Yes , role->No
Can be associated with AWS resources (for example EC2 instances)? user->No , role->yes

AWS supports 3 Role Types for different scenarios:
AWS service roles (for example: EC2, Lambda, Redshift,...)
Cross-Account Access: granting permissions to users from other AWS account, whether you control those accounts or not.

Identity Provider Access: granting permissions to users authenticated by a trusted external system. AWS supports two kinds of identity federation: - Web-based identity such as Facebook, Goolge- IAM support integration via OpenID Connect and SAML 2.0 identity such as Active Directory, LDAP.

To use/integrate your company credentials, use IAM Federation. It uses SAML standard

IAM best practices:
One IAM user per one person. IAM user should not be assigned to multiple people
One IAM role per application. IAM role should not be assigned to multiple applications
IAM credentials should not be in the code
Don't use root account except for initial setup

IAM policy: service(AWS), actions(functions), resources(specific resource in that service or all the resources in that service for the account), request conditions
can be assigned to user, group, or role

Use the Principal element in a policy to specify the principal that is allowed or denied access to a resource.
It must not be included in identity-based policies, the principal is implicitly the user that the policy is attached to (for IAM users) or the user who assumes the role (for role access policies)
The principal_block element is required in resource-based policies (for example, in Amazon S3 bucket policies) and in trust policies for IAM roles.
Resource-based policies are policies that you embed directly in an AWS resource. For example, you can embed policies in an Amazon S3 bucket or an AWS KMS customer master key (CMK).
In resource-based policies, use the Principal element to specify the accounts or users who are allowed to access the resource
In IAM roles, use the Principal element in the role's trust policy to specify who can assume the role. For cross-account access, you must specify the 12-digit identifier of the trusted account.

to attach the policy, use the attach-user-policy command, and reference the environment variable that holds the policy ARN.

```
$ aws iam attach-user-policy --user-name MyUser --policy-arn $POLICYARN
```

Verify that the policy is attached to the user by running the list-attached-user-policies command.

```
$ aws iam list-attached-user-policies --user-name MyUser
```

enable your applications to use temporary security credentials provided by AWS by attaching an IAM role to an existing EC2 instance
attach an IAM role to an existing EC2 instance
Call the associate-iam-instance-profile command to attach the instance profile, YourNewRole-Instance-Profile, for the newly created IAM role, YourNewRole, to your EC2 instance, YourInstanceId

```
$aws ec2 associate-iam-instance-profile --instance-id YourInstanceId --iam-instance-profile
    Name=YourNewRole-Instance-Profile
```

Renting virtual machines - EC2
Storing data on virtual drives - EBS
Distributing load across machines - ELB
Scaling the services using an auto-scaling group - ASG

Create/Launch EC2 instance
- Choose Machine Image (OS)
- Choose Instance type (CPU, RAM, etc)
- Configure Instance (VPC/Network, IAM role, File system, User data)
- Storage (Size, SSD)
- Tags (key:value pair)
- Configure Security Group (Firewall around your instance, enable SSH on port 22)

generate public private key pair, and download it

custom AMI are specific to a region

when trying to ssh into the instance we get permission denied. Permission 0644 for "private key file" is too open.

```
chmod 0400 <file name>
ssh -i <key file name> user@ip_address
```

connection timeout issue
This is a security group issue. Any timeout (not just for SSH) is related to security groups or a firewall.
connection refused
This means the instance is reachable, but no SSH utility is running on the instance
permission denied (publickey,gssapi-keyex,gssapi-with-mic)
This means either two things: You are using the wrong security key or not using a security key, or You are using the wrong user

public IP of your EC2 instance will change on instance restart

ec2 connect -> aws service to ssh into your instance from browser

Security Groups: network security in AWS
control how traffic is allowed in and out of EC2 machine
access to ports, authorised ip range, control of inbound and outbound traffic
they are specific to region/vpc combination
all inbound traffic is blocked by default
all outbound traffic is authorised by default
one security group can be connected to another security group - usually for communication between multiple ec2 instances

if you need a fixed public ip for your instance -> Elastic IP. can only have 5 elastic ip in your account
but best practice - use random public ip and register dns name to it or use a load balancer

By default, ec2 comes with a private ip (for internal AWS network) and a public IP (www, which we use to ssh)

EC2 user data - run the commands when the machine starts (ex - install updates, required softwares)
for example, every new instance we create should have apache server installed on it

aws ec2 instance metadata <- command to learn about themselves
http://169.254.169.254/latest/meta-data
internal ip, will only run in the ec2 instance
can retrieve IAM role but not the IAM Policy(to test IAM policy use dry run or policy simulator)

EC2 Instance types:
On demand - short workload, predictable pricing
Reserved (min 1 year) - Reserved instance(long workload), Convertible Reserved(long workload with flexible instance), Scheduled Reserved(every Thursday between a particular time period)
Spot instance - short workload, cheap, less reliable
Dedicated - entire physical server

Elastic Network Interfaces (ENI) -> logical component of VPC that represents virtual network card
ENI can have: primary private ipv4, 1 eip per private ipv4, 1 public ipv4, 1 or more security groups, a mac address
can be created independently and moved from the ec2 instance in case of failover
bound to AZ

ELB and ASG:

Scalability - system can handle greater loads by adapting
High availability - fault tolerant

Elastic Load Balancer (ELB) -> servers that forward internet traffic to multiple servers downstream
expose a single point of access (DNS) to your application
regular health checks of the instances
handles failures of downstream instances
can provide ssl termination for the website
can enforce stickiness with cookies
provides high availability across zones
separates public traffic from private

the health check is done on a port and a route, if the response is not 200 then instance is unhealthy

3 kinds of load balancers in aws:
Classic load balancer (old) - HTTP, HTTPS, TCP
Application load balancer - HTTP, HTTPS, WebSocket
Network load balancer - TCP, UDP, TLS

ELBs can be internal(private) or external(public)

Change the security group of the ec2 instance so that http communication is restricted with the elb, whereas elb's security group can communicate with any user

4xx errors are client induced errors
5xx are application induced errors
load balancer error 503 means at capacity or no registered target
if elb can't connect to app then check security groups

ELB access logs will log all access requests (you can debug per request)
CloudWatch Metrics will give you aggregate stats (ex - connections count)

Application load balancer - level 7
load balancing to multiple applications across machines (target groups)
load balancing to multiple applications on the same machine (containers)
supports websocket
supports redirect
supports routing - based on path in url, based on hostname in url, based on query string, header
port mapping to redirect to a dynamic port in ECS
good for microservices and container based application

Target groups could be -
EC2 instances managed by ASG (HTTP)
EC2 Tasks managed by ECS (HTTP)
Lambda functions - HTTP request is translated into a JSON event
IP addresses to private IPs

we get fixed hostname with ALB
we don't see the client IP directly, IP of client in header X-Forwaded-For, port in X-Forwaded-Port, proto in X-Forwaded-Proto

Network Load Balancer - level 4
forward tcp and udp traffic to your instances
handles millions of requests per second, less latency 100ms (vs 400ms of alb)
has one static ip per AZ, supports assigning eip, exposes ip instead of dns
used for extreme performance, tcp or udp traffic

ELB - stickiness
same client redirected to same instance behind a load balancer
works for CLB and ALB
the cookie used for stickiness has an expiration date you control
use case: make sure the user doesn't lose his session data
enabling it may bring imbalance

when cross zone load balancing used, each lb distributes load across all registered instances in all AZ

SSL/TLS - allows traffic between client and load balancer to be encrypted in transit (in-flight encryption)
Secure Sockets Layer (SSL) to encrypt connections
Transport Layer Security (TLS), newer version
load balancer does ssl termination
SNI - server name indication, handle multiple certificates with your load balancer

ELB - connection draining (CLB), Deregistration delay (target group, alb and nlb)
time to complete "in-flight requests" while the instance is de-registering or unhealthy

ASG - Auto scaling group
scale out to match increased load
scale in to match decreased load
automatically register new instances to load balancer

ASG have
launch configuration: AMI + instance type, EC2 user data, EBS Volume, Security groups, SSH key pair
min size, max size, initial capacity
network + subnet info
load balancer/target group info
scaling policies

possible to scale ASG based on CloudWatch alarms
auto scaling rules based on - target avg cpu usage, no. of requests per instance, avg network in, avg network out

ASG - scaling policies
Target tracking scaling - avg. ASG CPU at 50%
Simple/Step scaling - CloudWatch alarm cpu > 70%, then add 2 units, CloudWatch alarm cpu < 30%, then remove 1 units
Scheduled Actions - anticipate based on patterns

Scaling cooldown - asg doesn't take any action before previous scaling activity takes effect

EBS and EFS:

