|  |  |
| --- | --- |
| Identity Access Management (IAM) | |
| Basics | Centralized control of AWS acct. Shared access to AWS acct. Granular Permissions. Identity Federation  MFA, Temp access for users/devices/services. Password rotation policy. PCI DSS compliance  1) Users. 2) Groups = collections of users w same permissions.  3) Roles = assign to users, apps, services to give access to AWS resources  IAM policy: doc that defines 1 or more permissions. - Can be attached to user, group or role |
| Walk-through | Create IAM role -> Select trusted entity type (AWS service, AWS acct, web identity, SAML 2.0 federation, custom trust policy) |
| IAM Policy Simulator | Test effects of IAM policies before committing them to production  Test policies already attached to existing users  https://policysim.aws.amazon.com |
| Exam | IAM is global not regional service  Root acct = acct created when you first set up AWS acct, has complete admin access  New users have no permissions when first created  New users are assigned an access key and secret access key when accts are created. Not the same as username, password which are used to log in to AWS Management Console  Use access keys to access AWS via APIs and CLI  Secret access key can only be viewed once. If lost, need regenerate them  - Always set up MFA on root account. – Can create and customized password rotation policies |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Elastic Compute Cloud (EC2) | | | | | | |
| Intro | Secure, resizable compute capacity in Cloud. Like a VM, only hosted in AWS instead of own data center  Can control capacity, scaling of instances  Pay only for what you use. No wasted capacity compared to data center  No need provision physical servers, saving money and time | | | | | |
| Pricing Options | 1) On Demand (default): pay by hour or second depending on type of instance you run  - for short-term, unpredictable workloads that cannot be interrupted, poc apps | | | | | |
| 2) Reserved: reserved capacity for 1 or 3 years. Up to 72% discount on hourly charge  - predictable usage, need specific reserved capacity, can make up-front payment for further discounts  - reserved instances operate at regional level  a) Standard RIs: up to 72% off on-demand price  b) Convertible RIs: up to 52% off on-demand price. Can change to diff RI type of equal or greater value  c) Schedule RIs: launch within time window you define. Match capacity reservation to a predictable recurring schedule that only requires a fraction of a day, week or month | | | | | |
| 3) Spot: purchase unused capacity at discount of up to 90%. Prices fluctuates w supply and demand  - set max price you willing to accept. If spot price > max price, instance might be terminated or hibernated depending on what you select  - for apps w flexible start and end times, apps that are feasible only at low compute prices, users w urgent need for large amts of additional computing capacity | | | | | |
| 4) Dedicated: physical EC2 server dedicated for your use. Most expensive  - for compliance or licensing which does not support multi-tenant/cloud deployments  a) On-demand: can be purchased on-demand (hourly)  b) Reserved: can be purchased as a reservation for up to 70% off on-demand price | | | | | |
| Savings Plans:  - Save up to 72% of all AWS Compute usage regardless of instance type or Region  - Commit to 1 or 3 years to use a specific amt of compute power (measured in $/hour)  - Applies to EC2, Lambda, Fargate | | | | | |
| AWS Pricing Calculator: explore AWS services and pricing | | | | | |
| Instance Types | 1) General Purpose: M8g, M7g, M7i, M7i-flex, M7a, Mac, M6g, M6i, M6a, M5, M5n, M5zn, M5a, M4, T4g, T3, T3a, T2  For apps that use compute, memory, networking resource in equal proportions like web servers and code repos, gaming servers, microservices, midsize data stores, caching fleets | | | | | |
| 2) Compute Optimized: C8g, C7g, C7gn, C7i, C7i-flex, C7a, C6g, C6gn, C6i, C6in, C6a, C5, C5n, C5a, C4  For batch processing workloads, media transcoding, high performance web servers, HPC, scientific modeling, dedicated gaming servers, ad server engines, CPU-based ML inference, distributed analytics | | | | | |
| 3) Memory Optimized: R8g, R7g, R7i, R7iz, R7a, R6g, R6i, R6in, R6a, R5, R5n, R5b, R5a, R4, U7i, High Memory (U-1), X8g, X2gd, X2idn, X2iedn, X2iezn, X1, X1e, z1d  For workloads that process large datasets in memory, large enterprise DBs (like production SAP HANA in-memory database) | | | | | |
| 4) Accelerated Computing: use hardware accelerators or co-processors like floating point num calculations, graphics processing, data pattern matching more efficiently than possible on CPUs  P5, P4, G6e, G6, G5g, G5, G4dn, G4ad, Trn2, Trn1, Inf2, Inf1, DL1, DL2q, F2, VT1  For gen AI apps (question answering, code generation, video and image generation, speech recognition), HPC apps at scale in pharmaceutical discovery, seismic analysis, weather forecasting, financial modeling | | | | | |
| 5) Storage Optimized: for workloads that require high, sequential read and write access to very algrge datasets on local storage. Optimized for millions of low-latency, random IOPS  I8g, I7i, I7ie, I4g, Im4gn, Is4gen, I4i, I3, I3en, D3, D3en, D2, H1  For I/O intensive workloads that require real-time latency access to data such as relations DB (MySQL, PostgreSQL), real-time Db, NoSQL DB (Aerospike, Apache Druid, Clickhouse, MongoDB), real-time analytics like Apache Spark | | | | | |
| 6) HPC Optimized: run HPC workloads at scale like large, complex simulations and DL workloads  Hpc7g, Hpc7a, Hpc6id, Hpc6a | | | | | |
| EBS Volumes | Elastic Block Store (EBS): storage volumes that you attach to EC2 instances  Highly available: auto replicated within a single AZ to protect against hardware failures  Scalable: dynamically incr capacity and change volume type w no downtime or performance impact  Can only attach volume created in same AZ as EC2 | | | | | |
| 1) General purpose SSD (gp2): 3 IOPS per GiB, up to max 16000 IOPS per volume  - gp2 volumes < 1TB can burst up to 3000 IOPS. – Good for boot volumes or dev and test apps which are not latency sensitive  General Purpose SSD (gp3): latest gen. Baseline of 3000 IOPS for any volume size (1GB – 16TB). Up to 16000 IOPS. – 20% cheaper than gp2 | | | | | |
| 2) Provisioned IOPS SSD (io1): high performance and most expensive  - Up to 64000 IOPS per volume, 50 IOPS per GiB. – Designed for I/O intensive apps, large DB, latency-sensitive workloads  Provisioned IOPS SSD (io2): latest gen, higher durability and more IOPS. Same price as io1  - 500 IOPS per GiB, up to 64000 IOPS. 99.999% durability compared to 99.9%  Provisioned IOPS SSD io2 Block Express: SAN (Storage Area Network) in the cloud. Highest performance, sub-milliseconds latency  - Use EBS Block Express architecture. – 4x throughput, IOPS, and capacity of regular io2 volumes  - Up to 64 TB, 256000 IOPS per volume, 99.999% durability  - For largest, most critical, high-performance apps like SAP HANA, Oracle, Microsoft SQL Server, IBM DB2 | | | | | |
