Whizlabs & BrainCert for exam

Read the **white papers** mentioned in exam roadmap Re: Invent videos.

**Monitoring & Reporting**

**CloudWatch (important to understand what is monitored in different resources):**

Monitors a variety of services (compute, storage & CDN, DB and analytics, others like SQS, SNS, AWS bill, etc.), mostly around performance. by default, it monitors Host Level Metrics including:

* CPU utilization
* Network throughput
* Disk (IO only, not storage usage as it is not strictly performance)
* Status Check

And Everything else is a custom metric, like RAM utilisation (which again, is not strictly performance).

You can use GetMetricStatistics API or using third party tools offered by AWS partners to retrieve data.

Log data is stored indefinitely by default, but you can change the retention for each log group anytime, e.g. 6 months for EC2 but 2 weeks for ELB. Log remains even when resources are terminated.

Default metrics for many services are 1 minute (ELB for example is 1 minute) but can be 3 or 5 mins depending on the service. Minimum granularity is 1 minute (may be updated, **refer to whitepapers**). When default setting is on say for EC2, you only get 5 mins data points, even you push them out every 1 minute.

Custom High-resolution metric can report per second now, however minimum granularity provided by AWS is still 1 minute.

Alarm can alarm you when a state is reached, or even take actions (typically through Lambda) like decommission resources.

Can be used for on premises infrastructure too, just install SSM (systems manager) agent and CloudWatch agent and send metric to CloudWatch. This way you can have all your infrastructure monitored in one place.

To monitor EC2 with a custom metric, you need to give that instance a role to write data to CloudWatch (doesn’t have to be CloudWatch full access, in fact it’s against least previlage rules. But it’s a good baseline to start with). It can take up to your granularity interval for new custom metric to take effect (came into effect from next cycle).

Custom metrics appear under “custom namespace”, instead of AWS namespace, where most default AWS resources metrics appear.

EC2 uses Cron to trigger aws-script to shoot out data to CloudWatch, Cron is a general-purpose tool for periodic work in Unix-like system. Default dir/etc/Crontab

CloudWatch event: CloudWatch can also use Cron to schedule task to trigger other tasks like Lambda

Create a billing Alarm: need to verify email to receive alarm, using SNS. Billing and Alarm data can only be accessed from us-east-1

**Monitoring EBS:**

* Remember the 4 types of EBS from SA course, maximum volume 16T, 3IOPS/GiB, don’t have to remember numbers, use cases are more important:
  + GP2 for dev and test, up to 16000 IOPS, 160 MiB/s, min 1G
  + IO1 for MongoDB, Cassandra, etc. 50IOPS/GiB, 10000+ to 64000 IOPS, 320 MiB/s, min 4G
  + ST1 for DWH, Big Data Log, etc, cannot boot, 500 IOPS, 500 MiB/s, min 500G
  + SC1 for cheapest storage, cannot boot, 250 IOPS, 250 MiB/s, min 500G
* Bursting IOPS is possible but takes Burst credit. Burst only takes you to 3000 IOPS but increasing volume can take you to 10000 IOPS (@5334GB volume in the case of GP1). Each volume get’s credit of 5,400,000 IOPS credit initially, which is good enough for 30 mins (5400000/3000 = 1800 s). When you’re not using your provisioned IOPS, you’re earning credit.
* IOPS are counted in 256k blocks.
* EBS used to need pre-warm, but not anymore. New volume restored from snapshot, need to be read once (called initialisation) to get full performance (otherwise only initial access is slow).
* Possible metrics:
  + VolumeRead/WriteBytes
  + VolumeRead/WriteOps (IOPS per monitor interval, 1mins/5mins)
  + VolumeTotalRead/WriteTime
  + VolumeIdleTime
  + VolumeQueueLength (if above 0 means you don’t have enough IOPS). Solution:
    - When IOPS max is not met, increase volume
    - When IOPS ceiling is reached, change from gp2 to io1
  + VolumeThroughputPercentage
  + VolumeConsumeReadWriteIOPS
* Possible Status Check (Remember the meaning of warning/impaired)
  + Ok
  + warning (degraded/severely degraded, no getting expected IOPS)
  + impaired (stalled/not available)
  + insufficient-data
* When working with the last generation EC2, you can change volume size (via command line or console, and then have to go into the machine to adjust the file system though), volume type and IOPS performance without detaching it

**Monitoring ELB** (4 ways to monitor different things, investigate open systems interconnection aka OSI model of networks, 7 layers):

* Remember the 3 types of load-balancer, NLB has static IP (one per subnet only) and operate on TCP
* Four ways to monitor: CloudWatch, Access log, Request tracing (ALB only), CloudTrail log
* CloudWatch vs CloudTrail: performance vs AWS internal API calls (actions).
* ELB publishes data points of ELB and targets (EC2s for example, HTTP 4XX, 5XX count, etc.) to CloudWatch, metrics are simply these data ordered in time (think about a value to be monitored).
* When an ELB is provisioned, 28 metrics will be activated automatically, under “ELB” namespace in CloudWatch (may take 5 minutes to populate), or under “monitoring” tab
* Access log records traffic behaviours (CloudWatch is more about performance in stats) and stats, information stored in S3, used to analyse traffic pattern and troubleshoot issues. It’s an ELB specific feature and is disabled by default. Access logs in S3 does not get deleted like CloudWatch unless you do it. (Say you want to investigate instance that’s terminated due to scale-down).
* Access log can be huge, so it’s better utilized inside of Athena, or third-party log analysing tools like sumo logic.
* Request Tracing: For ALB only, tracing HTTP request. Client request got a X-Amzn-Trace-Id added, and got updated along the way by ELB, target services etc, so can be used for tracing once requests got into and behind your ELB.

Monitoring ElastiCache (Hard!!!):

* Two engines: Memcached/Redis (revision Dev courses)
* Four things: CPU Utilisation, Swap Usage, Evictions, Concurrent Connections
* Memcached: Multi-thread, always add more node if CPU load > 90%, compatible with existing Memcached implementations, no Multi-AZ
* Redis: No Multi-thread, scale when load on a CPU exceed 90/core number (because it’s single thread, you need to deal with the scenario where all tasks are on one core), Open source, kay-value pair, support master/slave replication for multi-AZ.
* Swap Usage: Swap file is the space on disk reserved when RAM is not enough (virtual RAM). Typically, it’s the same size as your Ram. Having to use swap page is something need attention. For Memcached should be around 0 and not exceed 50MB, if exceed you should increase Memcached\_connections\_overhead parameter, which allows more reserved memories for Memcached connections for connections, refer to whitepaper. Redis has no Swap Usage, instead use reserved memory.
* Evictions: evicting data when new data comes, and no memory is available. Memcached can scale up and out (because it can multi-thread, so scale up also works), Redis can only scale out (only more nodes can handle more connections).
* There’s no recommended setting for handling current connections, when overwhelmed can be large traffic or application not releasing connections correctly (set concurrent connections metric to monitor this).

Metrics from multiple region and Custom Dashboards

* Start with creating a new dashboard, and then add widgets.
* Dashboards are global, you can see all resources in one place if you want to and it doesn’t matter which region you switch to. However, to add a widget you should be in the right region.
* Always remember to save!!!
* It makes sense to create different dashboard for different purposes, dev/test/prod, or Server/view/controller, etc.