Elastic Block Storage Volume - network drive attached to instances
EC2 loses its main drive when terminated, store data somewhere else

latency, can be detached from one instance and attached to another instance
locked to an AZ, can be moved across AZs using snapshot
has a provisioned capacity, size in GB and iops

4 types:
GP2 (SSD): General purpose volume
IOI (SSD): High performance, low latency - large database workloads
STI (HDD): Low cost, throughput intensive workloads - log processing, kafka
SCI (HDD): Lowest cost, less frequently accessed
gp2 and ioi can be used as boot volumes
iops: io operation per second

newly created ebs volume needs to be mounted to the ec2 instance, after formatting the volume with any file system
billed for provisioned

instance store (ephemeral) is physically attached to the machine, ebs is a network drive

Elastic File System - can be mounted on many EC2 instances in multi-AZ
content management, web serving, data sharing
only for linux instances, posix
billed for usage

instance store locked to ec2 instance
EBS locked to an AZ
EFS locked to multi-AZs (Region?)

to migrate EBS volume, take snapshot and then restore it in another AZ

RDS, Aurora and ElastiCache:

Relational Database Service: managed db service for dbs using sql
automated backups, transaction logs, snapshots

read replica: up to 5 read replicas possible
could be with in AZ, Cross AZ, or cross region
reads are eventually consistent

can be promoted to their own db

network cost when data goes from one AZ to another

multi AZ (disaster recovery): sync replication
one dns name, automatic app failover(AZ, network, db, instance) to standby

the read replicas can be setup as multi AZ for disaster recovery

at rest encryption, if master is not encrypted than read replicas can't be encrypted
in flight encryption, ssl. enforce ssl from database

deployed in private subnet, security groups
iam policies to manage RDS
iam based authentication to log into rds, auth token

Aurora - proprietary aws technology
storage automatically grows in increments of 10 GB, upto 64 TB
15 read replicas
6 copies of your data across 3 AZs

aurora serverless - no capacity planning needed
good for infrequent, intermittent or unpredictable workloads
db instantiation and auto scaling based on usage

global aurora:
Aurora Cross region read replicas:
  useful for disaster recovery
  simple to put in place
Aurora global database:
  1 primary region
  5 read only regions

ElastiCache: Managed redis or memcached
in-memory db with high performance and low latency
write scaling using sharding, read scaling using read replicas
multi AZ with failover

can be used to store user session data

Redis: multi AZ with auto-failover
read replicas to scale reads and have high availability
data durability using AOF persistence(logs every write operation received by server)

backup and restore features

Memcached: multi-node for partitioning of data (sharding)
non persistent
no backup, restore
multi threaded architecture

Caching policy:
Lazy loading/Cache-aside/Lazy population : update the cache using missed data on cache miss
Write through : add or update cache when database is updated
Cache evictions and TTL: explicit delete or mem is full and cache is not used (lru) or item TTL expires

Route53:

managed DNS
common types of records:

- A: hostname to ipv4
- AAAA: hostname to ipv6
- CNAME: hostname to hostname
- Alias: hostname to aws resource

public domain names and private domain names

load balancing (DNS - client load balancing)
Health checks (limited func.)
routing policies
pay per month per hosted zone

TTL

if root domain than use alias, for non root domain use alias or cname

simple routing policy: redirect to a single resource, in case of multiple returns client will randomly pick one
weighted routing: control the % of requests that go to specific endpoint
latency routing: least latency from user to designated AWS region
failover routing: use secondary if health check on primary fails
geolocation: route based on user location
multi value routing: return healthy records

VPC:

virtual private cloud - private network to deploy your resources
Vpcs are region specific, 5 vpc per region
subnets - partition network inside VPC, locked to AZ, 200 subnets per vpc
Region comes with default vpc
public subnet accessible from internet, private not accessible from internet
to define access to internet, and between subnets, use route tables
Each subnet in vpc must be associated with a route table

route table contains a set of rules (routes) that determine where network traffic is directed.
Each subnet in your VPC must be associated with a route table which controls the routing for subnet.
A subnet can only be associated with one route table at a time, but you can associate multiple subnets with the same route table.

internet gateways helps our vpc instances connect with the internet
public subnets have a route to internet gateway
Provides a target in the vpc route table for internet-routable traffic
Preform network address translation (NAT) for instances that have been assigned ipv4

NAT gateways (AWS managed) and NAT instances (self-managed)
allow your instances in your Private Subnets to access internet while remaining private

Network ACL - firewall which controls traffic from and to subnet
allow and deny rules, ip address at subnet level
default everything allowed in and out

Security Group - firewall that controls traffic to and from an ENI/an EC2 instance
allow rules, ip addresses or other security groups

VPC flow logs: capture info about traffic going into your interfaces, VPC, Subnet, ENI (elastic network interface) flow logs

VPC peering: connect two VPC, privately using AWS network, must not have overlapping cidr block (IP address range), not transitive

VPC endpoints: connect to AWS services using private network instead of www, vpc endpoint gateway (s3, dynamoDB), vpc endpoint interface (other services)

Site to site vpn: connect on prem VPN to AWS over internet
Direct connect: physical connection

S3:

objects - files, buckets - directories
buckets must have globally unique name
S3 global service but buckets are region level

objects(files) have a key (Full path), prefix + object name
max object size is 5 TB, to upload more than 5 GB use multi-part upload

metadata
tags

Versioning: enabled at bucket level, version your files in S3
same key overwrite will increase the version
versions of a file: queue of versions but emulates a stack with head being the latest/current version

pre signed url: can be generated using CLI(download) or SDK(upload)
valid for default 3600 seconds
generate URL dynamically to download file instead of giving user the access to the bucket or give temporary access
cli commands:

```
aws configure set default.s3.signature_version s3v4
aws s3 presign s3://<object-path> --region <location>
```

Encryption: encrypt objects, 4 methods, SSE (server side encryption)
SSE-S3 : encrypt s3 objects using keys handled and managed by AWS, set header "x-amz-server-side-encryption":"AES256"
SSE-KMS : use AWS-KMS to manage encryption keys, user control and audit trail, "x-amz-server-side-encryption":"aws:kms"
SSE-C : manage your own encryption keys, https, encryption key in every http header, aws will discard the key, only thru cli
Client side encryption, encrypt data before sending, Amazon S3 encryption client
encryption in transit - ssl/tls, amazon s3 exposes https endpoint (mandatory for SSE-C), http endpoint not encrypted

S3 put api headers:
Content-MD5 - This header can be used as a message integrity check to verify that the data is the same data that was originally sent.
X-amz-server-side-encryption - The server-side encryption algorithm used when storing this object in Amazon S3 (for example, AES256, aws:kms).
X-amz-server-side-encryption-aws-kms-key-id - If the value of x-amz-server-side-encryption is aws:kms, this header specifies the ID of the symmetric customer managed AWS KMS CMK that will be

used for the object. If you specify x-amz-server-side-encryption:aws:kms, but do not provide x-amz-server-side-encryption-aws-kms-key-id, Amazon S3 uses the AWS managed CMK in AWS to protect the data.

X-amz-server-side-encryption-customer-key - Specifies the customer-provided encryption key for Amazon S3 to use in encrypting data. This value is used to store the object and then it is discarded; Amazon S3 does not store the encryption key. The key must be appropriate for use with the algorithm specified in the x-amz-server-side-encryption-customer-algorithm header.

x-amz-server-side-encryption-customer-key-MD5 - Specifies the 128-bit MD5 digest of the encryption key according to RFC 1321. Amazon S3 uses this header for a message integrity check to ensure that the encryption key was transmitted without error.

X-amz-storage-class - If you don't specify, S3 Standard is the default storage class. Amazon S3 supports other storage classes.