| 3) Throughput Optimized HDD (st1): low-cost HDD volume  - Baseline throughput of 40 MB/s per TB. – Can burst up to 250 MB/s per TB. – Max throughput of 500 MB/s per volume. – For frequently accessed, throughput-intensive workloads (big data, data warehouses, ETL, log processing). – Cost effective way to store loads of data. – Cannot be a boot volume | | | | | |
| 4) Cold HDD (sc1): lowest cost option.  - Baseline throughput of 12 MB/s per TB. – Can burst up to 80 MB/s per TB. – Max throughput of 250 MB/s per volume. – For colder data not frequently accessed, need lowest cost and performance not a factor  - Cannot be a boot volume | | | | | |
| IOPS: num of read and write ops per sec  - Metric for quick transactions, low latency apps, transactional workloads  - io1 or io2 | | | Throughput: num of bits read or write per sec (MB/s)  - Metric for large datasets, large I/O sizes, complex queries  - st1 | | |
| Elastic Load Balancer (ELB) | Load balancer distributed network traffic across a group of servers | | | | | |
| 1) ALB: HTTP and HTTPS. - Layer 7 and application-aware. – Sticky sessions  – Support advanced request routing to specific web servers based on HTTP header  - X-Forwarded-For header: identify originating IP addr of client connecting through load balancer | | | | | |
| 2) NLB: TCP and high performance. – Layer 4. – Elastic/static IP addr  - Can handle millions of request per sec while maintaining ultra-low latencies. – Most expensive | | | | | |
| 3) Classic Load Balancer: HTTP/HTTPS and TCP. - Legacy. | | | | | |
| 4) Gateway load balancer: load balance workloads for 3rd party virtual apps running in AWS (virtual apps purchased using AWS Marketplace, virtual firewalls from companies like Fortinet, Palo Alto, Juniper, Cisco, IDP/IPS systems from companies like CheckPoint, trend Micro) | | | | | |
| Error 504 Gateway Timeout: target failed to respond | | | | | |
| Route 53 Hosted Zone: container for DNS records. - A Record: route traffic using IPv4 addr  - Alias: route traffic addressed to zone apex, or top of DNS namespace to resource within AWS. | | | | | |
| CLI Pagina-tion | Can control num of items included in output when you run a CLI command  By default, CLI uses page size of 1000, e.g. if run *aws s3api list-objects my\_bucket* on bucket which contains 2500 objects, CLI makes 3 API calls to S3 but displays entire output in 1 go  - If error when running list commands on large num of resources, default page size might be too high, so get timed out error  - Use the *--page-size* option to request smaller num of items in each API call  - *max-items* same as LIMIT in SQL | | | | | |
| RDS | RDS for OLTP. RDS types: SQL Server, PostgreSQL, Oracle, MySQL, MariaDB, Aurora  Aurora: MySQL and PostgreSQL compatible with auto scaling  RDS can be multi-AZ, failover capability, automated backups | | | | | |
| OLTP: processes data from transactions in real-time  Data processing and completing large num of small transactions in real-time | | | | OLAP: processes complex queries to analyze historical data  Data analysis using large amts of data and complex queries that take a long time to complete | |
| RDS Multi-AZ and Read Replicas | Multi-AZ: exact copy of your DB in another AZ (primary and standby)  - Standby instance not visible by users. – RDS will auto failover to standby  - Multi-AZ for Disaster Recovery (DR) not for improving performance | | | | | |
| Read Replica: read-only copy of primary DB  - Good for read-heavy workloads and take load off primary DB. – Can be in same or cross AZ or cross region  - Each read replica has its own DNS endpoint. – Need enable auto backup in order to deploy read replica  - Read replicas can be promoted to be own DB but this breaks the replication  - Can add up to 5 read replicas to each DB instance | | | | | |
| RDS Backups & Snapshot | 1) Database snapshot: Manual, ad-hoc and user-initiated. Provides snapshot of storage volume attached to DB instance  2) Automated Backups: enabled by default. Creates daily backups or snapshots that run during a backup window you define. Transaction logs used to replay transactions | | | | | |
| Automated backup: - backups stored in S3. – free storage space equal to size of DB  - During backup window, storage I/O may be suspended for a few seconds while backup process initialize, which might cause increased latency  1) Point-in-Time recovery: recover DB to any point in time within a retention period of 1-35 days.  2) Full daily backup: takes full daily backup/snapshot and stores transaction logs throughout the day.  3) Recovery process: use most recent daily backup and apply transaction logs relevant to that day, up to recovery point | | | | | |
| Database snapshot: - not automated. – No retention period: snapshots not deleted even after you delete the original RDS instance, including any automated backups  - Back up DB instance in a known state as frequently as you wish, and then restore to that specific state at any time | | | | | |
| Restored version of DB will always be a new RDS instance w new DNS endpoint (backup or snapshot)  - Enable encryption at creation time by selecting encryption option in console  - Encryption uses AWS KMS w AES-256 encryption  - Using encryption will encrypt all underlying storage, automated backups, snapshots, logs & read replicas  - Cannot enable encryption on an unencrypted RDS DB instance  -- To do so, unencrypted DB -> unencrypted snapshot -> encrypted snapshot -> encrypted DB | | | | | |
| RDS Proxy | Apps -> RDS Proxy -> RDS.  – Help pools and shares DB connections to assist w app scalability and DB efficiency  - Serverless and scales automatically. – Preserves app connections during failover of DB  - Detects failover and routes requests to standby quickly  - Deployable over Multi-AZ for protection from infra failures. – Up to 66% faster failover times | | | | | |
| ElastiCache | | | 1) In-memory Cache (Key Value): easy to deploy, operate, scale an in-memory cache in cloud  2) Improves DB performance: retrieve info from fast, in-memory caches instead of slower disk-based storage  3) Great for read-heavy DB workloads: cache results of I/O intensive DB queries. Also for storing session data of distributed apps | | | |
| 1) Memcached: for basic object caching  - Scales horizontally, but no persistence, multi-AZ or failover  2) Redis: more sophisticated solution w features like persistence, replication, multi-AZ, failover  - Supports sorting, ranking data (e.g. for gaming leaderboards) & complex data types like lists, hashes | | | |
| MemoryDB for Redis | | In-memory DB (GB to > 100TB). – Highly available: multi-AZ, transaction log for recovery and durability  - Primary DB: entire dataset can be stored in memory (instead of DB + cache)  - Ultra-fast performance: supports > 160 million requests per second. Microsecond read and single-digit millisecond write latency | | | | |
| - For workloads requiring an ultra-fast, Redis-compatible primary DB  - For High performance apps that need an in-memory DB to handle millions of requests per sec  - For apps that are data-intensive, low-latency that require high scalability  - For Highly scalable microservices architectures | | | | |