Billing alarm:

* Enable from CloudWatch and set a threshold for the entire AWS account. Link an email address (or a list of email addresses) to receive billing alert.

AWS Organizations (important!):

* Manage multiple AWS account, it’s not in the service list, as it manages all services in some sense. It can do 4 things:
  + Centrally Manage Policies, you can group AWS accounts into departments, and attach policies all together
  + Control Access to AWS services. Much like IAM you can create Service Control Policies (SCPs) to allow or deny individual AWS services. Even IAM of that account allows it, SCP override it (from a API level).
  + Automate AWS account creation and management, instead of having to create account, and then apply individual policies, you can create an account, and apply a SCP to handle all access.
  + Consolidate Billing Across Multiple AWS Accounts (Take advantages of pricing benefits, only organization feature that can be used as a standalone, in “consolidated billing only” mode)
* Upon creation, you should either invite an account or create an account (make sense)
* Added accounts can then be allocated to OU “organizational units” (for example, Developers, HR, Marketing).
* Policies are managed separately and can be then attached to account/organization units. Policy can be black or white listing.

Tagging & Resource Group

* Tags are Key-Value pair metadata, can be inherited when resources are automatically provisioned.
* Resource Group is just group share one or more tags. (Since tags are normally added based on resource feature, groups can reflect that too)
* Two types of resource group: Classic Resource Group (can be global or regional) VS AWS Systems Manager (per region, but programmable, can be the basis to execute automation or review insights). Classic Resource Group is more of a global inventory check.
* Resource groups are not organized in per services basis, i.e. the same tag can be given to both EC2 and DynamoDB and belong to the same resource group
* Resource groups are usually created based on existing tags. Once group is created, you can add additional group tag to the whole group
* You can execute commands on system manager resource groups, like do something on all north Virginia TEST instances.
* Tags are case sensitive. You should basically tag everything.

EC2 pricing:

* Reserve instances can be scheduled (per week, month, etc.) or standard, can be convertible (size and type can change)
* Dedicated host can be on-demand or reserved as well

AWS config (comeback for revision, too much content, check FAQ):

* Fully managed service provides a list of resource inventory and configuration history, and configuration notifications for security and governance. Consider “recording what’s going on in AWS account”.
* Enable: Compliance auditing (e.g. if EC2 left port 22 open, this can be found automatically), security analysis, resource tracking, automated compliance checking, configuration snapshots
* Compliance checks can be triggered periodically, or upon configuration change.
* Rules must be deployed on a per region basis!!! for now
* Managed rules are fundamental, like “check if all resources are not publicly accessible”.
* Key component: Config dashboard (tells how many resources/rules are non-compliant), Config Rules (including AWS managed rules or Custom rules, both via lambda, sometimes periodic sometimes event triggered), Resources, Settings
* None like in CloudWatch, Lambda in AWS config only tells you weather or not a rule is broken, cannot take any further actions.
* For each resource, configurations can be review in a timeline (Not API call so cloud trail won’t log that, Cloud Trail normally only work on provision and deletion)
* Stores everything in S3, trigger Lambda (based on standard or custom rule) for reaction (SNS).
* Provide minimum access to the resource’s config users monitor, most of the time read only is enough. It also need write access to S3 to write log, and publish access to SNS, to send out notification. ideally only sysadmin should have full access, other people should have read only.
* Terms:
  + Configuration Items: point in time attributes of resources, what’s being monitored
  + Configuration Snapshots: Collection of Config items (summary load)
  + Configuration Stream: Stream of change (continuous load, for system that has configuration changes all the time you don’t want that sent to SNS all the time. Like how CloudWatch data points in a time line forms a stream)
  + Configuration History
  + Configuration Recorder
* Config is regional, must be activated for each region separately
  + Can select to monitor all resource or a group of resource.
  + Can setup S3 and SNS up on activation, needs a role also read only access to resources under monitor and write access to S3.
  + Next is to apply config rule, like “check if SSH is open”, then you can see if all resources are compliant
  + AWS config will report the number of resource non-compliant, and you can then go to “manage resource” to fix it from the console
* You’re able to see all config changes in a timeline manner, for each resource.

AWS Config VS AWS CloudTrail VS AWS CloudWatch

* Compliance (configuration setup) VS API call history VS Performance
* Trusted advisor and inspector can be mixed in the comparison too

Health Dashboards (reports on a per-region basis, however event log which reports what has been wrong with AWS recently, is global)

* Service health dashboard (status.amazon.aws.com) vs personal health dashboard. If a particular AWS service is working fine VS if a particular AWS service you use is working fine. If an AWS service has an outage and does not affect you, it won’t show in personal health dashboard.
* Service health check provide health check history as well, except during the last infamous S3 major outage that brought down the whole service health dashboard
* Personal dashboard will be empty if there’s nothing wrong, accessed from the service dashboard pages

**Deployment & Provisioning**

Deploy EC2: Use an AMI

* You may want AMI that’s hardened, or have packages pre-installed, etc.
* Spot instance use “persistent request” to keep requesting instances, whenever the spot price drops below bid price. Otherwise it will be provisioned only once. Persistent request can have a valid period as well.
* When spot instance is interrupted, you may have the option for terminate/stop/hibernation (keep the RAM, may not be available due to region and other reasons)
* Launch group option: all instances launched at once, or none gets launched at all.
* Placement group: instances in single AZ, trade durability for low latency
* Not tested, T2 now have the option to burst CPU capacity, by checking “T2 unlimited”
* You can enable “termination protection” to prevent instances from being accidentally terminated
* The EBS created with a new EC2 instance has an automatic snapshot in S3, and delete on termination is select by default as well

EC2 launch issues

* InstanceLimitExceeded error, reached the max number of EC2 instances allowed for a region, by default is 20, but can request on per-region basis. Possible scenario could be you have 17 already and try to launch another 5, regardless if they’re in a launch group or not, it will report this error. None will be launched in this case as it will fail during API call, not during the 4th instance launching.
* InsufficientInstanceCapacity error, AWS doesn’t have any more available capacity, doesn’t happen very often. Solution:
  + Wait a few minutes (supply/demand may change)
  + Try fewer instances (if you’re launching a lot at a time)
  + Select a different instance type (It might be a shortage in one type of hardware)
  + Try purchase reserved instances (work around it)
  + Do not specify AZ and it might work (it might be a shortage in one AZ)

Bastion host (Jumpbox, check VPC sector in SA course):

* A host in public subnet (has a route out to the internet), used to access private subnet. So, you SSH into bastion host, and SSH private EC2 via bastion host. This keeps private EC2 instances in private without internet access. (Out to internet you need a NAT instance/gateway and configure the destination to be 0.0.0.0/0, target is your NAT instance/gateway name)
* The access of a bastion host should be locked down as much as possible, this includes who can access, what IP range can access, Only SSH ports are open, etc. This can be ensured by AWS config
* Public IP for EC2s can only be setup upon creation, therefore if you launched an EC2, intended it to be public and forget to assign it with a public IP, you can only create an elastic IP and assign to the EC2.

