**General Information:**

Never going to be tested on numbers

Amazon monitors total resource usage and add infrastructure whenever it reaches certain thresholds

with AWS, ideally you tag everything and as much as possible

In AWS CLI, help shows all command options

Terms, limits, quirks, comparisons.

Focus: IAM -> VPC -> S3 -> EC2 -> RDS

**AWS Limits:**

5 elastic IP per region.

100 S3 buckets per account

50 domain names in Route53

**Performance:**

S3: at least 3500 PUT/POST/DELETE and at least 5500 GET per second. Create multiple prefixes for parallelize process

**Terms:**

Scale up: add Ram/CPU; Scale out: add more servers

Power user: User who has all access except IAM management.

Admin user: User who has all access including IAM management

AWS global infrastructure: Region - AZ - Edge Location, One AZ can contain more than one datacentre, but they are connected with super-fast network and latency is negligible, logically they can be deemed as one datacentre. Different AZ normally has different power supply and other support that's independent from each other's failure

AZ names are randomly allocated to every AWS account, to balance usage

AWS enables architects to test their approaches, because we don't need to guess infrastructure now

**AWS services available:**

Elastic beanstalk: features heavily in developer certification, enable developers to focus on code and automatically envision resources.

EC2 container service ECS: AWS docker

LightSail: Provision a server and takes care of everything, no knowledge need

Batch: not covered in any exam, used for batch computing (Maybe check if it works in conjunction of SWF, used to automate processes)

EFS: basically NAS you can mount to virtual machine (Try to figure why don't use a S3 for all the EC2 instances)

ElastiCache: only for database services, general purpose use cloud front

Migration Hub: monitors all the migration services (DB, application, server, etc.)

CodeStar: Project managing code integrating all the following developer tools

* CodeCommit: private source control
* Code build: compiler
* Code Deploy: automate deployment to EC2, lambda, onsite server
* Code Pipeline: CI/CD tool
* X-Ray: debugging serverless applications

CloudWatch: huge topic in sysops

CloudTrail: keep trail for all activities, highly recommended to turn on all the time.

Config: visualise AWS environment and monitor configuration (resource level)

OpsWorks: centralised configuration management, chef and cook and recipe comeback later

Service Catalog: Manage all the service you have, like all consulting companies have, not tested

Systems Manager: for EC2 maintenance and management, not tested, important for sysops

Trusted adviser: human AWS advisers about security, cost, etc

Managed Services: let AWS manage everything you need, don't need to set up yourselves

Media services: not tested

Machine Learning: Investigate more when all associated level tests are done

Redshift: DWH or BI, very complex queries

Direct connect: physical line between premise <-> VPC, used for edge computing and extent existing solution into the cloud

CloudFormation: people actually open source CloudFormation templates, CSV, YAML, XML and JSON are all data formats (rather than languages) but only JSON and YAML can be used to create CloudFormation templates.

Polly and Rekognition: text -> voice, voice -> text

Athena: search everything in S3 (say something in a CSV file)

EMR (elastic map-reduce): Big data tool, batch and parallel process large data set

Kinesis: ingesting large amount of streaming data into AWS.

QuickSight: AWS's new BI tool

Data pipeline: moving data between AWS services

AWS glue: ETL services

Cognito: Oauth of AWS

Inspector: testing agent running on EC2, can schedule

Macie: search S3 bucket for PII (personal id information), say you stored password on public S3

Certificate manager: free SSH certificate for domain registered through AWS and using route 53

WAF: application level (level 7) security, can monitor user activities etc.

Shield: DDoS protection, if attacked then it will not be billed

artefact: compliance

Mobile Hub: AWS backend for mobile app

Pinpoint: target push notification for mobile app

AWS AppSync: updates apps in real time

Device farm: testing with real devices, say 50 different android phones

Mobile Analytics: analytic tool for mobile apps

Step Functions: workflow control for lambda

Amazon MQ: rabbit MQ, message que

SNS, SQS, SWF: application integration services, Amazon uses SWF for online store

Amazon connect: cloud call centre

Simple email service: ending large amount of email, professional level topic

Alexa for business: business assistant like Siri

Chime: video conferencing tool, works well with low bandwidth

WorkDocs, WorkMail: Amazon's version of office 365

Greengrass: gateway for all IoT devices

GameLift: develop game in AWS (VR games)

**IAM (universal service, Security group and ACL don’t span VPCs):**

Can creating users and granting access, gives centralised control

Granular Permissions: identity Federation with Facebook, LinkedIn, etc.

Can do multi-factor authentication too, and it is recommended

Provide temporary access for users/devices and services. When AWS access has to depend on another set of credentials, create a federation proxy or identity provider and then use AWS security token service to create temporary tokens.

Granular Permissions: identity Federation with Facebook, LinkedIn, etc.

Manages password strength and their rotation policy (expiration, reuse, etc.)

integrates with other AWS services

Has User, Group (for users only that need identical access, users can inherit access from a group policy), Roles (for resources, including users), both group and roles use policies (JSON with access granted or denied)

When a new user is created, they have no access to anything by default, not even changing password, unless "require change password when login" is ticked or password management access is granted

When clicked into a user details, all policies attached for any reason (group, role, inline) will be displayed

When creating a resource in AWS, create a role to lock down low limit on what it can do

policies: A JSON document that defines permissions, then attach to user/group/roles. A user can have multiple policies, say one inherited from group and one attached directly for him (or inline). When policies contradict each other, denies overwrites allow, and AWS always allows only minimum access (align with security best practice).

It is recommended to only login to root when absolutely needed. Only one MFA is allowed for root, if MFA is lost you need verification process to get MFA removed.

Access key ID and secret access key for users can only be used in CMI or SDK, can be deactivated by root, for console access always user name and password. Private access key can only be acquired once upon creation. Access key pair must be regenerated if lost, cannot be recovered.

Account is used to uniquely identify an AWS account, it’s hard to remember so can be customised to anything, as long as it is universally unique.

Cross account access: Say allowing dev account access production account as read-only

* Create a developer in dev account, associate with a new group (say cross-developer)
* Get the account ID of dev account
* Log into the Production Account and create a policy which only has access to the resource that developer is allowed to see, which will then be attached to cross-account role in production account.
* Create cross account role, which is a special kind of role called “role for cross account access”, add the account ID of the developer account, attach the cross-account policy
* Give the new group in developer account an in-line policy, make it able to Assume Roles in IAM: production account: role/role for cross account access
* Logon to developer, select switch role from top right corner, give account, role name and switch
* You’re in production account with “role for cross account access”

**CloudWatch**

Billing must be activated, everything other than “out of free tier alarm” it’s not on by default. Can export report to S3. Can send PDF to email. Email notification can be one email or a list.