| ElastiCache for Redis  In-memory DB cache service. Sits in front of a DB  Millisecond read latency  E.g. website that needs store session data for its customers | | | | MemoryDB for Redis  Primary DB. Reduce complexity by removing DB + cache  Microsecond read, single-digit millisecond write  E.g. online gaming company w millions of users sharing digital assets |
| Systems Manager Parameter Store | | Store parameters used by apps, e.g. license keys, DB connection info, usernames, passwords  Info is passed to EC2 as a bootstrap script  - To maintain confidentiality of info and avoid hardcoding parameters in code  - Can encrypt parameters w AWS KMS keys  - Integrated w AWS services like EC2, CloudFormation, Lambda, CodeBuild, CodePipeline, CodeDeploy | | | | |
| AWS Secrets Manager | Centrally manage secrets used to access resources inside and outside AWS  Rotate secrets w/o code deployment. - Secure w control of fine-grain permissions and encryption w KMS | | | | | |
| Secret Types: RDS, Redshift, DocumentDB, other DB, API Keys  For DB: can store username and password, server address, DB name and port | | | | | |
| Secrets encrypted using KMS: use default Secrets Manager KMS Key or Customer Master Key (CMK)  CMK: logical representation of a master key, which holds key material to encrypt data. CMKs can only encrypt up to 4KB of data  Automatic rotation of secrets can be done every custom number of days (up to 365)  - Create new or use existing Lambda for secret rotation . – Can create read replica of secret in other region | | | | | |
| Secrets Manager  DB credentials, API Keys, Rotation of keys | | | | Systems Manager Parameter Store  Wider use cases, config variables, license keys | |
| EC2 Image Builder | Create EC2 images: virtual machine images (AMIs) & container images. - Simple to use: graphical interface. – Validate your images: test and validates images for security compliance, functionality, using AWS provided tests or own custom tests  EC2 Image Builder automates process of creating and maintaining images  - When software updates available, Image Builder auto create new image, run validation tests and make it available in AWS regions of your choice  - Allows sharing of AMIs w other AWS accts you own | | | | | |
| 1) Base OS: provide base OS image, e.g. Amazon Linux 2 AMI  2) Software: define software to install, e.g. .NET, Node.js, Python, latest security updates, latest kernel, security settings  3) Test: run tests on new image  4) Distribute: distribute image to regions of your choice (default = region you are in) | | | | | |
| Terminology: - Image pipeline = define the config and end-to-end process of building images. Includes image recipe, distribution, test settings  - Image recipe = creates a recipe for each image, which can be shared, version controlled, and re-used (Source image, e.g. Amazon Linux 2 AMI + Build Components, e.g. Apache Tomcat)  - Build Components: software components to include in image | | | | | |
| IAM role policies needed: EC2 Instance Profile for Image Builder + Amazon SSM Managed Instance Core  as Image Builder underlying use SSM | | | | | |
| Using AMIs in a diff region | AMIs are regional: can only be used in the region it belongs to  To use a AMI in a diff region, need create a copy in that region  When copying AMI, can select dest Region and whether to encrypt AMI copy  - copy encrypted AMI to unencrypted AMI not possible, all other scenarios are possible  - By default, AMI is copied w its original encryption state | | | | | |
| Extra | Customer want to route inbound traffic to api.customersite.net to Compute Optimized EC2 and inbound traffic to mobile.customersite.net to Memory Optimized EC2.  - Use ALB w host-based routing rules to forward traffic to instances  - Host-based routing rules to route traffic by subdomain names (X.site.net and Y.site.net). Path-based routing takes into account path in URL (/2018/July/blogs). Proxy servers can work but less efficient  EC2 need data on whether they are running on a GPU instance or local IP addr of instance  - Access instance metadata from running instance  EC2 instance in standby state:  - Billed the same way as instances that are in-service. Auto registered w ELB or target group after returned to service. Auto scaling don’t perform health checks on instances in standby state  Apps use several secrets in AWS Secrets Manager. Need identify secrets that are still in use  - Create a secretsmanager-secret-unused AWS Config managed rule. Create EventBridge rule to initiate notifications when AWS Config managed rule is met | | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S3 | | | | |
| Intro | S3: secure, durable, highly scalable object storage at a very low cost  - Manages data as objects rather than in file systems or data blocks  - Unlimited storage. – S3 objects can range in size from 0 bytes to 5 terabytes  - Stores files in buckets (similar to folders)  - Universal namespace: all S3 bucket name must be globally unique  - Example S3 URL: https://bucket-name.s3.Region.amazonaws.com/key-name  - Key (name of object, e.g. test.jpg) + Value (data itself = seq of bytes) + version ID + Metadata (last-modified…)  - Data is spread across multiple devices and facilities to ensure availability and durability  - 99.95% - 99.99% service availability depending on S3 tier  - 99.999999999% (11 9’s) durability for data stored in S3  - Tiered storage for diff use cases. – Lifecycle management. – Versioning (can even retrieve deleted obj) | | | |
| 1) Server-Side Encryption: set default encryption on bucket to encrypt all new objs when stored in bucket  2) Access Control Lists (ACLs): define which AWS accts or groups are granted access and type of access. Can attach S3 ACLs to individual objs within a bucket  3) Bucket Policies: specify what actions are allowed or denied | | | |
| S3 Storage Classes | | 1) S3 Standard: data stored redundantly across multiple devices in multiple facilities  - For frequently accessed data. – Highest cost  – Suitable for most workloads (websites, content distribution, mobile and gaming apps, big data analytics) | | |
| 2) S3 Standard Infrequent Access (S3-IA): for infrequently accessed data  - Used for data that is accessed less frequently but requires rapid access when needed  - Low per-GB storage price and a per-GB retrieval fee.- For long-term storage, backups, DR files. | | |
| 3) S3 One Zone-Infrequent Access: like S3-IA but data stored redundantly within a single AZ  - 20% less than regular S3-IA. – For long-lived, infrequently accessed, non-critical data | | |
| 4) Glacier: very cheap storage. – Optimized for data that is very infrequently accessed  - Pay each time you access data. – Use only for archiving data.  a) Glacier Instant Retrieval: long-lived data, access ≈ once per quarter.  b) Glacier Flexible Retrieval: long-term data archiving that occasionally needs to be accessed | | |
| 5) Glacier Deep Archive: long-term archiving for rarely accessed data  – For financial records that may be access once or twice a year or none | | |