ELB:

* ALB: Content based routing based on http/https header and other information (called application-aware). E.g. send sale request to a set of EC2s and marking to another set of EC2s. Consider Tesla send all traffic from Model X to one server and traffic from Model S to another server
* NLB: Superfast operating at transport layer (layer 4), for prod and latency sensitive load, cost the most
* Classica LB, still come up in exam, operate in both layer 4 (TCP) and 7 (HTTP/HTTPS), layer 7 functionality is very basic. Layer 7 allows IP based (X\_Forwarded header) sticky sessions, can be restricted to layer 4, which is purely TCP based routing.
* ELB scales automatically, however sudden traffic can overwhelm an ELB and if that’s possible you want to contact AWS to pre-warm it (only thing need pre-warm), so it has the appropriate level of capacity to handle the traffic that you expect. AWS need start and end date, expected rate/second, total size of a typica request.
* ALB changes IP all the time to keep with the demand, when a static IP or Amazon elastic IP (consider them the same thing) is required you can only use NLB (one per subnet). However, to get the best of both worlds, you can set an ALB behind an NLB.

ELB error messages (remember as much as possible):

* 4XX and 5XX, successful is 200, 4XX is client-side error (think about when you don’t have internet connection you get 404), 5XX is server-side error (think about 5 looks like S for server).
* 4XX can be:
  + 400 – Bad header, not comply with http standard
  + 401 – Unauthorized
  + 403 – Forbidden by firewall
  + 460 – Client closed connection due to timeout, client timeout too short
  + 463 – X-Forwarded-For header has more than 30 IPs
* 5XX can be:
  + 500 – internal server error, could be ELB configuration error
  + 502 – bad gateway, server closed the connection
  + 503 – no target (EC2) to send to
  + 504 – gateway timeout, application not responding, could be server, application, or DB
  + 561 – unauthorized server authenticates with ID provider (say Auth2) failed.
* For 4XX, inspect client, fix and retry; For 5XX, inspect server/ELB and retry

ELB CloudWatch metrics (You might want a SNS for SurgeQueueLength or SpilloverCount)

* ELB publishes metrics for itself as well as the instances in the back. Can monitor general health condition or performance. Metrics are by default gathered per minutes. Possible error code:
  + BackendConnectionErrors: number of connection errors to the backend instances
  + HealthyHostCount: number of healthy hosts registered
  + UnHealthyHostCount: obviously
  + HTTPCode\_Backend\_2XX/3XX/4XX/5XX, this is the count of each error code
  + Latency: time for instance to respond
  + RequestCount: number of requested processed in an interval (1 or 5 minutes)
  + SurgeQueueLength: number of pending requests, max size 1024, additional will be rejected, CLB only metric.
  + SpilloverCount: number of requests rejected due to SurgeQueue overflow, CLB only

Setting up ELB with monitoring:

* Make sure when EC2 is launched, auto-assign IP address is selected
* The Load Balancers direct traffic to “targets” (can be instance or IP), and health check can check if a path returns 200.
* The process of attaching an instance to an ALB/ELB is called register.
* Once all setup, use the DNS address to access ELB, and the servers behind it

Systems Manager (SSM)

* One central place to provide visibility and control over your AWS infrastructure, and you can inspect on a per resource group basis as well.
* Integrates with CloudWatch/CloudTrail/Config/Personal Health Dashboard/Trusted advisor, all appears as a page in SSM so you can see them in one place
* Run Command (On EC2 instances) for operational automation like (this can be AWS managed commands or custom script):
  + Stop, Start, re-size, terminate EC2s
  + patching and package install
  + Attach/detach EBS volumes, create snapshots, backup DynamoDB table, etc.
* Trusted advisor reposts and advises in the following areas (most of them only available to business and enterprise support plan users):
  + Cost optimisation (Low utilisation, etc.)
  + Performance (High/over Utilisation, etc.)
  + Security (Check IAM, ports, whether have MFA think the steps to secure your root account when you sign up)
  + Fault Tolerance (Availability risks)
  + Service Limits (e.g. Check if you have more than 80% of your allowed amount of EC2)
* Organize and view inventory, group resources by whatever rules you’d like, including on-premise systems.
* EC2 should have an SSM role, otherwise it may not have the privilege to execute some of the automations
* Automation: Do the same thing on multiple instances all together according to customisable “document”. (Say terminating all windows instances)
* Run Command: run command(script) without logging into the instance, can run AWS managed script or own script (say install git for 20 Linux instances, or update 20 windows instances). Can save results into S3.
* In patch manager you can configure a baseline, which is the lowest version to patch to. Then any system lower will be patched.
* Maintenance window: schedule a window for maintenance regularly, and you can set automation/run commands after maintenance window starts (automated, programmed and remote maintenance). Based on CRON.
* State Manager: another configuration manager for all inventory, can change configuration back to baseline when some individual instance was changed. (a bit better than Config, it’s a “manager”)
* Activations under “Shared resources”: allows you to register (by installing SSM agent) on-premise system to system manager for monitoring, automation, etc.
* Parameter store: A place to store DB connection string, passwords, etc. so that you don’t have to hard code. Parameter store can be encrypted.

**High Availability**

When talking about elasticity and scalability in SysOps test, it’s mostly about DB.

* Elasticity vs Scalability: short term vs long term, on demand vs permeant, May be asked “what’s the best option from a scalability/elasticity point of view”
  + Elasticity approach typically involves setting up autoscaling, alarm, threshold, etc.
  + Scalability approach: Add more reserved instances, etc.
* Traditional RDS is not very elastic, you cannot scale an RDS on demand (to put a read replica online takes a lot of time). Unless you use Aurora serverless.
* Aurora (My SQL and PostgreSQL compatible): can scale by using a bigger instance, can be elasticity using Aurora Serverless.

DRS Multi-AZ fail-over (Again, disaster recovery/ maintenance only, not for performance, not a scaling/elasticity solution, only helps maintenance performance like when taking snapshots/backups, creating read-replicas etc.)

* A full copy of prod server for fail-over, AWS manages the failover without changing the connection string, normally about 1 minute and don’t need administrative intervention.
* All data written to the primary DB will be synchronized to the stand by DB
* Support MySQL, PostgreSQL and Oracle use physical replica, where SQL Server uses internal mirroring technology for multi-AZ replication
* Back and restore is done through a secondary copy of your RDS so prod load is not impacted. You cannot do this against read replicas.
* You can force a failover by shutting down an instance manually. In the case of restoration, you restore the secondary DB and promote to the primary.

Read-replicas (Asynchronies replication, master-slave pattern):

* Basically a read-only replica of your database to improve performance
* Used to overcome IO limits.
* When source DB is down (e.g. for maintenance), Read replicas can still serve read traffics.
* Say you’re running DWH or BI loads, you can run that solely against read replicas (However a better approach might be import that data into redshift and run against redshift).
* Support MySQL, PostgreSQL, MariaDB using native asynchronous replications, for up to 5 replicas. Aurora storage is de-coupled from DB instance and by default spread across 3 AZ and 6 copies. Therefore, it doesn’t need physical read replicas instances. In other words, read replicas are implemented at storage layer which avoid the need of replicating data across instances.
* When creating a new read replica, AWS will take a snapshot. If multi-AZ is enabled, then this snapshot will be taken against your secondary and will have no performance impact. If multi-AZ is not enabled, the snapshot will suspend your DB for about 1 minute.
* A new read replica has its own DNS address, can be promoted to a new standalone DB. By doing this, the whole primary-secondary backup link will be broken.
* Read-Replica can be multi-AZ now too
* You can have read replica of read replicas, even in a different region.
* Key metric to monitor is Replica Lag, to keep an eye on replica latency

Creating Multi-AZ RDS with read replicas.