User based S3 security: IAM policies - which api calls should be allowed from which user

Resource based S3 security: bucket policies, Object Access Control List (ACL), Bucket Access Control List (ACL)

Bucket ACLs allow you to control access at a bucket level, while Object ACLs allow you to control access at the object level.

json based bucket policies

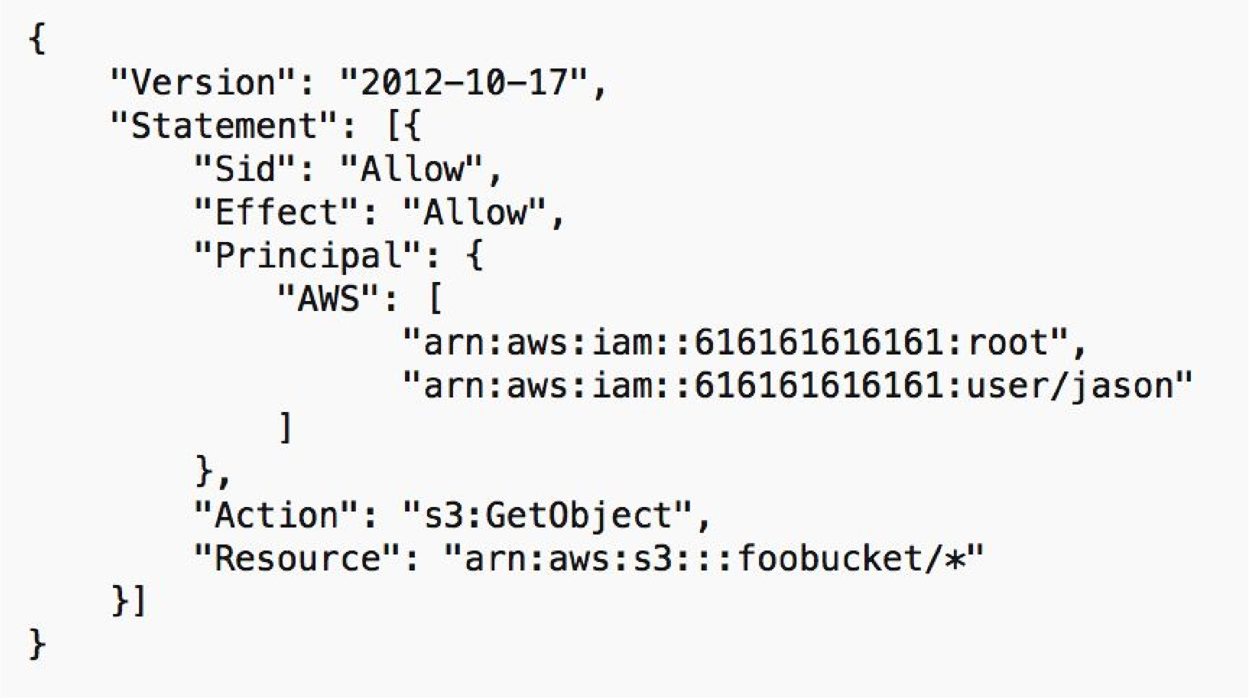
Resources: buckets and objects

Actions: set of API calls to be Allow or Deny

Effect: Allow or Deny

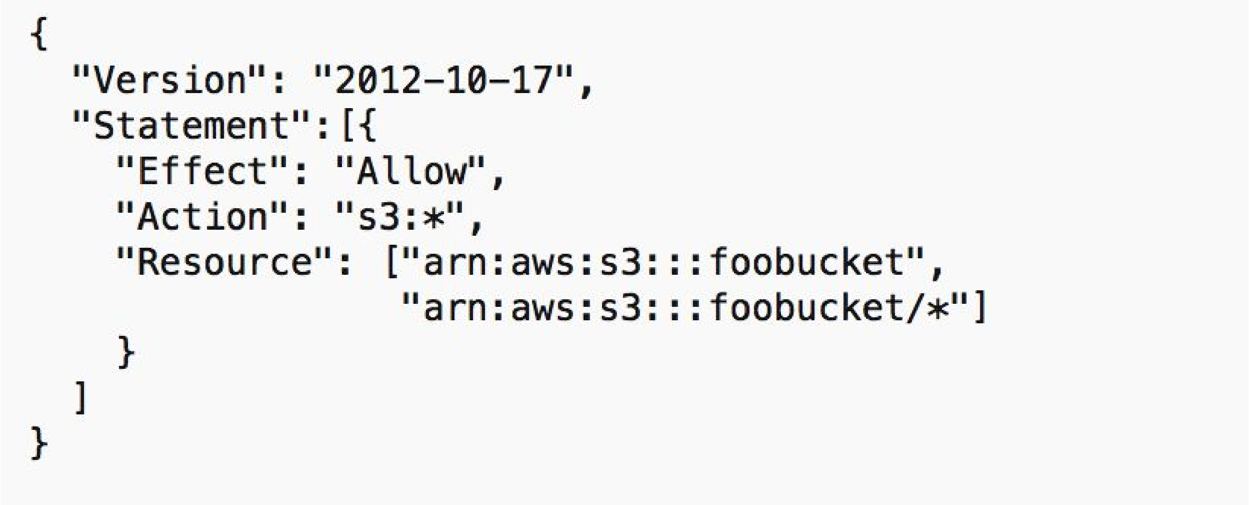
Principal: the account/user to apply the policy to

Bucket policies allow users to easily grant cross-account access without having to create roles using the “Principal” IAM element. To see this in action, let's assume we want to allow s3:GetObject on foobucket to the root account 6161616161 as well as the user jason under that account. This could be implemented with the bucket policy below:



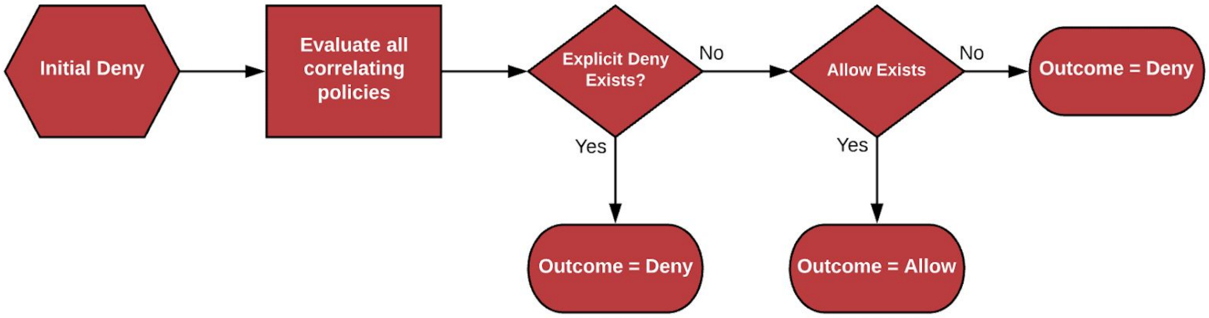
```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "Allow",
        "Effect": "Allow",
        "Principal": {
            "AWS": [
                  "arn:aws:iam::616161616161:root",
                  "arn:aws:iam::616161616161:user/jason"
            ]
        },
        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::foobucket/*"
    }]
}
```

the user based policy below allows whatever entity that has the policy attached to perform any action on foobucket:



```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect": "Allow",
    "Action": "s3:*",
    "Resource": ["arn:aws:s3:::foobucket",
                 "arn:aws:s3:::foobucket/*"]
    }
  ]
}
```

access will be determined by combination of all policies:



Use IAM policies if:

- You need to control access to AWS services other than S3. IAM policies will be easier to manage since you can centrally manage all of your permissions in IAM, instead of spreading them between IAM and S3.
- You have numerous S3 buckets each with different permissions requirements. IAM policies will be easier to manage since you don't have to define a large number of S3 bucket policies and can instead rely on fewer, more detailed IAM policies.
- You prefer to keep access control policies in the IAM environment.

Use S3 bucket policies if:

- You want a simple way to grant cross-account access to your S3 environment, without using IAM roles.
- Your IAM policies bump up against the size limit (up to 2 kb for users, 5 kb for groups, and 10 kb for roles). S3 supports bucket policies of up 20 kb.
- You prefer to keep access control policies in the S3 environment.

put-bucket-policy applies an Amazon S3 bucket policy to an Amazon S3 bucket.

aws s3api put-bucket-policy --bucket MyBucket --policy file://policy.json

If you are using an identity other than the root user of the AWS account that owns the bucket, the calling identity must have the PutBucketPolicy permissions on the specified bucket and belong to the bucket owner's account in order to use this operation.

If you don't have PutBucketPolicy permissions, Amazon S3 returns a 403 Access Denied error. If you have the correct permissions, but you're not using an identity that belongs to the bucket owner's account, Amazon S3 returns a 405 Method Not Allowed error.

grant public access to the bucket, force objects to be encrypted at upload, grant access to another account

- bucket setting to block public access, when you know bucket should not be public
- bucket policies are evaluated before default encryption
- S3 network security - vpc endpoints (for instances in vpc without internet)
- Logging and audit - S3 access logs can be stored in other s3 bucket, API calls can be logged in AWS CloudTrail
- User security - MFA Delete, pre signed urls valid for a limited time

S3 websites - host static websites, accessible from internet, 403 error if bucket policy denies access

path = bucket/prefix/number1/file01, bucket + prefix + file

Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to tell browsers to give a web application running at one origin, access to selected resources from a different origin. A web application executes a cross-origin HTTP request when it requests a resource that has a different origin (domain, protocol, or port) from its own.