| 6) S3 Intelligent-Tiering: 2 tiers, frequent and infrequent access. – Use if don’t know access patterns  - Auto moves data to most cost-effective tier based on how frequently you access each obj  - Charges a extra monthly fee of $0.0025 per 1000 obj on top of storage costs | | |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Storage Class | Availability / Durability | AZ | Min storage duration | Retrieval time | | S3 Standard | 99.99% / 11 9’s | ≥ 3 |  |  | | S3-IA | 99.9% / 11 9’s | ≥ 3 | 30 days |  | | S3 One Zone-IA | 99.5% / 11 9’s | 1 | 30 days |  | | Glacier Instant Retrieval | 99.9% / 11 9’s | ≥ 3 | 90 days | millisecond | | Glacier Flexible Retrieval | 99.99% / 11 9’s | ≥ 3 | 90 days | mins - hours | | Glacier Deep Archive | 99.99% / 11 9’s | ≥ 3 | 180 days | 12 hours | | Intelligent-Tiering | 99.9% / 11 9’s | ≥ 3 | 30 days |  | | | |
| Secure S3 buckets | | By default, 1) newly created buckets are private, 2) only bucket owner can upload, read, delete files, 3) no public access | | |
| Bucket policies: set up access control to buckets and applied at bucket level  - Cannot attach bucket policy to an individual object  - Can grant access to a group of files which need to be accessed by the same people  - Written in JSON as key value pairs. Impt keys: Effect (allow, deny), Principal, Action (s3:GetObject), Resource (arn:aws:s3:::mybucket/\*)  - To enable public access, 1) Turn off Block public access, 2) Bucket policy set allow to \* | | |
| Bucket ACLs: applied at object level. – Can apply diff permissions for diff objs within a bucket | | |
| S3 Access Logs: log all requests made to S3 bucket. – Not enabled by default. – Logs written to another S3 | | |
| S3 object owner can temporarily share objs w users outside their AWS acct, even those w/o AWS acct by creating a presigned URL  - Grant temp permissions to access S3 objects | | |
| S3 Hosting | | | Can host static website. Set index.html and error.html | |
| S3 Encryption | | | 1) Encryption in Transit: SSL/TLS or HTTPS  2) Encryption at Rest: Server-Side Encryption  a) SSE-S3: S3 managed keys, using AES-256 bit encryption (default when create bucket)  b) SSE-KMS: AWS KMS-managed keys  c) SSE-C: Customer-provided keys  3) Encryption at Rest: Client-Side Encryption: encrypt file yourself before upload to S3 | |
| - Enforce using Bucket policy: explicitly deny requests that don’t include *x-amz-server-side-encryption* parameter in request header. (StringNotEquals AES256)  - Deny requests that do not use aws:SecureTransport to enforce use of HTTPS/SSL (Bool false) | |
| Cross-Origin Resource Sharing (CORS) | | | | Allow resources in 1 S3 bucket to access resources in another S3 bucket  - Need ‘Access-Control-Allow-Origin’ header  - In S3 bucket that allow others to access, under CORS config, in JSON: [{“AllowedHeaders”: [“Authorization], “AllowedMethods”: [“GET”], “AllowedOrigins”: [“source bucket url”]}] |
| CloudFront | | | Content Delivery Network (CDN): system of distributed servers which deliver webpages and other web content. – Easy and cost effective way to distribute content w low latency and high data transfer speed  Uses edge locations: collection of servers in geographically dispersed data centers to keep cache of copies of objects  - Edge location will forward request from end users, download file requested and cache them locally  - Global network of 200+ edge locations  - Requests for content auto routed to nearest edge location, so content delivered w best performance  - Can configure restrict viewer access: viewers must use CloudFront signed URLs or signed cookies to access content (E.g. paid content only viewed by users w signed URL)  - Can add edge function to customize how CloudFront respond to HTTP requests  - Can add AWS WAF web ACL  - Can set geographic restriction (allow list or block list to countries) | |
| CloudFront Origin: origin of all files that distribution will serve. Can be a S3 bucket, EC2, ELB, Route53  - Can also work w any non-AWS Origin server, which stores the original, definitive versions of files  CloudFront Distribution: name given to Origin and config settings for the content to distribute using CDN | |
| Time to Live (TTL): period of time which objs are cached for. - Default TTL is 1 day.  – Can clear obj from cache before TTL is up, but you will be charged (use Invalidation) | |
| S3 Transfer Acceleration: enables fast, easy, secure transfer of files over long distances btw end users and S3 bucket using CloudFront CDN  - As data arrives at edge location, it is routed to S3, over an optimized network path | |
| - Can configure Bucket to restrict access to only CloudFront (Origin Access Identity)  - OAI = special CloudFront user that can access files in S3 and serve to users  - In S3 bucket, update bucket policy to allow OAI, and turn off enable all public access | |
| CloudFront AllowedMethods: which HTTP methods your CloudFront distribution will support  1) GET, HEAD. 2) GET, HEAD, OPTIONS. 3) GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE   |  |  | | --- | --- | | GET | Read data | | HEAD | Inspect resource headers; similar to GET but w/o the response body | | PUT | Send data to create new resource or replace an existing resource. Idempotent | | PATCH | Partially modify a resource | | POST | Insert data; used to create or update a resource. Not idempotent | | DELETE | Delete data | | OPTIONS | To find out which other HTTP methods are supported by the given URL | | |
| Athena | | To run standard SQL queries on data stored in S3. – Serverless, pay per query/per TB scanned  - Query log files stored in S3 (e.g. ELB logs, S3 access logs). – Analyze AWS Cost and Usage reports  - Generate business reports on data stored in S3. – Run queries on click-stream data | | |
| - Create S3 bucket to store Athena result  - CREATE DATABASE, then CREATE TABLE … LOCATION ‘s3://…’, then run queries | | |
| AWS Certificate Management | | | | AWS Certificate Management: create and manage SSL/TLS certs for securing website  Enables secure connection to website using HTTPS  - When using ACM w CloudFront, certificate must be created in us-east-1 Region  - Ensure that alternate domain name (CNAME) in the distribution settings matches the domain name in the cert so that app in other regions can use cert  - SSL/TLS cert from 3rd party provider also have to follow same procedure to be use w CloudFront |
| Extra | | Mtds of access control can be implemented using S3 bucket policies:  - Control access based on CIDR block, IP addr, based on time of day  Encrypt obj in S3 using server-side encryption but don’t want to use AES-256 key by S3  - Admin should send keys and encryption algo info w ea API call  Web app that collects confidential user data through POST request. Web app served through CloudFront. User names and phone nums must be encrypted at edge and remain encrypted throughout app stack  - Use field-level encryption on CloudFront. Field-level encryption adds an additional layer of security along w HTTPS. Sensitive info is encrypted at edge closer to user and remain encrypted throughout, ensuring only apps that need the data and have the credentials to decrypt can do so  Need asymmetric encryption key:  - SSE-S3, SSE-KMS, SSE-C only support symmetric key. Need use client-side encryption w customer managed encryption key that allow selecting either symmetric or asymmetric keys  To use SSE-C:  - need provide S3 object key and encryption key in API call | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Intro to Serverless Computing | | | | | |