* Multi-AZ can be set up when setting up the RDS instance, or later.
* Never make your RDS publicly accessible, even you need to maintain from public network you should use bastion host/jump box
* RDS can be encrypted and automatically backed up for security, however not required. Automatic backup is required if you want to have read replicas. When you want to create a read replica of a read replica, remember to manually activate backup on the source replica first (Interestingly turning on auto backup is not an option when creating a read replica. You have to specify a number for retention days, and a daily time window for this). Note automated backup window is not scheduled maintenance window.
* When you make any change after RDS instance is up and running, you can set it to take effect straightway, or wait for next maintenance window.
* You can let AWS handle your minor updates, which will happen during maintenance window (either you specify or leave it to AWS)
* RDS, being a DB running on AWS managed EC2 (with CloudWatch monitoring agent installed), shows more CloudWatch metric than a normal EC2 instance
* When forcing a failover with a reboot, it can take several minutes for the information (primary zone/secondary zone) to be reflected in the console.
* You can see what version of DB engine you’re running in console, or in command line requesting aws rds describe-db-instances --region an instance and check the returned JSON.

ElastiCache:

* Typical question would be a DRS is under a lot of load, what to do. In the case of read-heavy and data is not frequently changed, or compute intensive results like machine learning based results, use ElastiCache. Otherwise read replica.
* If ElastiCache is used, choose Memcached for simple (object based) caching or need multi-thread
* Choose Redis for Multi-AZ, open source, key-value pair caching and single thread is not a problem. Specific cases include Pub/Sub, Sorted Sets and an In-Memory Data Store, but no relational data.
* In case of OLAP is used and caused stress, moving to redshift might be a good choice also

Aurora:

* 3\* performance PostgreSQL, 5\* MySQL, 1/10 cost Oracle
* 10G minimum, 10G incremental, 64TB max. Up to 64 V CPU and 488G C memory
* Loss of 2 copy won’t affect write and loss of 3 copy won’t affect read. 2 copy is maintained in an AZ and minimum 3 AZ, 6 copies at least all the time. (Note it’s 6 copies of data forming a cluster, not 6 instances)
* Storage is self-healing (what level?)
* Up to 15 Aroura, read replicas, and up to 15 MySQL read replica.
* When 100% CPU utilisation is reached, scale up when write is an issue and scale out when read is an issue
* Can switch between RDS (provisioned Aurora) and Aurora serverless in console (not yet available in every region)
* Encrypted by default. And By default all replicas of encrypted resources are encrypted.
* Back track enables rollback to any point in time.
* You can have a different endpoint for read replicas, so it only allows read traffic operation.
* Failover order is determined by failover priority (tier 0 to tier 15), always fail to the lowest number (highest priority). If want to set an instance default failover, set it tier 0.
* When there’s 2 instances of the same failover priority, AWS promote the bigger instance. If they are of the same size, then random.
* Considering the architecture of Aurora, you should always think about a cluster in a region. Aurora can have cross region replica, which is an entire new cluster.
* If replication is disrupted, you must set it up again. Therefore when a failover happened, the last existing servers won’t serve as read replicas anymore. To avoid this it’s better to set up “Multi-AZ Deployment” at all times.

Troubleshoot Autoscaling, when new instances don’t show up:

* Key pair/Security group don’t exist
* Autoscaling config/group is not working/found
* AZ doesn’t support the instance type you’re trying to use/ AZ is lost
* Invalid EBS, no privilege (e.g. blocked by AWS organization)
* Attempting to attach an EBS block device to an instance-store AMI (when no additional EBS allowed)

Placement group:

Spread VS Clustered, refer to SAA notes.

They all must have unique name and doesn’t cost extra. Spread Placement Group has a limit of 7 instances per AZ.

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html>

**Storage & Data Management**

* S3, features and facts (charges) refer to SA/DEV note (Must do, lots of content!!)
* S3 intelligent Tiering: for unknow or unpredictable access pattern. You only choose from 2 tiers: frequent or infrequent access, and this server automatically move your data to most cost-effective tier based on your access pattern:
  + It moves object not accessed in 30 days to infrequently access tier
  + It moves object back to frequent access tier once an infrequently access object is accessed
  + 11 9’s durability, 99.9% availability
  + No cost for accessing data, extra monthly automation/monitoring fee of $0.0025/1000 objects
* Server request option: Record who, when requested what. Logs stored in S3 too (own bucket though) and can be huge for frequently requested objects. To avoid that you can setup lifecycle rules for log files.
* Versioning: keep all versions of objects, can be huge for frequently updated objects.
* Object-level logging: CloudTrail API call logs on object level
* CloudWatch option: enable monitoring
* MFA delete: Protect against accidental or malicious deletions. Once enabled, you need MFA for deleting an object, and enable/disable versioning.
* When a policy complains “policy does not apply to specified resource”, normally you have to be one step more specific with a wildcard: instead of “resource”, try “resource/\*”
* SSE-S3 vs SSE-KMS: KMS can provide CloudTrail auditing key and cost more.

EC2 Types: EBS vs Instance Store (DEV topic, review)

* When EC2 service was launched, all instances were backed by ephemeral storage. (non-persistence, temporary storage). EBS came later.
* Instance store root volume is 10 GB max, EBS can be up to 1 or 2T depend on the OS. EBS is quicker to boot also 1min vs 5min.
* By default, even EBS backed instance delete root device when instance is terminated. If you want root volume to persist after instance termination, you should select this at instance creation time. Attached additional volumes will persist by default.
* Instance store volume will all be deleted upon instance termination (or failure of an underlying drive), however EBS backed volume attached to instance store instance, will persist as well.
* As a result, instance store volume needs backup strategy (to S3/EBS), and don’t put and critical/production data there.
* to copy an instance-store-backed AMI, the user must have the following Amazon S3 permissions:
  + s3:CreateBucket (to create target bucket)
  + S3:GetObject (to get image from source)
  + S3:PutObject (to load image to target)
  + s3:GetBucketAcl (to check permission)
  + s3:ListAllMyBuckets (to find bucket)
  + s3:PutObjectAcl (to secure object)

EBS management/maintenance

* EBS and EC2 must be in the same AZ, considering latency requirements.
* To identify which volume is root, see which one has a snapshot already (root got a snapshot at launch time). Attached ones don’t have snapshot upon creation.
* EBS size and type can be changed on the fly now.
* To move an EC2 across AZ, the only way is to us a snapshot, then create an image from the snapshot, then boot from the image into another AZ. If want to boot in another region, copy that image to the target region. (or copy the snapshot then create image, order doesn’t matter)
* When wanting to delete the whole thing, remember to delete snapshots, images, and EBS that’s not root.
* Snapshots exist in S3, are incremental, not a full back up every time you take snapshots, so the first one takes the most time.
* Best practice when taking snapshot for root/changing EBS size or type is to stop instance, although it’s not necessary.
* Snapshots cannot be shared if encrypted as you cannot share the key to decrypt it. If not encrypted, you can share with AWS or even be made public.

Encryption & Downtime:

* Most AWS services, encryption can only be enabled upon creation (EFS, RDS, EBS, etc.), otherwise you need to re-create and transfer the data (rsync in linux or robocopy in windows), and this involves down time as it is a preferred practice when migrating data.
* For EBS, you cannot change encryption status (so cannot unencrypt if encrypted)
* Only S3 is flexible, and at an object level or bucket level. No downtime.
* A trick for encrypt an existing EBS, is to encrypt the snapshot and restore. However it is recommended to stop instance when taking snapshots also.