You can create customised dashboard within cloud watch for different use cases

Available metric includes disk, CPU, network and status, basically whatever charges money

Custom metric, like monitoring ram, EBS volume, is also possible, need to write code and run on EC2, SysOps topic

On Standard monitoring, alarm will work every datapoint (5 mins), if paid you get alarm for 1 min

CloudWatch event helps respond to state changes and notify whoever should concern, use lambda function

Not tested CloudWatch Log can go down to app layer monitoring, need to install an agent

Metrics can be viewed straightaway, without having to create a dashboard

**Storage:**

S3(global namespace, key-value based, but has a region when created):

* File size can be from 0 to 5TB, the is no limit for total storage, consider bucket as a folder except it has a universal namespace. One account can have 100 buckets at most.
* Read after write consistency happens almost instantly, eventual consistency can take seconds to sync through availability zone (only for overwrite, not new puts, S3 default setup, not user setup, not available for One Zone obviously)
* Standard built for 9\*4 availability, guarantee 9\*3. 9\*3 availability for IA, guarantee 99%, One Zone IA only have a 99.5% availability, guarantee 99. They all have 9\*11 durability.
* Lifecycle management/versioning/encryption/bucket policies (can be used to deny a specific user access)/access control list (ACL) for individual files
* S3 standard and IA can survive 2 AZ lost (stored in 3AZ), one zone - IA cannot, if lost its lost, so better use for re-producible file (still stored redundantly but only one zone). IA charges retrieval fee by GB.
* RRS: if comes up it is used for things that can be easily re-created. It sacrifices durability to 99.99%. Used for thumbnails, transcoded video, sacrifices durability. It's the most expensive and nobody use it now.
* Pricing: Storage, request, storage management, cross region transfer(replication), and transfer acceleration (CloudFront + backbone) cost money
* S3 transfer acceleration: Users using edge locations and AWS backbones to upload files to Bucket, URL: bucketname.S3-accelerat.amazonaws.com
* Standard URL: bucketname.s3-web-reigion.amazonaws.com
* Glacier: expeditated 3-5 mins, standard 3-5 hours, bulk 24 hours, IA and standard are million seconds. IA doesn't mean slow, just more expensive if access too frequently
* Bucket name must be unique, and it's associated with a DNS address
* You can give other AWS account S3 access, and do cross-account logging, commonly used within organizations
* Remember if successful from CLI you'll get a HTTP 200
* Client (encrypt then upload) or Server-side encryption may be used, down to object level. In transition, it uses SSH/TSL using https.
* Server-side encryption: SSE-S3, SSE-KMS (key management service), SSE-C(customer), and SSE plus Client-side encryption are the four encryption types (Check in depth)
* By default, the bucket and all objects are private
* S3 objects do not inherit bucket tags
* When a public file gets updated, it stops being public by default. Remember this because when hosting a static website on S3, all linked files must be public.
* S3 can be used to build simple FTP application
* For S3 you don't have to handle load balancing or other infrastructures, it scales.
* For static domain hosting, domain name must be the same as bucket name if the website is going public, traffic cost is negligible.
* S3 error message are in xml by default
* S3 versioning:
  + Keeps every single version of the tracked file, for big files it can be expensive.
  + S3 versioning can only be suspended but cannot be disabled once enabled.
  + S3 versioning work in the form of overwriting same name objects.
  + S3 versioning integrates with lifecycle rules
  + S3 can config MFA delete. S3 will put a delete marker if a file is deleted, and when delete marker is also deleted, the file will be restored.
* S3 cross-region replication:
  + Cannot be applied in the same region, and obviously no need.
  + Replication can be in another account, say there's a backup account for the entire organization.
  + S3 cross-region replication can be sub-folders or entire bucket. Requires versioning enabled on both buckets. Can be glacier if it's just a backup, or it can be any S3 storage class, commonly S3-IA.
  + Upon the creation of a replication bucket, existing files won't be copied, only new one will be replicated automatically. So existing files must be copied through. Easiest way to copy the existing files to replication bucket is to use recursive copy in CLI.
  + When object is deleted, replica bucket doesn’t delete it to protect from malicious deletions
  + The deletion of delete marker won't be synced anyways
* S3 Lifecycle management:
  + Doesn't require versioning but can manage current and previous versions separately.
  + S3 Lifecycle management can be used when say data is only relevant for a certain period.
  + You can setup different tiers of storage for each stage of lifecycle, S3 -> S3 IA -> Glacier -> delete(expired). Lifecycle rules can be set differently for current and previous versions.
  + When an object is transitioned into glacier, it remains a reference in S3 to trigger restoration. However, if an object is directly stored into glacier, you must download and re-upload

EFS:

* Elastic version of block storage, redundantly stored in multiple AZ. Uses network File System version 4 protocol, can support thousands of concurrent NFS connections.
* EFS and EC2s using it must be in the same security group
* Mounting instruction and command are provided in the console, can be mounted on any location, just run the provided command in terminal (maybe replace where you want to mount, can be any folder)
* EFS can be used to centralised mountable storage managing your application content, just like it's stored in EBS, so no need to copy like S3
* But EFS is still a file server, with permission management. It's better than EBS because can be mounted to multiple EC2s at once. So, can have 2 servers serving out the same file, and don’t need multiple copies of data.
* When used in conjunction with CloudFormation, just mount an EFS to any EC2 that’s provisioned by auto-scaling group, and it became a server without running any batch.
* EC2s connect to EFS through mount targets in each AZ, therefore it is recommended you have a mount target in each AZ.
* Can give EFS file/directory level permissions. Can encrypt files and have multiple performance level (General, Max I/O) and throughput mode (Bursting-Provisioned) available

CDN and CloudFront

* CDN is not quicker for the first user, and edge location only cache data for a certain period (TTL, time to live). TTL should not be too long for frequently changing objects, and decommission objects cost money. TTL is set by the object being cached
* CloudFront work with non-amazon, even on-premise services too, just set your source as "origin"
* A distribution is the name of a collection of edge locations (like a distributor), and it is the first thing you create in CloudFront
* There’re two different types of distribution, RTMP (real time messaging protocol, for adobe flash) and web distribution
* Once CloudFront is setup, you can block direct origin access, and use distribution only
* Cloud can filter user by only accepting sign URL or signed cookies. (Use case: secure CloudFront by only allow paid customer to access) However, if geo-restriction is activated, even signed URL won't work, because the door’s not there even you have the key.
* WAF works with CloudFront nicely
* CloudFront has global load balancing facilities, and works with all kinds of content
* CloudFront allows Geo-distributions, black or white list.
* Even when origin is disabled, CloudFront can still serve cached content

Storage gateway: must download onto VM in your datacentre, consists of a local storage/cache and a buffer that handles multi-part upload. works as an attached file system for S3

Types: File (NFS), Volume (Stored/Cached, iSCSI), Tape(backupVTL)

File gateway backups local datacentre to S3

EBS are not encrypted by default, but can be encrypted.