Cross Origin Resource Sharing (CORS):

- origin: scheme (protocol), host (domain), port
- get resources from different origin
- browser based mechanism to allow requests to other origins while visiting the main origin
- request won't be fulfilled unless the other origin allows, using CORS header

Access-Control-Allow-Origin, Access-Control-Allow-Methods

- preflight request/response
- for client to do cross-origin request on S3, we need to enable the headers
- can allow specific origin or *
- s3 bucket -> permissions -> CORS configuration -> paste CORSRule

S3 consistency: eventually consistent
read after write consistency for PUTS of NEW objects, except if you had done get before put
eventual consistency for DELETE and PUT of EXISTING objects

-> S3 and Athena

MFA-Delete, first enable versioning in S3 bucket, than permanently deleting object version and suspending versioning on the bucket will require MFA
only root account/bucket owner can enable/disable MFA-Delete, using CLI

S3 access logs: all requests logged into another s3, can be analysed using athena

S3 replication: cross region replication(crr), Same region replication(srr)
versioning should be enabled in source and destination, buckets can be in different account, asynchronous replication, proper IAM permissions
only new objects after enabling will be replicated, delete not replicated, no chaining of replication

CRR: compliance, lower latency access, replication across accounts
SRR: log aggregation, live replication b/w prod and test

S3 storage classes of object:
s3 standard - general purpose : high durability, across multi AZ, for big data analytics, mobile and gaming applications, content distribution
s3 standard infrequent access (IA): for less frequent but rapid accessed, disaster recovery, backups

s3 one zone infrequent access: same as IA but in single zone, 20% less cost, secondary data or data you can recreate
s3 intelligent tiering: low latency and high throughput same as standard, automatically moves objects between two tiers
glacier: low cost for archiving/backup, longer terms (10 years), 3 retrieval options (expedited(1-5 minutes), standard(3 to 5 hours), bulk(5 to 12 hours)), min 90 days of storage
glacier deep archive: 2 retrieval options (standard(12 hours), bulk(48 hours)), min 180 days of storage

S3 - object lifecycle: move objects between storage classes
infrequently accessed object to standard ia
archive objects not needed in the real time to glacier
automated using lifecycle configuration
transition actions, expiration actions

s3 performance might get restricted by KMS if you use sse-kms
to improve s3 performance for upload: multi-part upload for large files, s3 transfer acceleration (transfer to s3 edge location)
s3 byte range fetches - parallelize get

server side filtering - select query in s3 and glacier

s3 event notification - trigger based on s3 event, can filtered based on object name as well before triggering
target of event notification - sns, sqs, lambda

version is required whenever you use policy variables

```
{
        "Sid": "AllowAllS3ActionsInUserFolder",
        "Action":["s3:*"],
        "Effect":"Allow",
        "Resource": ["arn:aws:s3:::my-company/home/${aws:username}/*"]
}
```

Whenever a user makes a request to AWS, the variable is replaced by the “friendly” user name of whomever made the request.

aws athena - serverless analytics, directly against s3 files

s3 object lock and glacier vault lock - write once read many, block object version deletion/edit for a time period

AWS CLI, SDK:

perform interactions with aws without online console

interact with aws proprietary service
aws tasks against aws can be done using

- AWS CLI on local computer
- AWS CLI on EC2 instance
- AWS SDK on local computer
- AWS SDK on EC2 instance
- AWS instance metadata service for EC2

during cli installation aws executable's path is generally added to the environment variables
cli configure: aws access key and Secret key (don't share)
get user's access keys, IAM -> Users -> user profile -> Security Credentials -> Create access key -> download
aws configure <- to configure user access, access key id, secret access key, default region; will create config and credentials files in ~/.aws

aws cli on EC2 using IAM roles: assign IAM roles with policy authorizing what the EC2 instance can do
commands on cli:

```
aws s3 ls
aws s3 <command_name> help
```

aws cli dry run (simulate call, check permission) flag --dry-run. On success, the request would have succeeded but the DryRun flag is set.

error message can be decoded using STS command line

```
aws sts decode-authorization-message --encoded-message <err_msg>
```

aws ec2 instance metadata - learn about themselves
http://169.254.169.254/latest/meta-data
internal ip, will only run in the ec2 instance
can retrieve IAM role but not the IAM Policy(to test IAM policy use dry run or policy simulator)

to use MFA with cli, create a temporary session. run STS GetSessionToken API call

```
aws sts get-session-token --serial-number arn-of-the-mfa-device --tokencode code-from-token
--duration-seconds 3600
```

configure the profile and add aws_session_token in the credentials file

perform actions on AWS from your applications
AWS SDK: java, .net, node, python, go, etc.
AWS CLI uses python SDK
when default region is not specified, us-east-1 is chosen

aws limits:
API rate limits

DescribeInstances API for EC2 has a limit of 100 calls per seconds
GetObject on S3 has a limit of 5500 GET per second per prefix
KMS API has rate limit
For Intermittent Errors: implement Exponential Backoff
For Consistent Errors: request an API throttling limit increase

Service Quotas (Service Limits)
Running On-Demand Standard Instances: 1152 vCPU
to request a service limit increase open a ticket
request a service quota increase using the Service Quotas API

credentials provider chain(priority): command line options, env variables, cli credentials file, cli configuration file, container credentials, instance profile credentials
for sdk: env variables, system properties, default credential profiles file, ecs container credentials, instance profile credentials

all of our aws http api requests should be signed(AWS credentials). CLI and SDK do that for us otherwise use SigV4
SigV4: HTTP header, or Query String Option(pre-signed URLs)

AWS CloudFront:

Content Delivery Network (CDN), improves read performance, content is cached at the edge
ddos protection, integration with shield, aws web application firewall
can expose external https and talk to internal https backend

cloudfront origins - s3 bucket, custom origin(http) - alb, ec2, s3 website, any http backend
cloudfront geo restriction - whitelist, blacklist countries

cloudfront uses global edge network and cached for a ttl, good for static content
vs s3 crr setup for each region, files updated real-time, read only, good for dynamic content available to few regions at low latency

origin access identity - users access s3 content only using cloudfront urls

cache based on header, session cookies, query string parameters at edge locations
expire based on ttl
cache can be invalidated using api call
maximize cache hits by separating static(s3) and dynamic(alb, based on header and cookie) distributions

viewer protocol policy: between client and edge location - redirect http to https, https only
origin protocol policy: between origin and edge location - https only, match viewer

s3 bucket websites don't support https

signed url/cookie: attach a policy (url expiration, IP ranges to access data from, trusted signers)
signed url: one signed url per file, signed cookie: one signed cookie for multiple files

CloudFront Signed URL:
Allow access to a path, no matter the origin. Account wide key-pair, only the root can manage it. Can filter by IP, path, date, expiration. Can leverage caching features
S3 Pre-Signed URL:
Issue a request as the person who pre-signed the URL. Uses the IAM key of the signing IAM principal. Limited lifetime

ECS, ECR and Fargate:

Docker containers
ECS: cluster, services, tasks, tasks definition

dockerfile - docker configuration
docker image - built from dockerfile
docker hub - stores docker images
docker container - running instance of docker image

to manage containers, we have container management platform: ECS, Fargate, EKS

ecs clusters are logical grouping of EC2 instances, EC2 runs a special AMI made for ECS
EC2 instances run ECS agent(docker container), these ecs agent registers the instance to ecs cluster

ecs task definition - metadata in json format, tell ecs how to run a docker container
has image name, port binding for container and host, memory and cpu required, env variables, networking info, etc
specify which containers are included in your task and how they interact with each other

ecs services define how many tasks should run and how, ensure no of tasks desired is running across ec2 instances
can be linked to load balancers

dynamic port forwarding in load balancer, use it with ecs

ECR: private docker image repository, accessed through IAM,
cli command: aws ecr get login password | docker login previous command's o/p
docker push and docker pull command

Fargate: in ecs cluster we have to create ec2 instances on our own in fargate we just create task definitions

ECS agent needs a EC2 role to attached to it to function, ecs tasks need role attached based on the services they interact with

task placement strategy/task placement constraints to determine where to place or remove tasks from
task placement strategy: binpack(least available amount of cpu or memory), random, spread (evenly based on parameter(AZ, instance))
task placement constraint: distinct instance, member of(if satisfy the expression than place there)

ECS service auto scaling based on cloudwatch, target tracking, step scaling, scheduled scaling
ecs service auto scaling does not mean ec2 auto scaling
for that ecs cluster capacity provider

To properly instrument your applications in Amazon ECS, you have to create a Docker image that runs the X-Ray daemon, upload it to a Docker image repository, and then deploy it to your Amazon ECS cluster.
You can use port mappings and network mode settings in your task definition file to allow your application to communicate with the daemon container.