KMS vs Cloud HSM:

* Both are designed to generate and manage keys to protect your AWS resources. Cloud HSM normally used to protect keys.
* KMS is shared hardware, multi-tenant. When multi-tenant is not an option you cannot use (financial industry/Banking/Government). Free-tier eligible. Can use on EBS, S3, RDS, DynamoDB, etc. symmetric encryption/decryption (same algorithm both ways).
* Cloud HSM is dedicated hardware, not free tier, FIPS 140-2 Level 3 compliance (Level 4 being the highest, however it’s still very high and internationally recognizable). Can be used in: Digital Right Management, Public Key Infrastructure, document signing, transaction processing etc. Supporting asymmetric encryption.

AMIs (and difference from snapshot):

* Include 3 things: template of root (but can be used on different size/type of EC2), launch permissions (who can launch), Block device volume (additional block you want to attach)
* Multiple OS available, and can create custom (You must start from an existing one and SSH into the instance to config it, including installing packages)
* AMI must be registered before can be used. If AMI is generated from console, it’s automatically registered. If generated from CLI or API, you must generate manually.
* AMI does not span regions (or region bound)
* You can: keep it private, share with other AWS account (read-only access), make it public, sell it (Open source possible??). When you do any of these, you still have full control of the AMI and still charged for the storage of the AMI.
* You cannot: copy an encrypted AMI, you need to ask the owner to snapshot and create an encrypted copy for you and share you with both the file and the key.
* You cannot: direct copy an AMI with a billing Product code, like Windows, RedHat, marketplace AMIs, etc. Need extra licence/subscription. To copy you need to generate an AMI from a running instance

Snowball:

* Uses local network so much faster even than direct connect.
* Tamper-resistant enclosure hardware design
* 256-bit encryption by default, snowball client(A must for your local device) handles that.
* Region specific
* Use this when you have too many data (rule of thumb: more than one week to upload), for large one-off transfer, for physically/geographically isolated location (or cost prohibitive, like a military base)
* Snowball edge, snowball + edge computing, can run lambda as the data is copied to snowball, but must be requested so AWS can pre-configure it for you.

Storage gateway:

* On premise software (Storage gateway virtual Appliance) that combines AWS resource and your on-premise resource so the whole thing feels like on premise. Your on-premise resource must support ESXi or Hyper-V.
* File Gateway (NFS/SMB): file stored in S3 but feels like on premise, can use all S3 benefits like policy, versioning, lifecycle, replication, etc. Can be cheaper than local storage (NAS). Server -> Storage Gateway -> VPC/Direct connect/Internet -> different S3 class
* Volume Gateway (iSCSI). You can choose to store locally and use AWS to backup (Stored value, still need full capacity on site, backup is EBS snapshots in S3). Or store on AWS and use local as cache (Cached volumes, don’t need full capacity on site, you can still take snapshots inside of AWS).
* Tape Gateway (VTL for archiving, iSCSI too), backed up by glacier, and integrate with existing tape archiving infrastructure like Backup Exec, Veeam, etc.
* Gateway has a whole VM, including Host, Hypervisor, Cache, Buffer, driver, etc.

Athena (not tested a lot in exams but might be super useful):

* Serverless so no need to provision, paid by TB scanned, Query S3 using standard SQL. No complex ETL process.
* Mostly used to query log files (AWS and any log files) and generate BI reports.
* You still need to create a DB first in Athena, then create table from file (decoding JSON most likely, when analysing AWS logs you might be able to find templates), before you can run SQL queries.

**Security & Compliance (read white paper)**

Compliance Framework: Three ISO, PCI, HIPAA (US only, health related)

* Check aws.amozon.com/compliance for more
* ISO/IEC 27001:2005, a standard for information security management system. It’s critical to managed service provider.
* FedRAMP: government-wide program that standardize security assessment and continuous monitoring of cloud services. US program.
* HIPPA, a law to ensure the security of people’s healthcare information, also help to reduce administrative cost when it comes to healthcare information (among insurance, care provider, etc.), US program, not all AWS services are HIPPA compliant.
* NIST: US standard, best practices to managed infrastructure cybersecurity risks.
* PCI: Payment card data industry data security standard (PCI DSS), super important, to protect cardholders against misuse. 12 steps.
  + Install and maintain a firewall to protect data (you can do through Security group, ACL)
  + Do not use defaults for passwords and other security parameters (username)
  + Protect stored cardholder data (at REST)
  + Encrypt transmission (SSL/TLS)
  + Protect all system against malware and regularly update anti-virus software
  + Develop and maintain secure system and applications
  + Restrict access to cardholder data
  + Identify and authenticate access to system components
  + Restrict physical access to cardholder data
  + Track and monitor all network access (CloudTrail/CloudWatch/Config)
  + Regularly test security systems and processes
  + Maintain a policy
* SAS/SOC1/FISMA not important
* FIPS 140-2 is a standard for cryptographic modules, Cloud HSM is level 3.

DDoS (Distributed Denial of Service)

* Multiple mechanisms, such as large packer floods, botnets (can be bought), amplification techniques, reflections, etc. You can even build one with AWS of other cloud services.
* Amplification/Reflection (Layer 4): Send a request to a server that amplify the request
* Packer floods (Layer 7): overwhelm server with GET requests so it cannot establish new connections
* Slowloris attack (Layer 7): hold as much as possible connections open and slow things down (send partial http request)
* How to mitigate: Minimising surface use ELB, be ready to scale to absorb, safeguard exposed resources. learn normal behaviour, create a plan and follow the playbook.
* AWS shield, free, for ELB, CloudFront and Route 53, protect from SYN/UDP floods, Reflections attacks and Layer3/4 attacks, turned on by default.
* AWS shield advanced protect your application running on ELB, CloudFront and Route 53, from more sophisticated attack, $3000 per month. When DDoS happened, resource used to mitigate the situation won’t cost you money (ELB, CloudFront and Route 53).
* AWS shield advanced also give near real time notification and is handled by a dedicated team
* Basically remember, when DDoS happens, what to use: CloudWatch, WAFs, Autoscaling, AWS Shield over ELB, CloudFront and Route 53.

AWS Marketplace – Security Products/ penetration testing

* Buy all sorts of things, even a cloud guru training, and it go through your AWS bill.
* Not a service, so must look for it from the home page. Product can be search/browsed in categories, like firewalls, hardened OS, WAFs (web application firewalls), antivirus, security monitoring, etc.
* Products are region specific, can be free.
* For penetration testing, Kali Linux is the industrial standard. It’s a flavour of Linux with a variety of security tools (can heck Wi-Fi, if your hash is cracked previously at “only hash crack”), has its own certification program.
* Pricing model: free, hourly, monthly, annual, BYOL, etc.
* For penetration testing, you need to request and may need a marketplace product (kali Linux). Using a marketplace product does not automatically give your permission.