Stored volume backups EBS snapshots to S3, a complete copy is kept locally, maximum 16 TB. Cached volume only keeps recent state locally and rest in S3 as EBS snapshot, Maximum 32TB. Tape backups physical tapes to virtual tapes.

Snowball: Once Import/Export, reinvented for better management. Bypassing internet for large data set, snowball/snowball edge/snowball mobile

Snowball can import to or export from S3, Glacier must be restored into S3 first to use snowball.

Standard Snowball: 50/80 TB size, 256bit encryption, erased after data received and verified. Snow edge: 100TB, has compute capacity (use like a black box on plane). snowmobile: 100PB transition in truck for data centre migration

Snowball needs a local software client (CLI) to access, and credentials can be obtained from the console. You need manifest file and private key to access snowball.

Connect through ethernet and It works as the S3 buckets, except the physical device is local. You can compare performance before and after in different regions with showcase link, the further the faster

Multipart upload is faster and is a must for single file bigger than 5GB (limit for PUT). It is recommended for anything larger than 100M as it can:

* Improved throughput - You can upload parts in parallel to improve throughput.
* Quick recovery from any network issues - Smaller part size minimizes the impact of restarting a failed upload due to a network error.
* Pause and resume object uploads - You can upload object parts over time. Once you initiate a multipart upload there is no expiry; you must explicitly complete or abort the multipart upload.
* Begin an upload before you know the final object size - You can upload an object as you are creating it.

**EC2 & computing:**

Supported virtualisation type: Paravirtual Machine (PVM) and Hardware virtual Machine(HVM)

Runs on hypervisor, which run on host hardware. ec2 class/type naming: i3.xlarge, “i” means class, 3 means generation, xlarge means size, generally follows 2X rule. Letters for instance types never change, but generation might change.

Images are maintained by both AWS and community

Instance type (Not tested):

* M5: multipurpose, general purpose, Skylake CPU
* T2: General Purpose Burstable instances, cheapest, can be bursted a bit but for limited time. T2 unlimited can be bursted indefinitely
* R4: RAM optimised, 8:1 memory ratio
* X1/x1e: 2T memory, super memory optimised, 16:1/32:1 memory to CPU ratio
* I3: I/O intensive, 3.3 million IOPS, cheapest per IOPS
* EC2 Bare Metal: All support (disc management, networking, etc) are offloaded, so no hypervisor needed, can be used as dedicated host
* D2: lowest cost per unit storage, using magnetic storage, for DWH, log processing, etc. High throughput application
* H1: more CPU, less storage D2, for big data, MapReduce, application need processing a lot but don't store. cheaper than D2
* C5: compute intensive, custom Skylake, 2:1 memory CPU ratio, gaming, scientific, video encoding etc. Netflix uses it
* G3: graph intensive, parallel computing, huge number of cores, can do ML, AI, scientific research (weather, air dynamics)
* New elastic GPU: attachable GPU pay as you go, OpenGL compliant
* P3: Powerhouse, GPU based VM, Nvidia Tesla chip, for HPC (high performance computation)
* F1: FPGA (Field programmable gate array), ability to program the hardware, thus " circuit gate array that's programmable on field", bake algorithm onto hardware. images available in marketplace. FPGA functions are highly independent (helps parallel processing), highest performance compare to a micro controller, High frequency trading use them too. VHDL and VERILOG language. Custom data width (not just 32/64 bit) and operation is all possible (3bit computer)

Reserved instance: convertible instances is possible if you spend more. Reserved instance can be scheduled too.

Spot instance: as soon as spot price goes over you bid it will be terminated, you won't be charged if it's terminated by AWS

Dedicated host: to deal with stupid regulation that against multi-tenant virtualisation. Can be on-demand or reserved. Once set up both VPC and its instances cannot be changed back to default hosting.

EBS used to boot OS is the root drive, root volume cannot be encrypted unless use third party tool or turned into snapshot first. You cannot detach root volume and attaches to another instance

EBS classes:

* General purpose SSD(GP2) can do up to 10000 IOS, but more likely 3000
* Provisioned SSD (IO1) can go up to 20000 IOPS
* Throughput optimised HDD (ST1, for large data, log)
* Cold HDD (SC1, file server)

bootup script can be applied, can be in the form of plain text or file

By default, the EBS attached to EC2 are detached and deleted on termination. Only stored redundantly in single AZ.

SSH IP can be restrict down to my IP address from the terminal, but you may lose access after several hours when ISP changed your IP. For SSH logging on from mac/Linux, use SSH ec2-user@publicip -i keypair.pem. If using putty, username can be stored in Putty from the beginning.

To terminate an instance with protection, go to settings disable the protection first

System check: infrastructure is alright

Instance check: if instance OS works

Security group is layer 4 virtual firewall, any new rules will be applied immediately, It’s first line of defence for EC2

By default, all inbound is blocked, all outbound is allowed. Any inbound rule automatically introduces outbound rule (being Stateful)

You can only white list inbound with security group, black list need NACL

Because you cannot black list with security group, you can assign more than one group to one EC2

EC2 instance and EBS must be in the same availability zone, otherwise the latency won't make it work. EBS volumes now can be modified on the fly (size and type), which wasn't possible before

To deploy in another region, copy the image to that region, you can encrypt in the copy process

EBS Snapshots/ EC2 Images (AMI):

* Snapshot are for volume, exist on S3, snapshots are incremental.
* To migrate EBS instance to another AZ, create a snapshot, store as image (AMI), and re-implement in the other AZ.
* Best practice is only taking snapshot when EC2 is stopped so everything is flushed, but it is possible to take one when EC2 is running
* If taking a snapshot of a RAID array, you must: freeze file system, unmount RAID and shut down EC2 then take snapshot.
* Snapshot vs image, image is for the whole EC2 including OS and license and directly launchable, snapshot is only for one volume.
* Snapshot of encrypted volume is encrypted, restored volume from encrypted snapshot are encrypted
* Encrypted snapshots cannot be shared, because key is controlled by AWS account.
* There's a lot of special AMI for sale when launching a EC2, launching from your own image is also an option
* AMI is a great way to store your common steps for a solution, so when you launch an instance it works straight away. AMI based auto scaling is basically auto-heal, highly recommended
* EBS backed instance has the option the keep the root volume, instance store doesn't have this option
* Instance store might be cheap, considering IO cost and volume. But it really is not used a lot.