| Basics | AWS handles infra management: Capacity provisioning, patching of OS, auto scaling, HA  Competitive advantage: focus on your app, super scalable, faster speed to market, lower cost (only charged when code is executed)  1) Lambda. 2) SQS. 3) SNS. 4) API Gateway: create, publish, secure APIs at any scale. 5) DynamoDB. 6) S3 | | | | |
| Lambda | Serverless Compute: run code in AWS w/o provisioning any servers. - Auto-scaling and HA  Supported Languages: Java, Go, PowerShell, Node.js, C#, Python, Ruby  - Charged based on num of requests, duration and amt of memory used by Lambda functions  - Requests: 1st 1 million requests per month are free. $0.2 per month per subsequent 1 million requests  - Duration: charged in 1 millisecond increments. Price depends on amt of memory allocated to Lambda  - Price per GB-second: $0.00001667 per GB-second. 1st 400,000 GB-seconds per month free | | | | |
| Event-Driven Architecture: lambda functions can be auto triggered by other AWS services or called directly from any web/mobile app  - Events can be changes made to data in S3 bucket or DynamoDB table  - Can use API Gateway to configure an HTTP endpoint, allowing you to trigger your fn at any time using an HTTP request  Lambda Triggers: DynamoDB, Kinesis, S3, ALB, API Gateway, CloudFront, Alexa, SQS, SNS, SES, CloudFormation, CloudWatch, CodeCommit, CodePipeline, … | | | | |
| Need attach execution role to Lambda to give it permissions  - Default role only allow sending logs to CloudWatch | | | | |
| When you create a Lambda function, there is only 1 version: $LATEST (alias)  - Can create multiple versions of function code and use aliases to reference version you want to use (e.g. arn:aws:lambda:<region>:<account id>:function:<lambda name>:<alias> )  - Create version of function (i.e snapshot of function), then create alias that point to that version | | | | |
| Concurrent Execution Limits: safety feature to limit num of concurrent executions across all functions in a given region per acct  - Default 1000 per region. – If hit limit, will get TooManyRequestsException, w status code 429  - Can incr limit but need contact AWS support  - Reserved concurrency guarantees that a set num of execution will always be available for your critical function, but this also acts as a limit (reserved from the 1000) | | | | |
| Can enable Lambda to access resources in a private VPC  1) Provide VPC config info to function: private subnet ID and security group ID  2) Lambda configures an ENI, using an IP from the private subnet CIDR range | | | | |
| API Gateway | Publish, Maintain, Monitor, Secure APIs at any scale  Supported API Types: 1) RESTful APIs (optimized for stateless, serverless workloads). 2) Websocket APIs (for real-time, 2 way, stateful communication, e.g. chat apps)  API Gateway provides a single endpoint for all client traffic interacting w backend of your app  - Allows connecting to apps running on Lambda, EC2, Elastic Beanstalk, or services like DynamoDB, Kinesis  - Supports multiple endpoints and targets: send each API endpoint to a diff target  - Supports multiple versions: can maintain multiple versions of your API, so can have diff versions for dev, testing, prod env  - Serverless: cost effective and scalable. – Integrate w CloudWatch to log API calls, latencies, error rates  - Help manage traffic w throttling so backend can withstand traffic spikes and denial of service attacks | | | | |
| RESTful APIs: Representational State Transfer. – Optimized for serverless and web apps. – Stateless  - Supports JSON (JavaScript Object Notation; uses key-value pairs) | | | | |
| - In API Gateway, REST API have more features than HTTP API  - Enable CORS to be able to call API from a webpage that isn’t hosted on the same domain  - Adds *Access-Control-Allow-Origin* header to all responses | | | | |
| Serverless Architectures | | | | Event-Driven Architecture:  1) Asynchronous: events and asynchronous communication used to loosely couple app components. An event or message might trigger an action, but no response, expected or required  2) Loosely Coupled: services and components operate and scale independently of each other  3) Single-Purpose functions: stateless functions performing a short-lived task | |
| Step Functions | | Visual interface for serverless apps, which enables you to build, run serverless apps as a series of steps  - Output of 1 step may act as input to next.  – Orchestrate steps, including sequencing, error handling, retry logic  - Also logs step of each step, so can quickly diagnose and debug problems | | | |
| - Step Functions consists of State Machine (workflow itself) and Tasks (single step)  - Parallel steps possible as well. – Branching workflow possible as well | | | |
| Step Functions Workflows | | | 1) Standard Workflows: long running, durable and auditable workflows that may run for up to a year.  - Full execution history available for up to 90 days after completion  - At-Most-Once model: tasks are never executed more than once unless you specify retry actions  - Non-Idempotent actions: e.g. when processing payments, only want to process once, not multiple  - A request is non-idempotent if it always causes a change in state (e.g. sending same email multiple times causes a change in state as you end up w multiple emails in inbox) | | |
| 2) Express Workflows: short-lived of up to 5 minutes. – For high-volume, event-processing workloads  - At least-once model: ideal if there is a possibility that an execution might be run more than once or you require multiple concurrent executions  - Idempotent actions: e.g. transforming input data and storing result in DynamoDB  - Request is idempotent if an identical request can be made once or several times in a row w no additional side effects (e.g. reading data from DB or S3) | | |
| 2a) Synchronous Express Workflows:  Begins a workflow  Waits until it completes  Returns result  Great for operations that are performed one at a time. Workflow must complete before next step begins | | 2b) Asynchronous Express Workflows:  Begins a workflow  Confirms workflow has started  Result of workflow stored in CloudWatch logs  Great if services or operations don’t depend on completion and result of workflow |
| Ephemeral and Persistent Data Storage Patterns | | | Lambda is stateless: can’t permanently store any data in function (e.g. session data, customer data, …)  Lambda is ephemeral: not used for apps that need to run for longer than 15 minutes  To persist data, function must interact w a data store | | |
| Lambda data storage options: - Within Lambda: /tmp, Lambda layers  - External storage: S3, EFS, … | | |
| /tmp: temporary storage. – Provided in execution env of Lambda function. Default 512 MB, max 10GB  - Like a cached file system: data can be accessed by multiple invocations of your function sharing the same execution env in order to optimize performance  - Data is not persistent: available for lifetime of execution env | | |