IAM custom policies

* Different form AWS managed policies, 2 ways to create.
* Visual editor: choose service -> choose actions -> choose policy scope -> name the policy -> add description -> Generate.
* JSON: Manually to do the above
* Check IAM:PassRole

Role & Custom policy

* Again, always use roles, instead of user credentials.
* Role can be attached to EC2 on the fly now (almost instant, across all resources applied, across all region, only takes several seconds), used to be only possibly upon creations of EC2
* Sometime when copying to S3 reports authentication error, you might want to specify region by adding –region eu-west-2

MFA on individual resources & Reporting who has MFA, with IAM

* MFA can be configured at a per-user basis, just assign a MFA device in console.
* Assign MFA from console:
  + Aws iam create-virtual-mfa-device –virtual-mfa-device-name EC2-User –outfile /out/put/dir –bootstrap-method QRCodePNG
  + This will create the device for “EC2-User”, output to /out/put/dir, in PNG formatted QR code.
  + Copy the PNG to S3 then access from browser. So far you only created MFA, not assigned yet.
  + Aws iam enable-mfa-device –user-name –serial-number arn:aws:iam::[resources\_number]:mfa/EC2-User –authentication-code-1 [Code1] –authentication-code-2 [Code2]
  + All done
* Then to use resources protected by MFA in CLI, you need to go through STS, security token service. You can also enforce the use of MFA from CLI, via STS.
* To check all users, weather any of them has MFA, you can use “credential report” in IAM.

STS (Security Token Service)

* Grant temporary access to resources, users can come from: Active Directory federation (combining user from one domain like IAM with a list of users in another domain like Active Directive), Mobile Apps (Identity broker, open ID provider like Google/Amazon/Facebook), Cross Account Access.
* Remember: always authenticate against ID provider (active directory/Google/Facebook/etc), trade for a STS token (GetFederationToken API, include a policy and duration of 1 ~36 hours) which can be used by the caller app, to pass IAM checks.
* Can grant token to user authenticated with Cross Account Access too.

Logging:

* The good old CloudTrail, CloudWatch log, Config. And VPC Flow Logs.
* Log File access should always be locked down, prevent unauthorized access, and ensure a role-based only access.
* Always setup alerts when logs are created or failed to be created, using CloudTrail notifications or AWS config rules. Notification should not expose details too much, just point to the log file location is fine.
* Should set up log for system components changes, with AWS config form a configuration level and CloudTrail from an API call level.
* CloudTrail can do log file validation and encryption, to prevent log file from being modified.

AWS hypervisor, isolation of AWS resources, AWS firewalls

* Hypervisor is the software/firmware/hardware that creates and runs virtual machines. The Computer that runs hypervisor is a host machine, and the virtual machine running on it is called guest machine.
* EC2 runs on the Xen hypervisor, which can have guest OS running as Paravirtualization (PV) or using hardware virtual machine. However, the new generation of C class EC2 C5 which just got released is running on KVM. This was not officially announced, just leaked from some blog.
* HVM supports windows only, for Linux instances can be on both PV and HVM is available.
* HVM guests are “fully virtualised”, which means guest does not even aware that it’s sharing processing time with other VM. PV is lighter and used to be faster, but the performance gap is not almost closed, and AWS recommend using HVM wherever possible.
* Para virtualized guests rely on hypervisor to perform operations that normally require privileges. Normally they run in 4 different privilege modes provided by CPU: hypervisor runs in Ring 0, OS runs in Ring 1 and Apps run in Ring 3.
* Isolations: resource isolations are done by firewall which runs in hypervisor level, directly on hardware, and hypervisor/firewall handles security groups as well.
* Hypervisor can be accessed through the management plane, via a purposes-built administration host. The administration hosts are purposely designed, built, configured and hardened, and all access are logged and audited. Only specific admin personal should access and when jobs are done all access should be revoked.
* Instances are completely controlled by customer, AWS cannot access.
* AWS automatically resets and scrub (set to 0) every block of storage used by a customer, so customer data will never be un-intestinally exposed to another user.
* For memory, AWS scrub it before returning them to the pool of available memories. Basically, recover tools won’t work.

Dedicated instances vs dedicated host (You can choose either under tenancy when launching EC2)

* Dedicated instance run on dedicated host but can share hardware with other instances from the same account. Available as on-demand, reserve or spot, billed per instance.
* Dedicated host give you full control at host level, you can see sockets, cores, host ID, etc. And you can keep launching instances in the host dedicated to you (target instance placement). Billed per host instead of per instance.
* Dedicate host is not just an option when you provision an EC2, there is a separate sub-service in the left side menu of EC2.

System Manager EC2 Run Command

* Not tested a lot but used a lot in job.
* System manager can be accessed from EC2 menu or under management tools.
* To run a command, select from console, specify targets by tag or instance ID, CLI command is also created if you want to run in CLI.
* Command can also be issued from AWS Tools for Windows PowerShell and Amazon SDKs
* All EC2 being managed should have a role to be managed by system manager, and your instance need SSM agent installed.
* Scenarios: you need to update security patches for 100 instances, how should you do it.

Systems manager parameter store

* Central parameter store and passed in whenever used. Think about parameter file in informatica
* Workflow: Create a parameter store -> Add sensitive data -> accessed by different services
* Can have 3 type of parameters: String, String List, Secure String (for passwords, DB connection strings, license codes, etc.)
* Once setup, this information can be used by EC2, Lambda, CloudFormation, EC2 run command (no RDS!!, you can use parameter store to store connection string to RDS for EC2 and Lambda use, but cannot be used by RDS directly say saving DB password in parameter store), etc. They are referenced by name.

Secure S3 with pre-signed URLs

* Normally done via SDK as the signing processes is normally automated, however can be done in CLI as well.
* Use the command: aws s3 presign s3://bucket\_name/object\_name.xxx [--expires-in 300]
* This will generate pre-signed URL, and you can access this object before the expire time (in seconds, default is 3600 seconds).
* This way you don’t have to bucket policies or object ACLs.

AWS config with S3

* Several config rules important to know: s3-bucket-public-write-prohibited, s3-bucket-public-read-prohibited.

AWS Inspector (in security/compliance) vs AWS trusted advisor (in management tool)

* Normally when you need to generate report, you want inspector.
* Trusted advisor is for cost optimisation, performance, security, fault tolerance.
* Inspector is an automated security assessment looking for security vulnerabilities or deviations from best practices. Once done it provide detailed findings prioritised by level of severity, viewed in console or API.
* Workflow: Create an assessment target (by tag recommended) -> install agent (You might want to use SSM run command if you got a fleet of EC2s, otherwise there’s standard commands in the console, and you’re recommended to verify before running) -> Create Assessment template (Different rule packages available can choose one or more, and you can specify inspection durations, the longer the fuller the results) -> perform assessment -> review findings (PDF, can choose findings only or full report. Full report might take a while, 600+ pages document, including instance ID, finding summary/detail, what’s tested, etc.)
* For AWS inspector to work, you need to give the inspector read-only access so that it can inspect resources.
* Findings are viewable from report, as well as from console.
* Four packages:
  + Common vulnerabilities and exposures
  + CIS OS Security Configuration Benchmarks (OS hardened or not)
  + Security Best Practices
  + Runtime behaviour analysis
* Four severity level: High, medium, Low, Informational
* Trusted advisor: can advice on cost optimisation, increase performance, security (possible as well) and fault tolerance. It does core checks and recommendation for basic/developer plan (like when you created an account it was trusted advisor which told you to setup MFA) and detailed checking is for business/enterprise plan only.