Load Balancer:

* Types: Application (intelligent, layer7 for routing different requirement HTTP(S)), Network (layer7, for performance, TCP), classic (ELB, layer 4, legacy, don’t use), can cross AZ
* Load Balancer error 504: balancer is still there, but connection to EC2(gateway) timeout
* When using a classic ELB, User IPv4 address can still be passed in within the X-Forward-For header so you can know who accessed your page
* ELB can launch instances from instance template
* It is possible to load balancing subnets, but it has to be no more than one subnet per AZ, and at least two AZ.

When trying to access S3 from a EC2, it's fine if they are in the same region, if not, use the region flag

Remember the URL to check the metadata of a EC2: curl http:169.254.169.254/latest/meta-data/ (curl is a command line tool to fetch a URL)

Auto Scaling:

* use launch configurations to config standard instance, instances will be deployed evenly in different AZs. However, to change configurations of EC2 you must start over
* Spec is defined in launch configuration
* Auto scaling group typically receives traffics from load balancer
* Health check grace period: when to perform health check again after scaling is performed
* Auto scaling happens only in one region, multiple region must use Route53 to config

Placement group: multiple EC2 running close to each other for faster interaction, only certain type can be used, AWS recommend using identical instances. Separate: still close, but different hardware, multi AZ, clustered: one AZ; they all cannot merge, cannot move in instances, do not span VPC.

You can run container still works on EC2, just don't have to worry about OS.

Lambda:

* AWS takes care of everything you need to run your code, even load balancing or scaling configurations
* Lambda runtime includes: Go, C#, Python, Java 8, Node.js, you get 1 million free invocation per month
* Different types of lambda triggers: API gate way, Alexa, S3, Kenises, DynamoDB, SNS, CloudWatch, CloudFront
* Lambda can trigger another lambda. If multiple users trigger then there will be multiple lambdas, even though the code is same
* Lambda is charged by request, and execution time, it could be cheaper than servers running full time, function max running time is 5 mins(Microservices)
* Alexa use lambda to respond
* Architecture can be complicated, use AWS X-Ray to debug
* lambda can do things globally, like backing up from a S3 to another s3

Elastic container Service: (Docker, come back for exam tips)

* Remove inconsistency among dependencies, and guest OS differences. Wrap up all the dependencies as a container so the environment is always the same
* Docker can be used in build, test and deployment, highly reliable and scalable
* Difference from virtualisation (run code on virtual machine): virtualisation still have guess OS and dependencies, that consumes a lot of resources. Docker image is much lighter weighted
* Make micro-services more feasible
* ECS is a better way to manage docker containers on a cluster of EC2s, managed docker
* Regional, Multi-AZ
* Docker image: Basically cloud formation for docker, setting container up with a plain text script, script is managed by AWS ECR
* ECS task Definition: JSON file describes several docker containers to be created for an app, including port access, data access and command access, env variable setting etc
* ECS task definition maintains several docker cluster at all time, working as a load balancer

**Route 53:**

Important!!: understand different routing types/policies

There’re only so much top-level domain names, can be searched in root zone database. Top level domains are used to verify email, so newly registered first level domain may be rejected

.com can be top or second or any level domain, consider .com and .com.au

Supports NS, PTR, SOA, SPF, SRV, TXT types….

domain registrars assign domain name under one or more top level domains, and then registers with ICANN (internet corporation of Assigned Names and Numbers), which enforces uniqueness of domain names across the internet, and store then in a centralised "who is" database. godaddy.com is another popular one other than amazon route 53

It's called route 53 because DNS operates on port 53

Start of authority: SOA record stores information about how long the routing setting remains valid (TTL). TTL takes effect on local machine (can be cleared with hard refresh) or resolving server. The shorter the faster you can make changes.

When an address is requested -> ISP or cache -> first level domain DB -> registar NS -> SOA -> ‘A’ record or alternatives

Load Balancer don't have IPv4 addresses, so always use a DNS service to point to it. Therefore, cannot use a ‘A’ record for ELB.

"A record" doesn’t have to be naked name, it points directly to IP address, C name points to other domain names (e.g. to redirect to www.XXX.com to mobile.XXX.com).

A C-name (Canonical Name) cannot be used for naked domains. So if you registered multiple domain names for one product, and these domain names are different at bottom level, you can use C name record to route all these domain names to one A record, then point to one IP.

Alias name can only point to some AWS resources (Alias target) like ELB, and other records. Always choose an Alias name over a C name when applicable. That way when ELB address changes, route 53 will re-route straightaway, and it supports naked name. (a stable and recommended practice to point to AWS resources).

ELB has an internal IP address just never exposed to you, so it is a must to use an alias record to resolve domain name to DNS name. Say the .com name server is down, it can re-route to .net name server and find your IP/DNS name.

Route 53 is a Global service, make sense

A Route53 domain has the same domain name registered under multiple top-level domain names (as its internal NS) to prevent outage.

Routing policies (important), can be applied to A record (and those aliases which act as one):

* Must choose one policy for ‘A’ record, cannot mix and match multiple.
* Simple routing policy: default, point a domain name to a server, it is one record with multiple IP. Cannot even do health check, if need health check, use multi-value.
* Weighted routing policy: route to different server based on weight assigned, multiple record with the same domain name in route53, weight will be added up and pro-rata to %. Case: just want to keep a server running in case the main one is down. Each route has an independent TTL, refresh with the TTL for the current IP will not result in different result.
* Latency based routing: route traffic to provide the lowest latency, route then select the fastest region; set up multiple record with the same URL and point to ec2/ELB in different regions
* Failover over: health check based, only allow one active and one passive, cannot manage multiple node. once the active site goes down, route53 reroute to passive site.
  + This health check is under route 53, can monitor both IP or domain name (for ELB that doesn't have public IP), can monitor endpoint/endpoint collective behaviour or CloudWatch, check’s certain path and can send SNS if needed. Can set repeat counts, like only report after 3 fails
* Geolocation: Specify what resource response to which geolocation, say a shop shows euros in Europe shop and dollars in us shop. Won’t take latency into account, you can route request to very far resource if you want to.
* Multi-value answer: attach a health check on each EC2 node, when all healthy it's like simple routing, when not dead node will not be used. like ELB but not exactly the same. Consider the meaning of this policy name “multiple answers are correct”.
* Geoproximity: route base on latitude and longitude, not region

By design, Route 53 doesn't allow DNS information accessed by external resources. To work around this, use a public AWS resource to call R53 within the VPC (like a public EC2).