AWS Elastic Beanstalk:

deploying application on aws
instance configuration and os is handled by beanstalk
deployment strategy is configurable but performed by EB

three architecture models:
single instance deployment: good for dev
LB + ASG: for production or pre-production web app
ASG only: for non web app in production (workers) (process msgs of a queue)

EB has three components:
application
application version (each deployment)
env name (free naming: dev, test, prod)

deploy application versions to environments and can promote to next environment
rollback, control over lifecycle of environments

elastic beanstalk deployment modes:
all at once
rolling
rolling with additional batches

immutable
swap url, blue/green deployment

beanstalk lifecycle policy based on time or space.
elastic beanstalk uses cloudformation in the back
you can clone eb environment, ex. to deploy test version
beanstalk migration: to change lb or decouple db

Beanstalk with HTTPS

- Load the SSL certificate onto the Load Balancer
- Can be done from the Console (EB console, load balancer configuration)
- Can be done from the code: .ebextensions/securelistener-alb.config
- SSL Certificate can be provisioned using ACM (AWS Certificate Manager) or CLI
- Must configure a security group rule to allow incoming port 443 (HTTPS port)

Beanstalk redirect HTTP to HTTPS

- Configure your instances to redirect HTTP to HTTPS:
- https://github.com/awsdocs/elastic-beanstalk-samples/tree/master/configuration-files/aws-provided/security-configuration/https-redirect
- OR configure the Application Load Balancer (ALB only) with a rule
- Make sure health checks are not redirected (so they keep giving 200 OK)

For custom configuration files which are not readily available in Elastic Beanstalk, create YAML- or JSON-formatted documents with a .config file extension that you place in a folder named .ebextensions and deploy in your application source bundle.
Dockerrun.aws.json is used in multicontainer Docker environments that are hosted in Elastic Beanstalk
env.yaml is used to configure the environment name, solution stack, and environment links to use when creating your environment in Elastic Beanstalk.
cron.yaml is primarily used to define periodic tasks that add jobs to your worker environment's queue automatically at a regular interval
Appspec.yml (application specification) is used to manage each application deployment as a series of lifecycle event hooks in CodeDeploy and **NOT in Elastic Beanstalk**.

CICD - CodeCommit, CodePipeline, CodeBuild, CodeDeploy:

push our code in a repository and have it deployed on AWS
codecommit: storing our code
codepipeline: automating our pipeline from code to elastic beanstalk
codebuild: building and testing our code
codedeploy: deploying the code to ec2

codecommit authorization in Git: IAM Policies manage user / roles rights to repositories
Cross Account access: Use IAM Role in your AWS Account and use AWS STS (with AssumeRole API)
trigger notifications using AWS SNS or AWS Lambda or AWS CloudWatch event rules

We can migrate a Git repository to a CodeCommit repository in a number of ways: by cloning it, mirroring it, migrating all or just some of the branches, and so on. You can also migrate local, unversioned content on your computer to CodeCommit.

codepipeline -> visual workflow
sequential or parallel actions, build/ test / deploy / loadtest, stages can have multiple action groups
each pipeline stage can create artifacts, which are stored in S3 and passed to next stage
codepipeline state change -> AWS CloudWatch Events -> SNS notifications
aws cloudtrail can be used to audit aws api calls
if pipeline can't perform an action, IAM service role

codebuild - build instructions can be defined in buildspec.yml(phases: install, pre build, build, post build)
output logs to S3 and CloudWatch Logs, detect failed builds and trigger notifications using CloudWatch alarms
cloudwatch events and AWS lambda, sns notification
build can be defined in codepipeline instead of codebuild as well
buildspec.yml is the file name for the file that defines the build instructions for AWS CodeBuild.

codedeploy - appspec.yml
file section - how to source them (s3/github)
hooks - instructions to deploy new version

- applicationStop
- downloadBundle
- beforeInstall
- afterInstall
- applicationStart
- validateService
- beforeAllowTraffic
- allowTraffic
- afterAllowTraffic

config: one at a time, half at a time, all at once, custom
The name of the AppSpec file for an EC2/On-Premises deployment must be appspec.yml. The name of the AppSpec file for an Amazon ECS or AWS Lambda deployment must be appspec.yaml.

EC2/On-Premises: in-place or blue/green deployment
AWS Lambda: canary, linear, or all-at-once; traffic is shifted to the updated Lambda function versions during a deployment
Amazon ECS: blue/green deployment, traffic shifting

codestar : dashboard/wrapper around all the application management services

CloudFormation:

infrastructure as code, provisioning mechanism
template(json or yaml) uploaded in s3, can't be updated, upload new version
cloudformation template: resource(services), parameters(!Ref function to refer parameters or resources), mapping(switch case, !FindInMap),
output(optional, can import into other stacks, link stacks, !ImportValue), conditionals(environment, region, etc.), metadata
create, update and delete stacks using this
some unsupported aws services can be created using AWS Lambda custom resources
changeSets, nested stacks, stackSets (across multiple accounts and regions)

Change sets allow you to preview how proposed changes to a stack might impact running resources
StackSets allows us to provision a common set of AWS resources across multiple accounts and regions with a single cloudformation template

CloudWatch, X-Ray, CloudTrail:

CloudWatch:
- metrics: collect and track key metrics
- logs: collect, monitor, analyze and store log files
- events: send notifications when certain events happen in AWS
- alarms: react in real time to metrics/events

X-Ray:
- troubleshooting app and performance
- distributed tracing of microservices

CloudTrail:
- internal monitoring of API calls being made
- audit changes to AWS resources by your users

metric, variable being monitored, cpuUtilization, networkIn
they belong to a namespace(categories), dimension(instance id, environment, etc.) is an attribute of the metric, upto 10 dimensions per metric
metrics have timestamps
detailed monitoring for extra cost to increase frequency (ec2 every 1 minute), ec2 ram usage is not pushed as a metric (make custom push function)
higher metric resolution (storageResolution API parameter), send metric to cloudWatch (PutMetricData api call), use exponential backoffs in case of throttle errors

alarm trigger notifications for any metric, go to Auto scaling, EC2 actions, sns notif.

they have various options (sampling, %, max, min)
alarm states (ok, insufficient_data, alarm), period (length of time in seconds to evaluate the metric)

applications can send logs to cloudwatch using sdk
collects log from - beanstalk, ec2, lambda, vpc flow logs, api gateway, cloudtrail, log agents, route53
can be analysed, batch export to s3, stream to elasticSearch
can use filter expressions
log group(application), log stream(instances within application), log expiration policies
use correct IAM policies to send logs to cloudwatch, encryption at group level using KMS

cloudwatch logs agent, cloudwatch unified agent
metric filter
eventbridge: partner event bus (receive from saas), custom event bus (for your application), rules to process events, schema registry(analyze, structure)

X-ray (debugging), visual analysis, understand dependencies, identify bottlenecks, time sla, find errors and exceptions
tracing (follow a request), each component adds its own trace, segments, sub segments, sampling
IAM authorisation, KMS encryption

code imports the aws x-ray sdk (application sdk will capture calls to aws services, http/https requests, db calls, queue calls)
install the x-ray daemon or enable x-ray aws integration (should have IAM permissions, lambda run them for us, low level UDP packet interceptor on port 2000)
instrument application code
trace (segments collected together, end-to-end request, X-Amzn-Trace-Id), segment(each service/app will send them), subsegments (for granular details in segment)
annotations: key value pairs used to index traces and use with filters
Trace sampling rate: first request each second, five percent additional requests
subsegment can contain additional details about a call to an AWS service, an external HTTP API, or an SQL database. You can define arbitrary subsegments to instrument specific functions or lines of code in your application.
For services that don't send their own segments like DynamoDB, X-Ray can use subsegments to generate inferred segments and downstream nodes on the service map
Segment documents can be up to 64 kB in size
AWS X-Ray daemon is a software application that listens for traffic on UDP port 2000, gathers raw segment data, and relays it to the AWS X-Ray API
X-Ray takes the client IP from the X-Forwarded-For header in the request instead of from the source IP in the IP packet
To configure X-ray SDK use environment variables daemon address(AWS_XRAY_DAEMON_ADDRESS – Set the host and port of the X-Ray daemon listener. By default 127.0.0.1:2000 for both trace data (UDP) and sampling (TCP)), context missing(AWS_XRAY_CONTEXT_MISSING – Set to

LOG_ERROR to avoid throwing exceptions when your instrumented code attempts to record data when no segment is open.), trace id(AWS_XRAY_TRACING_NAME, used be segment)

X-Ray Write APIs:

- PutTraceSegments: Uploads segment documents to AWS X-Ray
- PutTelemetryRecords: Used by the AWS X-Ray daemon to upload telemetry. Ex:
  - SegmentsReceivedCount, SegmentsRejectedCounts, BackendConnectionErrors
- GetSamplingRules: Retrieve all sampling rules

X-Ray Read APIs:

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

cloudtrail: enabled by default, history of event/api calls made by the aws account
something deleted look into cloudtrail