Shared responsibility model

* Security of the cloud is AWS, security in the cloud is customer
* AWS in charge of global infrastructure, hardware/software/networking, managed services.
* User in charge of all things in the user’s control. AWS in charge of what you cannot touch.
* The model changes for different service types: For EC2s you have to manager OS and antivirus etc. For container services like RDS, EMR and Elastic Beanstalk, you don’t manage OS layer. The Platform layer hence become a managed service so it’s not AWS’s responsibility. For abstract services like SQS, SES, DynamoDB, S3, etc. you don’t manage anything beyond what you can touch with API or user endpoint.

Other Security Aspect.

* Security group are stateful, which means it allows inbound and outbound.
* AWS artefact: Generate compliance documents/reports (ISO certs, PCI cert, Service Organization Control reports) so that you can demonstrate to your auditor/regulator that you’re doing things right.
* CloudHSM vs KMS, Single vs Multiple tenancy, CloudHSM support asymmetric encryption (public+private key), compliant with FIPS 140-2 & EAL-4 and more expensive.
* Instant encryption is only available with S3, RDS, DynamoDB, EFS, EBS need encryption with migration, that’s created a new encrypted environment and migrate. (in cases like you have to encryption suddenly, the case for DynamoDB might change for 2019, but not sure)

**VPC**

VPC (same across all 3 certs but worth revision again)

* Think about Datacentre, so when you set up account, every region has a default VPC.
* Let you provision a logically isolated environment in AWS that you define, and launch resources in your own manner, so you have complete control including creating subnet, managing IP range, configuration of route tables and network gateways.
* You can use a hardware VPC to connect your on-premise datacentre to your own VPC, to use AWS as an extension of your own infrastructure.
* Traffic from region -> VPC via internet gateway or virtual private gateway (for VPN)
* Once into the VPC, you hit router first, which go through Route table (define which SN can talk to which SN) -> Network ACL -> private or public subnet, which is protected by security groups
* Traffic from public won’t be routed to private subnet and won’t pass NACL of private subnet.
* One Subnet is always in one AZ.
* Available cidr range: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, the smaller the prefix, the bigger the range, range = 2^(32-prefix). You always lose 5 address by AWS default, 0 is network address, 1 is reserved for VPC router, 2 is reserved for DNS server, 3 is reserved for future use and last is network broadcast address.
* Once all set up, we can: launch resources in subnet of our choosing, assign custom IP address range for each subnet, configure route table to define which SN can talk to which SN, attach a internet gateway if you want it public (you can have one only, it’s multiple AZ, highly available), use NACL to block IP addresses, apply security group in different SN as they’re multiple AZ by nature.
* Default VPC vs Custom VPC:
  + default is internet accessible from beginning, you don’t get private subnet
  + each EC2 always have public and private IP address, EC2 in private subnet won’t have public IP addresses (don’t need one if you’re using jump box/NAT gateway to access)
* VPC peering can cross account, behaves as they’re in the same VPC, allows communication via resources’ private IP address. No transitive peering!!
* VPC can have dedicated tenancy
* When you create VPC, it creates a default route table, a default security group and a default NACL
* Next step is to create subnets
  + Bu default it will be associated to the default route table. You don’t want your default route table to have a way out to internet, so better setup another route table to direct all IP V4/V6 traffic to IGW. This way all subnet is private by default and can only go out to internet when associated with this new route table
  + For subnets you want to be publicly accessible, enable auto-assign IP, this will be inherited by the EC2s launched in this subnet later
  + Once you associate a new subnet with a new route table that targets an IGW, it’s no longer associated with the default route table and cannot talk to other subnets anymore. To enable communication, you can setup private subnet’s security group to accept the public subnet’s cidr range
* Now you can launch resources into subnets. Note that during launch when you’re choosing security group, you won’t see the ones you set up outside the VPC, because security group don’t span VPCs, only span subnets.
* You need to manually setup a way out for your private subnet, even you can SSH into it with a bastion host. To allow traffic out to the internet, you need a NAT instance or NAT gateway.
  + In the case of NAT instance, create an instance and disable source-target check, because it’s neither source nor target. NAT instance can be a bottle neck or single point of failure (being a single instance with limited throughput in only one AZ).
  + You can have your NAT instances auto scaling and Multi AZ (script-based failover), it used to be the practice.
  + Once NAT instance is ready, allow port 443 HTTPS in its security group and point all traffic targeting outside to NAT instance.
  + In the case of using NAT gateway, use egrass only NAT gateway for IPv6 and normal NAT gateway for IPv4. Once setup public IP is automatically assigned.
  + No matter using NAT gateway or instance, it should be in the public subnet. NAT gateway takes 10~15 minutes to be up and running.
  + In the case of using NAT gateway, you need one NAT gateway in each AZ, add all of them to the route table and failover will happen automatically.
  + Note, NAT instance sit behind public subnet’s security group, while NAT gateway sit ahead of public subnet’s security group.

VPC security group and NACL

* Default one allows everything pass in and out.
* Each subnet can only associate with one NACL, whatever you want to allow/block goes into that NACL. You can reuse one NACL on any number of subnets, but only within its VPC as it is VPC specific.
* When you create a new NACL, it denies everything. Only VPC default allows everything upon creation. Therefore, you have to setup rules. AWS recommend setting rules numbers in incremental of 100, with IPv6 = IPv4 rule number + 1. This is giving you chance to change it 100 time in maintenance.
* The lower the rule number, the higher it is evaluated. In the console, rules are ranked by importance
* NACL is assessed before security groups, NACL is stateless. Also you can specifically block an IP, where in security group you can only allow.
* Ephemeral port: the random port you use to communicate after you made a request on a well-known port. Say you made a http request on port 80 out, then the server request you to provide a random port to receive the data. The range of choices depends on your environment. For ELB and NAT gateway it is 1024 – 65535, for Linux it is 32768 – 61000, for windows server 2003 is 1025 – 5000, etc.
* You need to enable ephemeral port in your outbound rule.

VPC endpoints:

* An alternative route to private subnet -> NAT gateway -> internet -> Other AWS service. VPC endpoint gives an internal gateway so you can bypass internet
* Two choice of endpoint: interface/gateway, gateways are much more reliable.
* Create an endpoint from side menu of VPC, it acts like a NAT gateway, but only can go out to specific AWS services say S3, and it happens internally, thus, bypasses internet.
* The AWS resources you try to access does not necessarily have to be in the same VPC.

VPC flow logs:

* Log IP traffic going in and out VPC, supported by CloudWatch Log
* Can operate on 3 levels: VPC, subnet, Network Interface.
* Filter can be setup upon the creation of flow log so that is logs only allow traffic, only rejected traffic, or all traffic.
* Flow log need a role to be able to write to CloudWatch, and you need to create a log group in CloudWatch to receive the logs.
* You can stream these logs to lambda or Elasticsearch services so you can be proactive (might be a feature available for all CloudWatch metric though). You can also export the log to S3 for archiving.
* You cannot log the VPC that’s peered with your VPC, unless the peered VPC is in your account. You cannot tag a flow log, you cannot change any configuration once the flow log is created.
* Not all traffic is monitored, traffic to contact Amazon DNS server, windows licence authenticator, 169.254.169.254 for metadata, DHCP and to default VPC router will not be logged.