**Database:**

To move RDS between VPCs, use snapshots and recreate

Relational vs NoSQL: For relational, you must understand the data in advance, otherwise it is hard to change after, or you have to insert null value

Ways to lower latency: add read replicas, use direct connect, put database closer to you customer or use CloudFront/ElastiCache

Aurora is AWS's RDB, better performance than MySQL and better disaster recovery

DynamoDB: stores everything as JSON, features heavily in developer's certification test

the combined key and name length must be smaller than 400k

Online Transaction Processing (OLTP) vs Online Analytics Processing (OLAP): OLTP is more like processing online shopping transactions, OLAP deals more complicated query (RedShift)

RDB is mostly used for OLTP, high IO and low throughput

Data warehousing DB is normally a copy of production DB, so that production DB will not be over pressured

ElastiCache: in-memory caching, Support MySQL, MariaDB, PostgreSQL, Oracle, and Microsoft SQL Server.

ElastiCache is designed to reduce latency and improve performance, cached information may include read intensive database result or computation-intensive calculation result

ElastiCache vs read replica: cache if a small amount of resource is accessed frequently, read replica is full capacity of RDB is frequently required

ElastiCache types: Memcached for objects and Redis for key-value pairs. Memcached has no multi AZ, Redis has multi AZ, It is basically a CloudFront for cache the most common queries (e.g. top sellers), thus allows better performance for web-apps

RDB instance doesn't have a public IP address either like ELB, only a DNS name, AWS manages the mapping from the DNS to the internal IP address

If EC2 cannot connect to a DB instance, most likely it is due to port 3306 not being accessible to the security group that EC2 is in, you should open port 3306 to EC2's security group

Automated backup: enabled by default, stored in S3, size equal to your DB, daily snapshot + transaction log, in case of recover (manually triggered), AWS restores by apply logged transactions to the latest snapshot

Snapshots: manually, user initiated, stored after RDS instance is terminated (automated backup will be deleted), normally comes with several seconds of down time.

whenever an RDS snapshot is restored from a snapshot, it will be a new instance with a new endpoint (DNS address)

Once a DB is encrypted, all data, snapshots, read replicas, and automated backups are encrypted. Encrypting an existing RDB instance is not possible, you can only create a snapshot and encrypt during the copy process.

Multi AZ - any changes made on an RDB in one AZ will be duplicated in another RDB in another AZ, for redundancy, in the case of losing active DB, it will fail over to the duplicated in another AZ

In the case of multi AZ, when failing over to secondary RDB, the endpoint (DNS) does not change, AWS handles the mapping to the new private address. It is used for database maintenance too and no admin intervention required.

Read replica: redundancy for performance boost. Default for Aurora, available for other RDB but need to be turned on. You can have up to 5 read replica. They can be in different AZ or even region.

Read replica is not available for SQL Server or Oracle, only AWS and open source RDBs, must have automatic backup turned on in order to deploy read replica, each replica has different endpoint.

Read replicas can be used in conjunction with multi AZ, a read replica can be promoted into a standalone database for other things

Read replicas do replications Asynchronously

Upon the creation of read replica, you can specify region, AZ and whether to encrypt the database

Multi AZ can be set up when needed, e.g. before deployment, as it is just a standby instance, does not relevant to dev or testing

DynamoDB, single digit latency, always on SSD, 3 geographically distinct data centre (not AZ, but still different facility)

Consistency model: if need read within one second write, use eventual consistent read, if need less than one second, use strong consistent read

DynamoDB is charged based on read capacity unit and write capacity unit, one read capacity unit can handle 1 strong consistent read per second or 2 eventual consistent read per second, up to 4kB

One write capacity unit can handle 1 write per second up to 1KB. So, it can be expensive for write, but cheap for read, because of the unit size.

For read/write capacity unit, you can pay-as-you-go or reserve for 1 year to 3 years. You can monitor in the metrics to compare between provisioned and consumed to change on the fly.

Scaling DynamoDB does not have down time (called push button scaling), but scaling RDB might have

Redshift can have a single node of 160GB, or multi node configuration with a lead node handling client communication, and up to 128 compute nodes

Columnar data store - Redshift actually store Colum data sequentially on the media, which makes column query more IO efficient

Columnar data store make compression more efficient, since they're all same type of data. You don't have to worry about what scheme to use, AWS select the best for you.

Redshift uses Massively Parallel Processing (MPP) to boost fast performance, achieved by multi-node configuration

RedShift is charged by hour (only compute node, unit is "one instance hour"), and backups, and transfer within a VPC

Redshift is currently only available in 1 AZ, but can restore snapshots to another AZ for potential recovery needs

Aurora (Probably will not be tested):

* bespoke SQL engine for AWS, only works with AWS, cannot run locally, 5X faster than My SQL, 1/10 cost of commercial DB due to read replicas being automatically enabled
* Aurora scaling has down time, but only takes several minutes storage scales in 10 GB incremental, computer scales up to 32 CPUs and 244GB of memory
* Aurora maintains 2 copies in each AZ and in at least 3 AZ, so 6 copies minimum; 15 additional Aurora replica and 5 Additional MySQL read replica is possible but cannot be failed over to.
* Aurora is self-healing, by constantly scanning for error and fix it; Not supported by all regions at the moment.
* When Aurora is provisioned, you can assign priority to each instance, it's like rank in army, when a higher tier instance is down, it will fall over to the highest tier instance working

When the first Aurora is provisioned, it creates a whole cluster and the cluster name is used to generate the endpoint of the entire DB. Always use Aurora cluster endpoint, instead of instance endpoint, unless specific cases.

* Aurora automatic copies are only for recovery and performance (physical media only), the virtual machine running the instance is still a single point of failure, unless using "create an Aurora replica" option
* Aurora replicas can only be used for read but not write, write has to go through the primary instance. In the case of failover, endpoint DNS will remain the same and the new instance will be able to write
* In case of deleting a cluster, replicas must be deleted first before primary instance and the whole cluster can be deleted
* Aurora is designed to transparently handle to loss of 2 copies without affecting the write availability and loss of 3 copies without affecting the read availability

Kenises: fully managed service for bring in streaming data at massive scale to the cloud (IoT, social media, etc), and then become available for processing within a second (for redshift/EMR).

**VPC:**

Isolated cloud resources, can even run the whole thing on dedicated hardware

Should be able to build one from memory, just think about it as a virtual data centre in the cloud

1 subnet = 1 AZ (1 subnet completely sits in one AZ, but one AZ can have more than one subnet)

Best practice: one Public and one private subnet per AZ and replicate in multiple AZ for failover. Worst practice: public in one AZ and private in a different AZ, then any AZ fail the system fail.

When AWS account is setup, AWS create a VPC for each region (then you create additional logical VPC, and you have complete control within the VPC)

Use case: public facing subnet with internet access and private facing subnet without internet access

VPC can be used as an extension of an existing physical datacentre, as an extension or as the cloud services of an edge computing datacentre.