| Additional libraries needed by function can be included in Lambda deployment package (ZIP file containing your code), but this will incr your deployment package size  Instead use Lambda Layers: best practice to add libraries and SDKs as layer that can be reference by multiple functions  - Or for large dependencies like image manipulation libraries, graphics libraries, or particular version of AWS SDK  - Deployment will be faster as ZIP file containing your code is smaller  - But if want change version of library included, cannot update dynamically, need create new layer and reference that | | |
| With S3, it is a object store, so can only store and retrieve objects. Not a file system  - Cannot directly open and write data to objects in S3  - To change data, need upload a completely new object/version | | |
| EFS: shared file system and acts like a file system. – Data is persistent and can be dynamically updated (e.g. open a file and write to it)  - Mounted by function when execution env is created. Can be shared across invocations  - To use EFS, Lambda function must be in same VPC as EFS file system | | |
| Lambda Environment variables & Parameters | | | | Env variables: adjust function behavior w/o changing code  - Use key-value pairs. – Env variables are locker when version of function is published | |
| Lambda configurable params:  1) General config: memory, ephemeral storage, fn timeout  2) Triggers. 3) Permissions: execution role  4) Function URL: HTTP(S) endpoint used to access your function using a browser  5) Tags: User-defined key-value pairs that help organize function (e.g. for cost reporting)  6) VPC: allow function to access resources that are in a customer VPC  7) Monitoring and Operations Tools: CloudWatch, CloudWatch Logs, X-Ray (performance issues)  8) Concurrency: - reserved concurrency to ensure critical function can always run and restricts other concurrent requests  - Provisioned concurrency lets function scale consistently w/o any fluctuations in latency  9) File Systems: EFS file systems that your function need to connect to. Need be in same VPC | |
| Lambda Event Lifecycle & Errors | Lambda Invocations: can invoke Lambda function synchronously or asynchronously  Synchronous: waits for response, service calling function will know Lambda completed successfully or not  Asynchronous: no acknowledgement, no notification if Lambda failed to complete successfully | | | | |
| Lambda Retries: default perform 2 retries  - Common errors: something went wrong in function code or function timed out  - Retry approach: waits 1 min before 1st retry, wait 2 mins before 2nd retry | | | | |
| Dead-Letter Queues (DLQs): save failed invocations for further processing. – Handles failures only  - Associated w a particular version of a function  - Can be an event source for a function, allowing you to reprocess events  SQS: Holds failed events in queue until they are retrieved  SNS: sends notifications about failed events to 1 or multiple destinations (fan out) | | | | |
| Lambda destinations: optionally configure Lambda to send invocation records to another service  Invocation success: send record to 1 destination when invocation was successfully processed  Invocation failure: send record to another dest when invocation and all retry attempts have failed  Supported destinations: SQS, SNS, Lambda (to trigger another Lambda function to start auto error-handling process), EventBridge (successful response can be used to trigger EventBridge event to successful invocations are tracked) | | | | |
| Lambda Deployment Packaging Options | | | | When creating a function using console, a .zip file containing the code you provide is auto created by Lambda. This .zip file is your deployment package (containing app code, dependencies)  - Can also create deployment package yourself and upload directly from local machine up to 50MB  - To upload deployment packages > 50 MB, upload .zip file to S3 in region where you create function. Then specify S3 object when creating function | |
| Lambda layers: Distribution mechanism: a .zip file archive that can be referenced by your function containing function dependencies like libraries, custom runtimes, …  - Layer can be used by multiple functions that have same dependencies  - Using layers reduce size of deployment package, enabling function to initialize faster | |
| Lambda Performance Tuning | | | | Control CPU capacity by configuring function memory: from 128 MB to 10,240 MB = 10 GB  - Adding memory will improve performance if function is either memory or CPU bound  - Adding memory may also reduce duration function runs for | |
| 1) First time function invoked, Lambda creates the execution environment  2) Configures memory, runtime, configs  3) Runs function static initialization code; import libraries and dependencies (add latency)  4) Runs function code | |
| To optimize static initialization, factors that contribute to latency  1) Amt of code that needs to run during initialization phase  2) Function package size: including imported libraries, dependencies, Lambda layers  - Avoid importing an entire SDK if don’t need to  3) Performance: libraries and other services that require connections to be set up | |
| Advanced API Gateway | | Can use API Gateway Import API feature to import an API using definition file  - Supported protocols: OpenAPI (formerly known as Swagger)  - Can use an OpenAPI definition file to create a new API or update an existing one | | | |
| Legacy Protocols: e.g. SOAP (Simple Object Access Protocol) which returns response in XML format  To deal w legacy apps using soap:  - Can configure API Gateway as a SOAP web service passthrough  - Can use API Gateway to transform the XML response to JSON | | | |
| API Gateway Mock Endpoints: allows developers to create, test, debug software  - Mimics/simulate responses and behaviors of a real API, also described as a mock integration  E.g. frontend done, backend not ready, use mock endpoints to simulate response from back to frontend | | | |
| API Gateway Stage: logical reference to the lifecycle state of the API (e.g. dev, prod, v3, …)  - Each stage can be associated w a diff endpoint (e.g. dev, prod, test env)  - Each stage has a unique invoke URL: e.g. https://jfd.execute-api.us-east-1.amazonaws.com/dev, https://jfd.execute-api.us-east-1.amazonaws.com/prod  - jfd = API ID, dev/prod = Stage Name | | | |
| Stage Variables: key-value pairs that act like env variables  - Changes behavior of API according to value of variable  - Use stage variable to configure the endpoint that a stage is going to route HTTP requests to  E.g. create 2 lambda function, 1 dev, 1 prod. In API Gateway create method, to specify Lambda function: ${stageVariables.lmbfunc}  - Add invoke permission to Lambda function to allow API Gateway to call execute-api on fn | | | |
| API Request & Response Transform-ations | | | Frontend -> API request -> API Gateway (can modify request parameters) -> API request -> Backend  Backend -> API request -> API Gateway (can modify response params) -> API request -> Frontend  - For HTTP APIs: param mapping is used to modify API requests and responses  - Specify API request or response params you want to modify  - Specify how you want to change the params  - For request: can change header, query string, request path.  – For response: can change header, status code | | |