Clean up VPC:

* Delete resources in subnet -> NAT Gateway -> endpoint, IGW (cannot detach if NAT is in position) -> VPC with all other objects all together

CIDR calculations:

* Four things need to be able to calculate: netmask, first IP, last IP, total IP count.
  + Netmask: 255.255.255.255 minus last IP,
  + First IP: IP, say 10.0.0.0/24, first is 10.0.0.0 or 10.0.0.1 if talking about first available.
  + Total: 2^(32-CIDR range), /24 gives you 2^8 = 256 address, from .0~.255, however .0 and .255 is always reserved and .1, .2, .3 are reserved by AWS
  + Smallest is /28 and largest is /16
  + Normally you’ll be given a number of devices and be asked what’s the smallest CIDR range possible to fit all of them in one VPC

Direct Connect Gateways

* Your own datacentre -> Single AWS direct connect -> private virtual interface -> direct connect gateway -> multiple VPCs
* To make your multiple VPCs extensions of your datacentre, using multiple direct connect connections is not practical and too expensive. Therefore, you use direct connect to get into AWS, and direct connect gateway handles connections to each VPC via AWS backbone.
* You need to add a route to your route table to allow outbound traffic to direct connect gateway so it can go to your data centre.
* It is very similar to a NAT gateway or a VPC endpoint, except it is for your own datacentre.
* Possible question: you need to expand to another region, and need the security and performance of direct connect, what should you do. Don’t put in another direct connect connection, use a direct connect gateway instead.
* Uses Border Gateway Protocol (BGP) to share routing information

DNS

* Resolve human friendly addresses to IP address, like a phone book, almost as old as internet
* Top level domains: things after the last dot in a DNS domain, like .com. There’s only a limited number of top-level domains.
* Domain registars: make sure Domain names are not duplicated. Registered addresses are stored in a WhoIS database. Popular ones are GoDaddy.com, amazon, 123-reg.co.uk
* Start of Authority (SOA) Record contains the following information, each zone only have one SOA:
  + The name of the server that supplies data for the zone
  + The owner/admin for the zone
  + The current version of the data file served for the zone
  + Default TTL for the zone, you want it as short as possible (default is 300 in route 53 but you want it to be 60 seconds) so changes reflexes fast
* NS Records: standard, uses top level domain as an entry to search for domain. Workflow: input a domain -> look up by first level domain -> if not cached, lookup in WhoIS (in case of your registar is AWS, route 53)-> return a SOA, with an “A” record or else
* “A” record: most fundamental, return exact IP
* CNAMES: resolve one domain to another, like m.XXX.com to mobile.XXX.com, so you can have more than one domain name mapped to one DNS record. Cannot got to a naked domain name (nothing like [www./mobil](http://www./mobil). In front your domain name), for naked domain name, use Alias records. Basically used to route in the same domain.
* Alias Records, map a domain to another DNS, maybe an AWS resources like an ELB (actually, that’s the most stable way to point a DNS to a ELB), CloudFront distribution etc. AWS handles backend DNS changes when using an Alias records (e.g. when the backend IP of a ELB changes, Alias automatically point to the new IP with the same DNS). Given a choice, always use an Alias Record over a CNAME.
* MX records, PTR records are also popular but not tested
* DNS takes time to propagate around the world

Route 53

* It’s global
* You can purchase a domain name under a variety of top-level domain under route 53. However, when you purchase, you automatically get several addresses under other top-level domain as well, to prevent outage of your primary domain.
* Routing policy
  + Simple routing: randomly route among all possible addresses (if use “A” records, all IPs are under same record and separated by a new line). You can only have one simple routing record in route 53, since if you want randomness, you should put all options in one record.
  + Weighted routing: give weight in whole numbers and it will calculate based on the percentage of the sum. Each record only has one “A” or Alias record and you should have multiple records each have a weight number.
  + Latency based routing: route based on which region you can get the fastest response. Note the logic is to choose region first, and look up what record is available under that region (you can possibly set resource from another region even though it doesn’t make sense, say route request from Melbourne based on which region in [South America, North America, Australia], obviously it will choose Australia, but you can have “A” name of a server in EU as the target in the Australia record set). You cannot mix routing policies among “A” records, so have to choose one between latency and weighted. You can have multiple addresses in one record.
  + Failover routing: Health check your active, if fail it fails over to passive. Health check can be endpoint monitoring (say homepage can respond), calculated health check or CloudWatch status. Health check should be created separately, key parameters including failure threshold, request interval. Once active become health again, the failover will fail back. Note when you test it by manually stopping an instance, the IP address changes when it’s started again.
  + Geolocation Routing: Route European users to a site that display Euro and local language. Location can be country or continent. Different from latency as you might be physically closer to somewhere you have huge latency to.
  + Multi-value answers: when one fails health check, remove it from the record set and randomly route traffic to the remaining server.

**Automation & Optimisation**

CouldFormation (Check Developer note):

* Created from JSON or YMAL template, which is first uploaded to S3 (a new bucket will be created automatically, and you can only delete this bucket after you deleted the stacks it created), then AWS makes API calls. Your JAON and YMAL file indicate the end status of the infrastructure you’re provisioning or changing.
* Benefits include: consistency with fewer mistakes, less time and effort, can version control, free, can managed dependencies, can be used in the case of deletion and rolling back.
* YAML structure refer to dev note.
  + AWSTemplateFormatVersion is always: 2010-09-09
  + Description is comments
  + Metedata: tags
  + Parameters: parameterise input values, e.g. what key pair to use for an EC2
  + Conditions: provision resource based on environment
  + Mappings defines which region it goes to, and other region-specific values.
  + Transforms allow you to refer to extra code (not just template, but Lambda as well) from outside your template. There’re standard ones in AWS for you to set up association with other types of resources.
  + Resources: what you’re deploying, only mandatory field.
  + Output: confirmation text or can be input to another CloudFormation template.

CloudFormation Practical:

* Can start from a sample, like LAMP stack, RoR stack, WordPress blog, etc.
* Can specify if to rollback when something failed, default is to roll back when anything fails. To disable it in API, use –disable-rollback, or set DisableRollBack to True. More likely you’d want it not to roll back for trouble shooting.
* When stack is created, details can be viewed in the console including the outputs

Elastic BeanStalk:

* Is a Service for deploying web applications developed in Java, .NET, PHP, Node.js, Python, Ruby, Go and Docker;
* Can be deployed on Tomcat, Nginx, Puma, Passenger, IIS
* Not very configurable other than hosting Web Apps
* Allows you to only focus on the code and don’t worry about underlying infrastructure
* Elastic Beanstalk will handle deployment, capacity, load balancing, auto-scaling and health-check.
* It is free, you only pay for what you use, and you have full control of the underlying resources if you choose to do that yourself. You can specify EC2 type but it’s optional.
* Platform Updates are automatically managed.
* Dashboard allow you to manage application health. Integrated with CloudWatch and X-Ray for monitoring and debugging.

Elastic BeanStalk Practical:

* Start with “create a web-app”, give it a name, specify a platform, provide code (source bundle zip).
* When it’s completed, you can see details including exactly what resources were created. You also get a URL to access the app you uploaded
* To delete stack, simple do it in console for deleting the whole thing

OpsWorks:

* Automate server configuration using code using puppet or Chef, works with existing puppet/Chef
* Puppet/Chef are the two most popular configuration management tools, both work on Windows/Linux. For application configuration, account, scheduled tasks, etc.
* Whenever asked to use puppet or Chef, OpsWorks is the one to go