Two ways to access a VPC: via internet gateway or via Virtual Private gateway and VPN, then go through router and route tables to each subnet, filtered by ACL (Ip address blocks)

Private subnet must be accessed via (bastion host) jump box or VPN, internal address options: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16. Once this is set up, you cannot change the address block

When internal address is not enough, you have to create another VPC, so you can have another set of internal addresses

By default, number of VPC per region is soft capped at 5.

CIDR.xyz to calculate, whatever number after slash is the bits that's untouchable, and for those addresses available, you always loose the first four and last one

You may want to name your subnet with CIDR address range and AZ just to keep track

The following can be done for VPC:

* Launch instances into a Subnet
* Assign custom IP address range for each Subnet
* Configure route tables between subnets, sort of like level 4 security, much better control over resources using subnet ACL
* Create internet gateway and attach to VPC so it is accessible from public, but you can only create one, and it is normally not a performance bottleneck or single point of failure concern

When we setup AWS account we have a default VPC, all subnets are internet accessible by default for convenience and user friendliness. No private VPC available, and all resource (say EC2) will be public be default.

Public VPC has both private and public IP addresses, private VPC only have private IP address

VPC peering: connect VPC together via direct network route, and they behave as they are on the same private Network, this can be done with VPCs from other AWS account, can cross region. Always have a centre VPC, no transitive peering allowed

(Again!!) Security Group does not specify inbound or outbound, access control list has inbound and outbound managed separately(stateful vs stateless)

When a VPC is created, it creates ACL (allow everything), security group (VPC specific), and a route table (which talks to all subnets and enable all subnets to talk to each other), but not subnet or internet gateway

Upon creation, next step is to provision subnet, then internet gateway, and another route table for public access. you don't want your default route table to be public.

Upon creating a subnet, it will be associated with the default route table by default, therefore you don’t want your default route table to have a way out to internet. For public subnet, create another route table also. When route table is created, associate it with your public subnet and allow auto-assign public address to allow public access (Or auto-assign can be done when provision EC2, at a resource level)

AZ name means different AZ for different users, so people will use them evenly, for each cider address specified, the first four and the last one is not available, they are reserved for:

* .0 Network address
* .1 Reserved by AWS VPC router
* .2 .3 reserved for future use
* .255 broadcast address, not supported so reserved

Without creating an Internet Gateway and attaching it to a VPC, the VPC will remain private and not accessible. You cannot attach multiple internet gateway to one VPC

Attaching a security group to private subnet to only allow access from public subnet's CIDR address. However, this only allows data inbound, if you run yum update it will fail as it doesn't have internet access

When you have your own VPC, it's like you have your own region and AZ when provisioning EC2

NAT (network address translation) instance is on the way out, NAT gateway is replacing it, but NAT instance is still being tested

NAT instance is a specially configured EC2, basically just passing on stuff, must disable source/target check as it will be neither source nor target of any traffic (EC2 perform source/target check by default).

Add a route direct all the traffic to public to the NAT instance, inbound will be handled by public subnet and the private subnet’s security group, only outbound will be handled by NAT instance -> IGW

In this configuration, the NAT will be a single point of failure. Its network throughput and computational power are all bottlenecks.

NAT gateway will only handle ipv4, egress only internet gateways will handle ipv6 but not required

Similarly, configure route table to make all traffic out to the internet pointing to the NAT gateway, AWS will manage patch, auto scaling, no need to associate with a security group. However, NAT gateway is still one AZ, cannot span multi AZ

Only advantage of NAT instance is that NAT instances can be used as bastion servers (jump box, used to SSH private instance) and performs functionalities like filtering and security tasks, although this introduces more risk.

ACL: can only be applied in one VPC, cannot span VPCs, like security group, so the default one from you default VPC won’t work. By default, a newly created ACL denies everything (only the one created with VPC allows everything). ACL rules are suggestion to be named with numbers in the incremental of 100 (this way you have a lot of room to change mind), since rules are evaluated in incremental order.

ACL sits before security group so if traffic is blocked by ACL, it won't reach security group

One VPC must have an ACL, if not specified, it will use the Default.

Subnet in VPC must be associated with an ACL, one ACL can be applied to multiple subnet, but one subnet can only have one ACL. When new one is applied, old one is removed.

Security group is stateful, when inbound is configured, outbound came into effect straight away. NACL is stateless, you can white or black list them separately

Flowlog: monitor information flow for VPC, log can be exported as file to S3, or streamed to lambda for dynamic response. Has a role to talk to CloudWatch, can be set up to monitor certain things

Flowlog does not log the VPC that's peered with your VPC, unless it's under the same account, Cloud watch only work in one account. Some other exceptions won't be monitored as well, mostly tooling connection are not monitored. Logs cannot be tagged. You cannot change the configuration after creation, cannot change IAM role.

Can dump log into log flow in CloudWatch, you need to create Flowlog first

You can use VPC endpoint to access AWS resources within the same region, through private IP address, without having to access public internet

To clean up a VPC, delete all resources under it first (there's dependencies, order EC2 -> endpoint/gateways -> VPC)

AWS private link connects AWS partners' service to your VPC via AWS network, instead of public internet

Direct Connect:

* connect your datacentre, office, premise to AWS, reduce network cost, increase bandwidth throughput.
* Not a VPN, but a dedicated connect to VPC which bypasses internet. VPN can be a good option too when you need it instantly, do not need a lot of bandwidth and can tolerant internet failure.
* Dedicated line is provided by a Telcon to AWS direct connect facility, then AWS handle the routing and fibre connection to AWS data centre
* Available in 10 or 1 Gbps, lower than 1 Gbps are available through AWS direct connect partners. Uses Ethernet VLAN trucking 802.1Q
* Can take several months to setup

**Application services:**

SQS (Simple queue service), first ever AWS service, worth reading FAQs

distributed queue system, temporary repository for messages that awaiting processing, great way to decoupling components

Can be set to autoscaling, can be consumed by server, lambda, EC2 etc. It is pool based so it doesn't push. it supports 256KB data of any format, including JSON, lifecycle 1 min to 14 days, default 4 days

Support standard or FIFO, highly reliably and available, order is most likely maintained but not guaranteed in standard, FIFO has guaranteed ordering, limited 300 transaction per seconds

When pulled, message will become invisible until computation is done, but will only be deleted when processing is over, so in case of failure, it can be pulled again

visibility time out has max limit of 12 hours, and default is 30 seconds. when processing is longer than timeout, it could be processed by another server because the first server did not return with confirmation. Say if a task takes minutes, set the invisible time to 10 minutes is more reasonable than using the default 30 seconds

short pulling: pull all the time and respond instantly if there's nothing. Long pulling pulls periodically and only return when there is something or time out. Long pulling can save cost, but when there’s always enough task, or you want instance response when there’s no job, short pooling make sense.