SQS, SNS, Kinesis:

asynchronous/event based communication, app -> queue -> app, applications decoupled
sqs: queue model, sns: pub/sub model, kinesis: real-time streaming

producers send messages -> sqs queue -> consumers poll messages
4 days message retention
can have duplicate messages(at least once delivery), out of order messages(best offer ordering)
256kb per message

delay queue - by default or using message parameter(DelaySeconds)
poll sqs messages by consumer (receive upto 10 at a time)
have to process the message within visibility timeout, then delete(DeleteMessage) from queue using msg id
if consumer needs more time to process a message than tell the queue to increase the timeout (ChangeMessageVisibility api call)

dead letter queue - if the consumer fails to process a message multiple times(threshold, redrive policy), send the message to DLQ (dead letter queue that you created) from main sqs

sqs - long polling, if consumer doesn't find any messages in the queue it will wait for the wait time specified. decreases api calls on empty queues
enable by default or api call WaitTimeSeconds

sqs fifo queue - name must end in .fifo
lower throughput, no per msg delay, sent exactly once
MessageDeduplicationId - to avoid duplicates
sequencing - MessageGroupId, that many consumers

sqs extended client (java library)
sending message larger than 256kb -> producer sends it to s3 and metadata to queue -> consumer receives metadata and extracts file from s3

SNS - one message to many receivers, fan out

Kinesis - big data streaming tool
kinesis streams - low latency streaming ingest at scale
streams are divided into shards/partitions
data can be reprocess, multiple apps can consume the same stream
data is immutable
number shards = number of max kcl possible
kinesis analytics - perform real time analytics on streams using sql
kinesis firehose - load streams into s3, redshift, elasticsearch

AWS Lambda:

no need to manage servers, just deploy the code
lambda - virtual functions, auto scaled and on-demand, short executions
lambda + api gateway : rest api
lambda + kinesis : data transformations on the fly
lambda + dynamodb : database event related processing
lambda + s3 : s3 event can invoke lambda function
lambda@edge + cloudfront :
lambda + cloudwatch events EventBridge : invoke lambda based on our architecture state change
lambda + cloudwatch :
lambda + sns : react to notification
lambda + sqs : process sqs queue messages
lambda + cognito : react when user logs in

example: create a thumbnail on the fly - new image in s3 -> trigger lambda which creates a thumbnail -> stores in another s3 and inserts data to dynamodb
example: serverless cron job - cloudwatch events EventBridge triggers every one hour -> lambda function performs a task

package code and deployment dependencies, upload it to create lambda function, lambda stores code in s3 and encrypts it at rest a layer is a ZIP archive that contains libraries, a custom runtime, or other dependencies. use libraries in your function without including them in the deployment package.

lambda function invocation can be synchronous or asynchronous (on-demand invocation)
synchronous - wait for the result, which is returned right away, when you use cli, sdk, api gateway, application load balancer
error handling on client side
user invoked(alb, api gateway, cloudfront, s3 batch), service(cognito, step function), other(lex, alexa, kinesis firehose) are synchronous

lambda function asynchronous invocation by s3 event, sns, cloudwatch events/eventbridge, codecommit trigger, codepipeline, etc
from cli set flag --invocation-type Event
lambda reads from an event queue, 3 tries on errors after that it should be pushed to dead letter queue

event source mapping, invoke your lambda function when that event occurs
necessary when the records need to be polled (not pushed events), kinesis data streams, sqs, dynamodb streams
lambda function itself synchronous invocation
streams - iterator for each shard, batch records, on error batch is reprocessed, destination(send the result of asynchronous invocation, upgrade compared to dlq)
queue - long polling, batch size, dlq
lambda event mapper, needs permission to read from sqs

by default, AWS executes Lambda function code securely within its own VPC
lambda with ENI
Deploying a Lambda function in a public subnet does not give it internet access or a public IP
Deploying a Lambda function in a private subnet gives it internet access if you have a NAT Gateway / Instance
You can use VPC endpoints to privately access AWS services without a NAT

Triggering lambda function:
- Synchronous: RequestResponse
  - user invoked(alb, api gateway, cloudfront, s3 batch), service(cognito, step function), other(lex, alexa, kinesis firehose)
- Event source mapping:
  - Poll trigger: dynamodb, kinesis, sqs
  - configuration of the event source mapping is on the Lambda side
- Asynchronous: Event
  - Push trigger: s3, sns, cloudwatch events/eventbridge, codecommit trigger, codepipeline
  - configuration is made on the source (S3/SNS)

the number of concurrent executions for poll-based event sources is different from push-based event sources iam role vs resource based policy

to expose lambda function as HTTP(S) endpoint, use application load balancer (or api gateway)
lambda function should be registered in target group
json to http and http to json conversion in alb, and multi header value in target group setting

Lambda@Edge - deploy lambda functions along side cloudfront cdn
run lambda functions to customize content that CloudFront delivers, execute functions in AWS locations closer to the viewer.
functions run in response to CloudFront events
use lambda functions to change CloudFront requests and responses at the following points:
after CloudFront receives a request from a viewer (viewer request)
before CloudFront forwards the request to the origin (origin request)
after CloudFront receives the response from the origin (origin response)
before CloudFront forwards the response to the viewer (viewer response)

lambda environment variables, encrypt using kms

lambda automatically monitors functions on our behalf, reporting metrics through Amazon CloudWatch
CloudWatch metrics include total invocation requests, duration, concurrent executions and error rates, throttles, async delivery failures
throttles, Dead Letter Queues errors and Iterator age for stream-based invocations are also monitored
CloudWatch does not monitor lambda's reserved concurrent executions but can view it through the lambda console

active tracing, lambda tracing with x-ray

lambda - upto 1000 concurrent executions, can be limited using "reserved concurrency", invocation over the limit will trigger a throttle
throttle in synchronous invocation - return throttleError 429
throttle in asynchronous invocation - retry and then go to dlq

first request served by new instances has higher latency than rest - cold start
provisioned concurrency - allocated before the function is invoked, cold start never happens

lambda function dependencies - install the package along your code and zip it together
if zip less than 50 MB than upload to lambda else upload to s3 and reference it

custom runtime or reusable dependencies using lambda layers

create connection to other services outside of function (context), so that it can be reused during invocations

lambda execution limits per regions - max execution time 15 min, memory 128MB to 3TB (64 MB increments),
env variables 4KB, disk capacity in function container (/tmp) 512MB <- local directory, concurrency executions 1000
lambda deployments limits - size of compressed zip file 50MB, size of uncompressed deployment (code+dependencies) 250MB,
can use /tmp directory to load other files at startup, env variables 4KB

DynamoDB:

highly available with replication across 3 az
event driven programming with DynamoDB streams

each table has a primary key(partition key, or partition key and sort key)
each row(item) is made of attributes, max size of item 400KB

throughputs - rcu(read capacity unit), wcu(write capacity unit)
can use auto scaling of throughput to meet demand or use "burst credit"
1 wcu - one write per second for an item upto 1KB in size
1 rcu - item upto 4KB in size, one strongly consistent read per second or two eventually consistent read per second
eventually consistent read: read after write, unexpected response because of replication
strongly consistent read, correct data
if we exceed rcu or wcu then, ProvisionedThroughputExceededExceptions

PutItem - write data (create or replace) to dynamodb
UpdateItem - update attribute of existing record
Conditional write - write only if (to deal with concurrency)
DeleteItem - delete a record, optional condition
DeleteTable - delete whole table and its items
BatchWriteItem - 25 put or delete in one call, upto 16 MB of data or 400 KB per item
GetItem - eventually consistent read by default, ProjectionExpression to retrieve only certain attributes
BatchGetItem - upto 100 items, 16MB of data

Query: partition key (must be equal), sort key (could be comparison), filter expression (client side filtering)
upto 1MB of data or number items specified in limit
can query a table, local secondary index or global secondary index
efficient

Scan: uses entire table to filter out data, consumes a lot of rcu
return upto 1MB of data, pagination to continue reading

parallel scans - faster but consumes even more rcu, use limit to mitigate

Local Secondary index - alternate sort key(range key) of a partition, 5 LSI per table, defined at creation

Global Secondary index - secondary partition key + (optional) sort key on the whole table
this new table can project all attributes or main table's primary and sort keys only or some additional attributes and the main table's primary keys
must define rcu and wcu for the gsi, possible to add later or modify (unlike lsi)

optimistic locking/concurrency - conditional write or delete, check version (as opposed to performing a lock)

Dynamodb accelerator - dax, cache for dynamodb, write through
solves hotkey (too many reads) problem, 5 min ttl by default
upto 10 nodes in cluster, multi az, secure

Atomic counters can be implemented using updateItem

DynamoDB streams - change in db, can be read by Lambda or EC2
24 hours of retention, aws managed shards
Kinesis adapter can be used
when the table is modified send:

