**Color Code: From SAA, New**

**General Information:**

In AWS CLI, help shows all command options

Terms, limits, quirks, comparisons.

Focus:

**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

**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, on top of A, AAAA, C name, etc.

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

When register for a domain, it starts with a hosted zone. A Route53 domain has the same domain name registered under multiple top-level domain names (as its internal NS) to prevent outage.

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

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).

**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

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 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

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???

HTTP error code, 3XX means redirection, 4XX means client side error (think about 404 when no connect), 5XX means server side error, 504 means gateway time out, 429 means too many request

SAML 2.0 federation

Nginx, Passenger and IIS