You can add your own mechanism to handle duplication (or any other application level tracking)

Delay Seconds: when a message arrives, make it invisible for a few seconds (What’s the use case??)

SWF (Simple workflow service)

* Compare to SQS, SWF manages workflows that could be code, script or human actions, SQS message oriented
* 3 actors: starter, decider, worker
* SWF decider can monitor the workflow and assigns tasks to workers according to decider's decision. tasks are assigned once and once only (Main difference from SQS)
* Maximum workflow period can be 1 years measured in seconds

SNS (Simple notification service), includes SES (simple email service)

* publish-subscribe paradigm, notifications are pushed to subscribers, if needed it can pool (pull) notifications use SQS
* Can deliver to IOS, google, fire OS, using Baidu message service in China. Can also send SMS or email to SQS or any http(s) endpoint
* Subscribers subscribes to "topics", one topic can support deliveries to multiple endpoint types. e.g. group IOS, android and regular users and regular users and send them different messages (smartphone gets push notifications and normal gets text)
* Pay as you go, prices vary according to delivering method
* web based point-and-click interface
* data format is JSON

Elastic transcoder: convert media file format. Provide pre-sets for popular formats so don't need to guess, pay based on the minutes and resolution. Use case: source video -> lambda -> transcoder -> S3 for web consumption

API gateway: Routing API calls to services (lambda or EC2 server).

Can use API caching to speed up API response speed by caching response for a TTL instead of letting it run every time, scales automatically, however you want to throttle to prevent over flushing. To do this you can connect to cloud watch to log all request

When resources are not accessible because of same origin policy, active cross origin resource sharing (CORS)

Kinesis: data streaming service (in analytics session), you send your continuous data transfer in small sizes (KBs) to kinesis. Producers send data to Kinesis for 24 hrs - 7days, data stored in shards, then consumers consume data from shards, and send results to other AWS services.

Maximum 5 transaction per second, 2M max for read and 1000 transaction per second, 1M max for write

Kinesis firehose: send stream to firehose and shoot to S3 then waiting to be picked up by lambda or some other pulling mechanism, doesn't have shards, don't concern consumers. For something more require more of a warehousing fashion.

Kinesis Analytics: Allow SQL data in kinesis, and send processed data out

Real World examples: come back later

Workspaces: Remote computer (VDI), a bundle of computing storage and software resources

It runs Windows 7 experience provided by windows serve 2008, by default you get local admin access. workspace is persistent and data on D drive is automatically backed up every 12 hours, and do not need an AWS account to access

**White papers:**

Zero up-front, just-in-time, efficient utilization, pay as you go, reduce time to market (especially when multi-internationalising). Infrastructure as code, auto/proactive scaling

Best practices: understand fully for case studies

General design principles:

* Stop guessing capacity need, be ready to cater capacity change with design
* Test at production scales, it's cheap and on-demand anyways
* Automate to make experimentation easier (e.g. Use script to repeat process, automate testing, etc)
* Allow for evolutionary architecture (consider architecture change with design, and automated testing makes architecture change less risky)
* Data-driven architecture fact-based decisions (even through automation), so it evolves automatically
* Improve through game days: simulate production event and improve on the fly

**Security pillar (First concern of everyone move to the cloud, DD's specialty)**

Design principles

* Apply security at all layers and components
* Enable traceability
* Automate response to security events (from messaging someone, to automatic responses)
* Focus on your share of responsibility, don’t worry about AWS
* Focuses on your system based on responsibility model (AWS actually removed some of your responsibility, like guarding the server, patching OS and software. Whatever "of" the cloud, not in)
* Automate best practices, use hardened images (removed unnecessary program, ports, access, account, service, etc) so it is deployed with protection every time.

Definition

* Data protection:
  + classify it into audience groups, and implement privilege access system, and encrypt where possible, both at rest and in transit
  + AWS support this by IAM allowing easy encryption and key rotation, detailed logging (CloudTrail), allow versioning
  + How are you Encrypting data at rest? (denied unauthorised access, encrypted with ELB, EBS, S3 and RDS)
  + How are you Encrypting data in transit? (SSH, HTTPS)
* Privilege management:
  + ACLs, Role based access control, password rules (strength, rotation, etc.)
  + How do you protect your root credentials? (MFA? HSM?)
  + How are roles defined for users and system resources?
  + How are you limiting automated access? (from applications/scripts)
  + How are you managing keys and credentials?
* Infrastructure protection: AWS handles most of the things, here it means VPC low level security concerns:
  + How are you enforcing network and host level boundary protection (Security group only? ACL also? EC2 in public or private subnet? Jump box used?)
  + How are you enforcing AWS service level protection (Users/group with minimum access only?)
  + How are you protecting the integrity of you EC2 (say you're running windows; do you have anti-virus?)
* Detective control
  + CloudTrail, CloudWatch, AWS Config, S3, Glacier
  + How are you logging AWS logs (CloudTrail in each region? Log management from third party?)
* Best practices:
  + Key AWS Services: See above
  + Extra resources: AWS security whitepaper

**Reliability pillar**

Design principles

* Test recovery procedures (not just test it works, also test it recovers from failure, Netflix tools are available for that)
* Automatic recover for failure (monitor KPI, even anticipate and prevent risk)
* Horizontally not vertically (scaling out, replace huge resource with smaller resource so they don't share common point of failure, trade a bit performance for total failure)
* Stop guessing capacity (avoid over/under provision, build it in design)

Definition

* Foundation
  + Like the foundation of house, AWS handles most of this for you, it's designed to be limitless, with a service limit for each resource to prevent you over provision, unless you raise tickets
  + How are you managing AWS service limits? (Is there anyone in charge??)
  + Are you planning your network topology? (Is there a single point of failure?)
  + How do you deal technical issues? (Do you have an account manager? AWS specialist?)
* Change management
  + Change management are automated and trackable with cloud-formation
  + How does it adapt to changes when in demand?
  + How are you monitoring the system to know a change is needed? (CloudWatch?)
  + how are you executing change management? (Process?)
* Failure management
  + Always assume failure will occur, and always consider why they occur and how to prevent them
  + How are you backing up data? (S3 multi AZ? Cross region?)
  + How does your system withstand component failure? (redundancy, remove single point of failure)
  + are you planning for recovery? (ELB, auto scale, Failover)
* Best practices
  + IAM, VPC, CloudTrail for change management, CloudFormation for recovery