KEYS_ONLY - Only the key attributes of the modified item.
NEW_IMAGE - The entire item, as it appears after it was modified.
OLD_IMAGE - The entire item, as it appeared before it was modified.
NEW_AND_OLD_IMAGES - Both the new and the old images of the item.

event source mapping - to read(poll) from stream

dynamodb ttl - delete after expiry date time
to reduce storage and manage table size over time
ttl number - epoch timestamp

transaction - ability to create / update / delete multiple rows in different tables at the same time
all or nothing type of operation, consumes 2x wcu/rcu, transactional read mode

dynamodb as cache for session state (vs elasticache)

write sharding - add a suffix to partition key so that data is distributed
write types - atomic write, batch write, conditional write(in case of concurrent write)

table cleanup - drop table and recreate it
copying table - aws datapipeline(EMR), or backup and restore

RequestLimitExceeded - provisioned throughput exceeds the current throughput limit for your account
ThrottlingException - rate of requests exceeds the allowed throughput
ProvisionedThroughputExceededException - rcu or wcu exceeded or the provisioned write capacity for the global secondary index is less than the write capacity of the base table

API Gateway:

expose rest api
AWS Lambda + API Gateway: No infrastructure to manage, Support for the WebSocket Protocol
Handle API versioning (v1, v2…), different environments (dev, test, prod…), security (Authentication and Authorization)
Create API keys, handle request throttling, Swagger / Open API import to quickly define APIs
Transform and validate requests and responses, Generate SDK and API specifications, Cache API responses

endpoint types:
edge-optimized(default): for global clients, requests routed through cloudfront edge locations, api gateway is in one region
regional: for clients in same region
private: accessed within vpc

deployment stages: api breaking changes, stage variables and lambda alias
linear
canary deployment
all at once

integration type: integrate with backend
Mock - returns a response without sending the request to backend
HTTP - configure integration request and integration response, data mapping using mapping template(vtl), sqs
AWS - configure integration request and integration response, data mapping using mapping template(vtl), for Lambda custom integration
AWS_Proxy - forward the request json payload, receive response json payload, lambda proxy
HTTP_Proxy - forward the request/response, application load balancer

api gateway cache - cache per stage, per method cache setting can override, ttl is 300 seconds
if no invalidate cache policy than any client can invalidate the api cache
A client of your API can invalidate an existing cache entry and reload it from the integration endpoint for individual requests. The client must send a request that contains the Cache-Control: max-age=0 header

you can define a stage variable in a stage configuration, and then set its value as the URL string of an HTTP integration for a method in your REST API. Later, you can reference the URL string using the associated stage variable name from the API setup. This way, you can use the same API setup with a different endpoint at each stage by resetting the stage variable value to the corresponding URLs. You can also access stage variables in the mapping templates, or pass configuration parameters to your AWS Lambda or HTTP backend.

A Lambda authorizer is an API Gateway feature that uses a Lambda function to control access to your API. There are two types of Lambda authorizers:
A token-based Lambda authorizer (also called a TOKEN authorizer) receives the caller's identity in a bearer token, such as a JSON Web Token (JWT) or an OAuth token.
A request parameter-based Lambda authorizer (also called a REQUEST authorizer) receives the caller's identity in a combination of headers, query string parameters, stageVariables, and $context variables.

usage plan:
who can access one or more deployed API stages and methods
how much and how fast they can access them
uses API keys to identify API clients and meter access
configure throttling limits and quota limits that are enforced on individual client
API keys:
alphanumeric string values to distribute to your customers, ex - WBjHxNtoAb4WPKBC7cGm64CBibIb24b4jt8jJHo9
can use with usage plans to control access
throttling limits are applied to the API keys
quotas limits is the overall number of maximum requests

To configure a usage plan
create one or more APIs, configure the methods to require an API key, and deploy the APIs to stages.
generate or import API keys to distribute to application developers (your customers) who will be using your API.
create the usage plan with the desired throttle and quota limits.
associate API stages and API keys with the usage plan.

cloudwatch logs - info about request/response body
cloudwatch metrics - cache hit, cache miss, count, latency(max 29 seconds then timeout), integrationLatency, 4xx error (client side), 5xx error (server side)

HTTP Status Code	
400	Bad Request Exception
403	Access Denied Exception, autorisation issue
404	Not Found Exception

409	Conflict Exception
429	Limit Exceeded Exception
429	Too Many Requests Exception
502	Bad Gateway Exception, usually for an incompatible output returned from a Lambda proxy integration backend and occasionally for out-of-order invocations due to heavy loads.
503	Service Unavailable Exception
504	Endpoint Request Timed-out Exception

x-ray - full picture

enable cors to receive api calls from another domain
options pre-flight request must contain headers:

- Access-Control-Allow-Methods
- Access-Control-Allow-Headers
- Access-Control-Allow-Origin

security
iam:

- great for users / roles already within your AWS account, + resource policy for cross account
- handle authentication + authorization, uses signature v4

custom Authorizer:

- great for 3rd party tokens
- flexible in terms of what IAM policy is returned
- handle Authentication verification + Authorization in the Lambda function
- pay per Lambda invocation, results are cached

Cognito User Pool:

- manage your own user pool (can be backed by Facebook, Google login etc…)
- no need to write any custom code
- must implement authorization in the backend

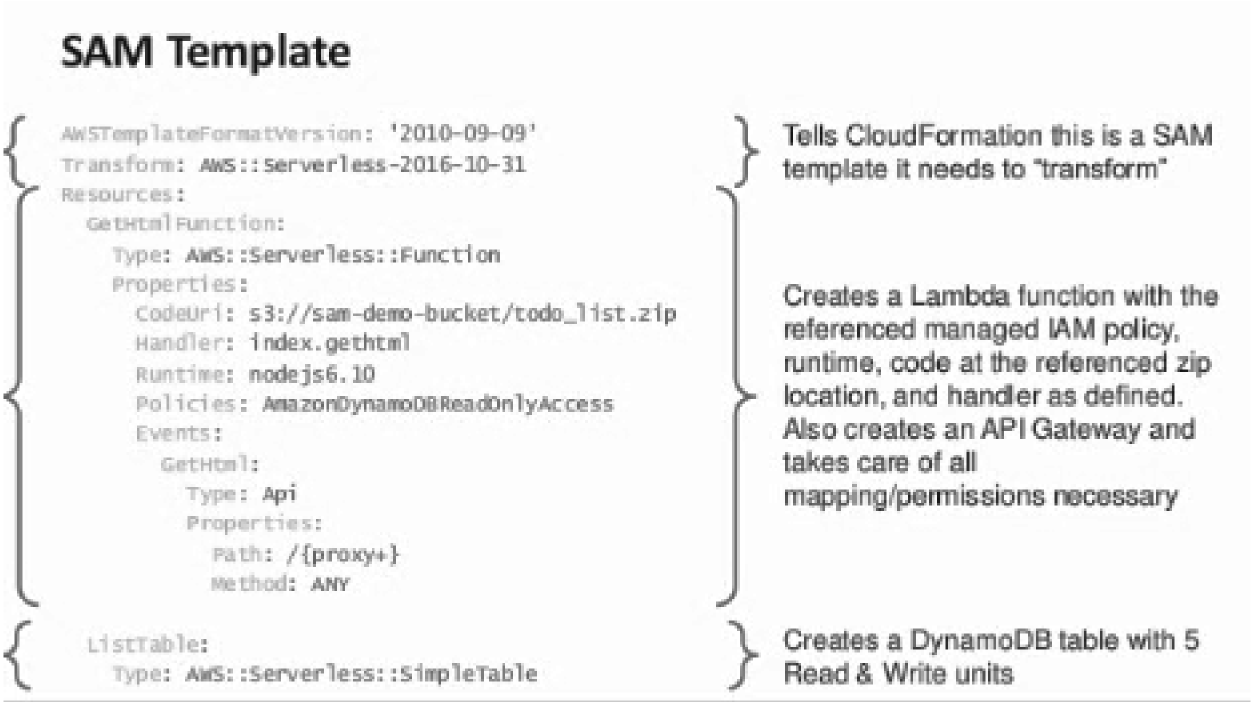
Api gateway integration timeout - 29 sec
Lambda execution timeout - 15 min

SAM (Serverless Application Model):

- configuration using yaml code, creates cloudformation templates
- using codedeploy deploy lambda functions

sam can run lambda, api gateway, dynamodb locally

Transform Header indicates it’s SAM template, Transform: 'AWS::Serverless-2016-10-31'
Write Code - AWS::Serverless::Function, AWS::Serverless::Api, AWS::Serverless::SimpleTable
Package & Deploy - aws cloudformation package / sam package, aws cloudformation deploy / sam deploy



SAM Template

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetHtmlFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://sam-demo-bucket/todo_list.zip
      Handler: index.gethtml
      Runtime: nodejs6.10
      Policies: AmazonDynamoDBReadOnlyAccess
      Events:
        GetHtml:
          Type: Api
          Properties:
            Path: /{proxy+}
            Method: ANY