| API Gateway Caching & Throttling | | Cache endpoints response, reducing num of calls made to endpoint and improve latency for requests to your API. – Default TTL is 300 seconds | | | |
| API Gateway Account Level Throttling: prevent API from being overwhelmed by too many requests  - Default limits steady-state request rate to 10,000 requests per second per Region  - Max concurrent requests is 5000 requests across all APIs per Region. – Can request incr for both limits  - If exceed either, will receive a 429 Too Many Requests error message | | | |
| AWS X-Ray | Tool to help analyze and debug distributed apps  - Allows troubleshoot root cause of performance issues and errors  - Provides a visualization of app’s underlying components  - X-Ray Service Map provides an end-to-end view of requests as they travel through your app  - Collects info like latency, HTTP status codes, errors  - Can integrate w AWS services like EC2, ECS, Lambda, Elastic Beanstalk, SNS, SQS, DynamoDB, ELB, API Gateway or own apps written in Java, Node.js, .NET, Go, Ruby, Python  - X-Ray SDK auto captures metadata for API calls made to AWS services using AWS SDK | | | | |
| 1) Install X-Ray agent/daemon on EC2 instance. 2) Configure your app using the X-Ray SDK  3) X-Ray SDK gathers info from request and response headers, code in app, metadata about the AWS resources on which it runs, and send this trace data to X-Ray  - X-Ray SDK sends data to X-Ray daemon which buffers segments in a queue and uploads them to X-Ray in batches. So need both X-Ray SDK and X-Ray daemon on your systems | | | | |
| When instrumenting your app, can record additional info about requests by using annotations  Annotations: simple key-value pairs that are indexed for use w filter expressions, so you can search for traces that contain specific data and group related traces tgt in console | | | | |
| Extra | For published version of Lambda function:  - code and config info cannot be changed  API Gateway time out even though Lambda finished under time limit. Which API Gateway metrics in CloudWatch can help troubleshoot?  - IntegrationLatency, Latency  To use Lambda in a VPC:  - Create Lambda functions inside VPC w AWSLambdaVPCAccessExecutionRole policy attached to Lambda execution role. (which include permissions like ec2:CreateNetworkInterface, ec2:DescribeNetworkInterfaces, ec2:DeleteNetworkInterface)  Deploy AWS SAM template in eu-west-1 Region, but creation of stack fails:  - Lambda@Edge functions can be created only in us-east-1 Region | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DynamoDB | | | | |
| Intro | | Fast and Flexible NoSQL DB. – Consistent, single-digit millisecond latency at any scale  - Fully Managed: supports key-value data models & Supported doc formats are JSON, HTML, XML  - Serverless, auto scale, integrates well w Lambda. - Uses SSD storage for fast performance.  – Data spread across 3 geographically distinct data centers for resilience.  - Choice of consistency models: eventually consistent reads (default) & strongly consistent reads  - Eventually consistent reads: consistency across all copies of data usually reached within a second (best for read performance)  - Strongly consistent reads: Always reflects all successful writes, i.e. writes reflected across all 3 locations at once (best for read consistency)  - Support for ACID transactions, aka DynamoDB Transactions | | |
| - DynamoDB table made up of items (records), each with attributes  - DynamoDB stores and retrieves data based on a primary key  Primary key can be 1) Partition Key or 2) Composite Key (partition key + sort key)  Partition Key based on a unique attribute  - Value of partition key is input to internal hash function which determines partition/physical location on which data is stored  Use Composite Key if partition key is not unique, but partition + sort is unique  - All items w same partition key stored tgt and then sorted according to the sort key value | | |
| DynamoDB access control | | | Use IAM users or roles to grant permission to access DynamoDB | |
| Restricting User Access: use a special IAM condition to restrict user access to only their own records  - Add condition to IAM policy when access granted only if partition key value matches their User\_ID  “Condition”: {“ForAllValues:StringEquals”: { “dynamodb:LeadingKeys”: [“${www.mygame.com:user\_id}”], }}  - Fine-grained access control w IAM possible | |
| DynamoDB Secondary Indexes | | | Allows flexible querying based on an attribute that is not the primary key by using global secondary indexes and local secondary indexes  - Secondary index allows you to perform fast queries on specific columns in a table. Select the cols you want included in the index and run searches on the index, rather than on entire dataset | |
| Local secondary index  - Same partition key as original table but diff sort key  - Gives diff view of data, organized according to an alternative sort key  - Queries based on this sort key much faster using the index than the main table  - Can only be created when you are creating the table. Cannot add, remove or modify later | |
| Global secondary index  - Diff partition key and diff sort key. - Gives completely diff view of data  - Speed up any queries relating to this alternative partition and sort key  - Can create when you create your table or add later | |
| Demo | | | Table class: DynamoDB Standard vs DynamoDB Standard-IA (infrequent access) | |
| Scan vs Query API call | | Query: finds items in a table based on the primary key attribute and a distinct value to search for  - Use optional sort key name and value to refine the results  - Can use ProjectionExpression param to return only specific attributes/cols you want  - Results always sorted by sort key in ascending numeric/character order by default  - To reverse order, set ScanIndexForward param to false  - By default, queries are eventually consistent, but can explicitly set query to be strongly consistent  Scan: examines every item in table. By default, returns all data attributes  - Use ProjectionExpression param to return only specific attributes/cols you want | | |
| Query is more efficient than a Scan.  - Scan “loads” the entire table, then filter out values to provide the desired result, removing unwanted data  - Scan operation on a large table can use up the provisioned throughput for a large table in a single operation | | |
| Scan operation processes data sequentially, returning 1MB increments before moving on to retrieve next 1 MB of data. – Scans 1 partition at a time. To improve scan performance:  1) Can configure DynamoDB to use parallel scans instead by logically dividing a table or index into segments and scanning each segment in parallel  - Best to avoid parallel scans if table or index already incurring heavy read/write activity from other apps  2) Isolate scan operations to specific tables and segregate them from your mission-critical traffic  - Even if this means writing data to 2 diff tables | | |
| To improve both query and scan performance: - Set a smaller page size.  – Running a larger num of smaller operations allow other requests to succeed w/o throttling  - Avoid scans in generally: design tables in a way that you can use the Query, Get or BatchGetItem APIs | | |