**Performance efficiency pillar**

Design principles

* Always checking if you're using the best service available
* Democratize technology: Make tech accessible for non-IT specialist, package it into a push-button service. This is what AWS does, and AWS architect should aim for this too.
* consider ability to go global, when serving customers far away, you should consider launch the service in another location
* Use serverless architecture (Only cost money when someone using a service. Otherwise even virtual server cost money)

Definition

* Compute (Use the right instance, switch instance size and type quickly, however stopping instance is still required)
  + How do you choose the most appropriate instance types? And continue to ensure that you have the most appropriate instance type as AWS introduces more and more?
  + How do you monitor the instances post launch to ensure performance?
  + How do you ensure that the quantity of your instance just matches demand?
* Storage (consider: Content type block or file, pattern of access random or sequential, throughput, update/Access frequency, availability/durability requirement)
  + How do you select the correct storage service for your system? And continue to ensure that you have the most appropriate solution as AWS introduces more?
  + How do you monitor your storage solution to ensure performance?
  + How do you ensure that the capacity and throughput of your storage solution matches demand?
* Database (Consider what feature do you need: availability? No-SQL?)
  + How do you select the correct DB for your system? And continue to ensure that you have the most appropriate DB as AWS introduces more?
  + How do you monitor your DB to ensure performance?
  + How do you ensure that the capacity and throughput of your DB matches demand?
  + Time Space trade-off (Same concerns as all the aspects above, just regarding caching)
* Key AWS Services: Autoscaling, storage (EBS, S3, glacier) and DBs, ElastiCaching and CloudFront, direct connect, RDS read replicas

Cost Optimization (Pay the lowest price possible and still achieve the objectives)

* Design principles
  + transparently attributing expenditure (tagging to know who spent on what)
  + Use managed service to reduce the cost of ownership
  + Trade capital expense with operation expense
  + Utilize economics of scale
  + Stop spending on data centres (However maybe there's advantages using edge computing and legacy system)
* Definition
  + Match supply and demand
  + Don't over or under provision, use auto scale and CloudWatch, or use lambda so supply always meet demands
  + How you make sure your capacity is just enough, and still can handle demand changes
  + are you optimising scaling?
* Cost effective resource
  + For example you can run the same task with 7 hours of T2.micro, or 15 minutes of m4.2xlarge, m4 would be cheaper
  + Have you selected the right resource type? Have you selected the right pricing model?
  + Are they managed services? it makes sense to outsource things to managed service.
* Expenditure awareness
  + longer need quote from difference vendors, but could cost if you forget to turn off or scale down
  + How do you monitor? How do you decommission resource that you no longer need?
  + Factor in data transfer charges when designing architecture
* Optimizing over time
  + keep track of the changes made to AWS, constantly re-evaluate your architecture, subscribe to AWS blog
  + Use trusted advisor so that you get notified when there are optimisations available
* Key AWS Services: Autoscaling, EC2 reserved instance, AWS trusted advisor, CloudWatch alarm, SNS, AWS blog

Operational Excellence (2016 new pillar: responses and execution operations should be automated and documented, tested and reviewed)

* Design principles
  + Perform operations with code
  + Align operations with business objectives
  + Make regular, small, incremental changes, like software development
  + Test for responses for unexpected events
  + from operation events and failures
  + keep operations procedures up-to-date
* Definition
  + Preparation
    - workloads should have runbook (checklist for daily tasks) and playbook (response plan, escalation path and stakeholder notification)
    - CloudFormation, reduce the opportunities for human error
    - Tool level use auto scaling, AWS config, and tagging properly
    - What best practices are you using? How are you managing configurations?
* Operation: Consider business continuity
  + Should be standardised, easy to track, audit and rollback.
  + Should not require downtime and manual execution.
  + A larger of logs and metrics should be collected and reviewed to ensure continuous operations.
  + Consider CI/CD pipeline
  + How are you evolving while minimising the impact of change?
  + How do you monitor your workload to ensure it is operation as expected?
  + Don't rely on manual procedures
  + Responses should be automated too, avoid escalations, should have a comprehensive playbook. In the case of escalation is necessary, it should result in stakeholder notifications
  + How do your response unplanned event, how to escalate in an un-planned event?
* Key AWS Services:
  + AWS config has a detailed inventory of AWS resources, service Catalog, design automation using SQS and other services
  + Use AWS developer code tools, and cloud trail
  + Use CloudWatch -> SNS

read best practices and well Architected

* Always decoupling components and try to remove tight dependencies all the time
* Always implement elasticity, by proactive, event-based, or demand based(monitor)
* consider security, minimise access to necessary

**Exam tips:**

* Big data, social media data gathering/streaming – Kinesis
* Big data processing - Elastic map reduce
* BI – Redshift
* EBS backed vs instance store for volume: EBS is persistent, can be detached and re-attached, won't lose data when stopped. instance store is ephemeral, lost when terminated
* OpsWork - chef uses a recipe to maintain a consistent state, look for keyword chef, recipe or cookbook, sysops admin topic
* Get public IP of Ec2: use curl to fetch metadata <http://169.254.169.254/latest/meta-data>
* Resource group & tagging: metadata written in JSON, automatically created resource will inherit tags

**TEMP DUMP**

CloudWatch vs CloudTrail: CloudWatch is for monitoring, CloudTrail is for auditing. CloudTrail operates per region, uses S3 to store log

Consolidated billing: Root -> organization Unit -> AWS account, saves cost by utilizing scale. Always use top level security and use it for billing only. 20 account max but can add

Consolidated billing can have alarm on all or individual account, create account (use model ID of existing account) and add policy too (denied or allow uses of services)

VPC peering: comeback when VPC knowledge is solid

cross region resource sharing

Resource group

AWS account soft caps

AWS config

Elastic IP

instance store

bastion host

What happens to EBS if instance reboots

Security white paper/FAQs

Data centre

RDB snapshot and backup, redshift

Elastic beanstalk/lightSail

ECS/Docker container/hypervisor

EC2 placement group does not span VPC

EFS/AWS glue/Data pipeline/ S3 bucket policy. S3 availability and durability chart. cross-region replication rules, S3 Request Headers

elastic cache

Logging/CloudWatch/CloudTrail/AWS config/OpsWorks/Pricing/ AWS managed services/ Spot instance/ Consolidated billing

Flow Log (Instance level?)

VPC

CloudFront: what happens if direct origin access is disabled and TTL expired

VPN connections, Virtual private gateway

CloudFormation catagory

ELB header/autoscaling Health check grace period(lab)

EC2/ hypervisor/

Firewalls and securities

networking (netmask??)

Use cases

CIDER range???

federation proxy or identity provider?

Route table

Encryption

Application Load Balancer

CloudFormation valid sections: 9 of them

Storage Gateway

Active Directory Integration: Comeback later

Security token service: Comeback again

VMDK???