  ListTable:
    Type: AWS::Serverless::SimpleTable
```

Tells CloudFormation this is a SAM template it needs to "transform"

Creates a Lambda function with the referenced managed IAM policy, runtime, code at the referenced zip location, and handler as defined. Also creates an API Gateway and takes care of all mapping/permissions necessary

Creates a DynamoDB table with 5 Read & Write units

Node.js and Python functions, you can specify the function code inline in the template, zipfile parameter
Changes to a deployment package in Amazon S3 are not detected automatically during stack updates. To update the function code, change the object key or version in the template.

Cognito:

give users an identity, they can interact with our application
cognito user pools: sign in for app users, integrate with api gateway and application load balancer
cognito identity pools (federation identity): users can access aws resources, integrate with cognito user pools as an identity provider
cognito sync (AppSync): synchronize data from device to cognito

cognito user pools - serverless database of mobile and app users, can use federation identity provider
user pool flow, aws lambda trigger: authentication events, sign up, messages, token creation

cognito identity pools (federation identity) - give users temporary aws credentials
access private s3 bucket for example

AppSync:

combining data from one or more sources
managed service that uses graphQl
retrieve data in real time with webSocket
data synchronization

Step Functions:

orchestrate workflow, lambda functions, ec2, ecs, on prem servers, api gateway
represent flow as json state machine
sequence, parallel, conditions, timeout, error handling
max execution time 1 year, manual approval
retry and catch to handle errors

AWS STS:

security token service - grants limited and temporary access to AWS resource (1 hour)
AssumeRole: Assume roles within your account or cross account
AssumeRoleWithSAML: return credentials for users logged with SAML
AssumeRoleWithWebIdentity: return creds for users logged with an IdP (Facebook Login, Google Login, OIDC compatible…), AWS recommends against using this, and using Cognito Identity Pools instead
GetSessionToken: for MFA, from a user or AWS account root user
GetFederationToken: obtain temporary creds for a federated user
GetCallerIdentity: return details about the IAM user or role used in the API call
DecodeAuthorizationMessage: decode error message when an AWS API is denied

aws:MultiFactorAuthPresent:true
GetSessionToken returns: Access ID, Secret Key, Session Token, Expiration date

If there's an explicit DENY, end decision DENY
If there's an ALLOW, end decision ALLOW Else DENY

IAM policies are attached to users, roles, groups
S3 bucket policies are attached to buckets
when evaluating if an IAM Principal can perform an operation on a bucket, the union of its assigned IAM policies and S3 bucket policies will be evaluated

create dynamic policy with IAM, special policy variable ${aws:username}

AWS managed policy, Customer managed policy, in line policy (directly in principal) pass role: iam:passrole

AWS security and encryption:

encryption in flight - https, ssl enabled website
server side encryption at rest - usually data key is used
client side encryption - envelope encryption/client side data key, server can't decrypt data

KMS - key management service - encryption for an aws service
integrated with IAM for authorization
Amazon EBS: encrypt volumes, Amazon S3: Server side encryption of objects, Amazon Redshift: encryption of data, Amazon RDS: encryption of data, Amazon SSM: Parameter store, etc

symmetric keys (AES-256 keys):
aws services integrated with KMS use this, never get access to the Key unencrypted (must call KMS API to use)
necessary for envelope encryption

asymmetric keys (RSA and ECC key pairs):
public key(encrypt) and private key(decrypt) pair
used for encrypt/decrypt or sign/verify operations
public key downloadable, can't access private key

able to fully manage the keys & policies: Create, Rotation policies, Disable, Enable
able to audit key usage (using CloudTrail)
three types of Customer Master Keys (CMK): AWS Managed Service Default CMK, User Keys created in KMS, User Keys imported (must be 256-bit symmetric key)

for data > 4 KB you can't use KMS, use envelope encryption

To give access to KMS to someone: make sure the Key Policy allows the user, make sure the IAM Policy allows the API calls

kms keys are region bound, copy snapshot across regions - snapshot encrypted with key 1, copy and re encrypt with key 2 in another region, restore in another region which will now use key 2

Default KMS Key Policy:
    created if you don’t provide a specific KMS Key Policy
    complete access to the key to the root user = entire AWS account
    gives access to the IAM policies to the KMS key
Custom KMS Key Policy:
    define users, roles that can access the KMS key

define who can administer the key
useful for cross-account access of your KMS key

Copying Snapshots across accounts:
create a Snapshot, encrypted with your own CMK
attach a KMS Key Policy to authorize cross-account access
share the encrypted snapshot
(in target) Create a copy of the Snapshot, encrypt it with a KMS Key in your account
create a volume from the snapshot

encrypt/decrypt api call
if size > 4KB than envelope encryption - GenerateDataKey API, data key dek file, client side encryption using that and than put the encrypted file in envelope along with encrypted data key dek file
AWS Encryption SDK implemented Envelope Encryption for us
Data Key Caching:
re-use data keys instead of creating new ones for each encryption
helps with reducing the number of calls to KMS with a security trade-off
use LocalCryptoMaterialsCache (max age, max bytes, max number of messages)

KMS Request Quotas
when you exceed a request quota, you get a ThrottlingException, to respond, use exponential backoff (backoff and retry)
for cryptographic operations, they share a quota, this includes requests made by AWS on your behalf (ex: SSE-KMS)
for GenerateDataKey, consider using DEK caching from the Encryption SDK
you can request a Request Quotas increase through API or AWS support

can encrypt cloudwatch logs with KMS keys
associate-kms-key : if the log group already exists
create-log-group: if the log group doesn't exist yet

SSM Parameter store: securely store configuration and secrets
can be encrypted using KMS
version tracking, configuration management using path and IAM
notifications with cloudwatch events, integration with cloudformation

SSM Parameter Store Hierarchy

```
/my-department/
    my-app/
        dev/
            db-url
            db-password
        prod/
```

db-url
db-password
other-app/
/other-department/
/aws/reference/secretsmanager/secret_ID_in_Secrets_Manager
/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2

Secrets manager:
capability to force rotation of secrets every X days
automate generation of secrets on rotation (uses Lambda)
integration with Amazon RDS (MySQL, PostgreSQL, Aurora), mostly meant for RDS integration
secrets are encrypted using KMS

Secrets Manager(expensive):
Automatic rotation of secrets with AWS Lambda
Integration with RDS, Redshift, DocumentDB
KMS encryption is mandatory
Can integrate with CloudFormation
SSM Parameter Store:
Simple API
No secret rotation
KMS encryption is optional
Can integrate with CloudFormation
Can pull a Secrets Manager secret using the SSM Parameter Store API

AWS SES:

send emails using smtp, or aws sdk
can receive emails

AWS Certificate Manager (ACM):

host public ssl certificates
ACM loads SSL certificates on the following integrations:
Load Balancers (including the ones created by EB)
CloudFront distributions
APIs on API Gateways
Use IAM as a certificate manager only when you must support HTTPS connections in a Region that is not supported by ACM. IAM securely encrypts your private keys and stores the encrypted version in IAM SSL certificate storage. IAM supports deploying server certificates in all Regions, but you must obtain your certificate from an external provider for use with AWS. You cannot

upload an ACM certificate to IAM. Additionally, you cannot manage your certificates from the IAM Console.

Amazon Inspector:

Using Amazon Inspector is an automated security assessment service that helps improve the security and compliance of applications deployed on AWS.

Quicksight:

Amazon QuickSight is a fast, cloud-powered business intelligence service that makes it easy to deliver insights to everyone in your organization. As a fully managed service, QuickSight lets you easily create and publish interactive dashboards that include ML Insights. Dashboards can then be accessed from any device and embedded into your applications, portals, and websites.

CloudHSM:

AWS CloudHSM provides hardware security modules in AWS Cloud. A hardware security module (HSM) is a computing device that processes cryptographic operations and provides secure storage for cryptographic keys.

OpsWorks

It is a configuration management service that provides managed instances of Chef and Puppet.

AWS CodeStar

It provides the tools you need to quickly develop, build, and deploy applications on AWS.

With AWS CodeStar, you can use a variety of project templates to start developing applications on Amazon EC2, AWS Lambda, and AWS Elastic Beanstalk.

The project dashboard in AWS CodeStar makes it easy to centrally monitor application activity and manage day-to-day development tasks such as recent code commits, builds, and deployments.

Because AWS CodeStar integrates with Atlassian JIRA, a third-party issue tracking and project management tool, you can create and manage JIRA issues in the AWS CodeStar dashboard.

Whitepapers:
Serverless architecture with AWS lambda
Running containerised microservices on aws
Microservices on AWS
Api gateway faq
Cloudwatch event faq