| DynamoDB Provisioned Throughput | | | Measured in Capacity Units: specify read and write capacity unit on creation of table  Write Capacity Unit (WCU): 1 WCU = 1 \* 1KB write per sec  RCU: 1 RCU = 1 \* strongly consistent read of 4KB per sec OR 2 \* eventually consistent reads of 4KB per sec (default) | |
| DynamoDB On-Demand Capacity | | | Charges apply for reading, writing and storing data  DynamoDB instantly scales up and down based on activity of app  - Great for unpredictable workloads, new apps where you don’t know usage pattern yet, when want to pay for only what you use. Otherwise use Provisioned Capacity (WCU + RCU) | |
| DynamoDB Accelerator (DAX) | | | Fully managed, clustered in-memory cache for DynamoDB. Ideal for read-heavy, bursty workloads  Up to 10x read performance improvement. Microsecond performance for millions of requests per sec  DAX is a write-through caching service: data written to cache and backend store (DynamoDB main table) at the same time  - Allows pointing DynamoDB API calls to DAX cluster  - If cache hit, DAX returns result, otherwise if cache miss, then DAX performs an eventually consistent GetItem operation against DynamoDB and return result of API call  - Not suitable for apps that requires strongly consistent reads, write-intensive apps, low read operations, don’t require microsecond response times | |
| DynamoDB TTL | | | Defines expiry time for data. Expired items marked for deletion  Great for removing irrelevant or old data (e.g. session data, event logs, temp data)  Reduces cost of table by auto removing data which is no longer relevant  - When current time > TTL, item will be expired and marked for deletion. Within 48 hours, will be deleted  - Can filter out expired items from queries and scans  - Need to have a column w epoch/Unix/POSIX time and set this attribute as TTL attribute | |
| DynamoDB Streams | | | Time ordered sequence of item level modifications (e.g. insert, update, delete)  Logs encrypted at rest and stored for 24 hours  Streams accessed using a dedicated endpoint. - By default, primary key is recorded  - Before and after images can be captured.  - Use cases: audit/archive transactions, trigger an event based on a particular transaction, or replicate data across multiple tables | |
| Provisioned Throughput Exceeded and Exponential Backoff | | | | ProvisionedThroughputExceededException: request rate is too high for read/write capacity provisioned on DynamoDB table  AWS SDK will auto retry requests until successful (auto use exponential backoff)  If not using AWS SDK, need reduce your request freq or use exponential backoff |
| Extra | Multiple updates may occur simultaneously and cause original editor’s changes to be overwritten w/o a proper review process. How to prevent this in DynamoDB write option?  - Use conditional writes: conditional write succeed only if item attributes meets ≥ 1 expected conditions. Else return an error.  DynamoDB stream KEYS\_ONLY events vs NEW\_IMAGE events  - KEYS\_ONLY events records only partition and sort keys of changed DynamoDB items while NEW\_IMAGE events contain entire DB item | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| KMS and Encryption on AWS | | | |
| Intro | | Managed service to create and control encryption keys used to encrypt data  Integrated w many AWS services to make encrypting data in those services (S3, RDS, DynamoDB, Lambda, EBS, EFS, CloudTrail, AWS Developer Tools, …)  Customer Master Key (CMK): encrypt/decrypt data up to 4 KB  - To generate/encrypt/decrypt Data key. – Data key then used to encrypt/decrypt data. – Envelope encryption | |
| CMK | | - Symmetric vs Asymmetric key (public, private keys)  - Key material origin: KMS, External, Custom Key Store (CloudHSM = dedicated tenant for you, more ex)  - Single or Multi-Region key. – Key policy: resource based policy (who can use KMS) similar to IAM policy  - Can use alias when using CMK. – Need specify who can administer key & who can use key | |
| KMS API | aws kms encrypt --key-id KEYID –plaintext fileb://secret.txt --output text –query CiphertextBlob | base64 –decode > encryptedsecret.txt # Encrypts plaintext into ciphertext by using CMK  aws kms decrypt --ciphertext-blob filb://encryptedfile.txt --output text –query Plaintext | base64 –decode > decryptedsecret.txt # Decrypts ciphertext that was encrypted by CMK  aws kms re-encrypt --destination-key-id KEYID --ciphertext-blob fileb://encryptedsecret.txt | base64 > newencryption.txt # Decrypt ciphertext and re-encrypt file w a diff CMK  aws kms enable-key-rotation --key-id KEYID # Enable auto key rotation every 365 days  # Previous versions of key is saved so that you can still decrypt files that were previously encrypted  aws kms generate-data-key --key-id KEYID --key-spec AES\_256. # To generate data key to encrypt > 4KB data | | |
| Certificate Management | | | AWS Certificate Manager (ACM): create and manage public and private Secure Sockets Layer (SSL)/Transport Layer Security (TLS) certificates  - Can be used w AWS services like ELB, CloudFront, API Gateway, web apps  SSL/TLS certs: digital certs used to verify authenticity of website  - Enables secure connection btw visitors and website  - Used during encryption process to encrypt data in transit  - To use ACM w CloudFront, must create cert in US-East-1 region |
| Extra | App encrypt data to disk by using AWS KMS. App will decrypt file when user requests to download it. Dev needs to use the GenerateDataKey API to encrypt PDF file. Dev needs to use an AWS KMS symmetric customer managed key for encryption  - Write encrypted key from the GenerateDataKey API to disk for later use. Use plaintext key from the GenerateDataKey API and a symmetric encryption algo to encrypt the file  If use AWS Encryption SDK to encrypt an object  - Must decrypt using Encryption SDK, cannot use S3 encryption client, vice versa | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Other AWS Services | | | | | |
| SQS | | Message queue service. Queue = temp repository for messages awaiting processing  Visibility Timeout: amount of time message is invisible in the SQS queue after a reader picks up that message (default 30 seconds, max 12 hours)  - Messages can contain up to 256 KB of text in any format (XML, JSON, unformatted text)  - Pull-based, not pushed-based. – Messages will be processed at least once  - Messages can be kept in queue from 1 min to 14 days (default retention is 4 days) | | | |
| 1) Standard queues (default): provide best effort ordering (might have duplicates)  - Nearly-unlimited num of transactions per sec. – Guarantees message is delivered at least once  - Best-effort Ordering: generally delivered in same order as they are sent. Occasionally (due to highly-distributed architecture that allows high throughput), more than 1 copy of a message might be delivered out of order  2) FIFO queues: order in which messages are sent and received is strictly preserved  - Exactly-Once processing (duplicates are not introduced).  - Limited to 300 transactions per second (TPS), but have all capabilities of standard queues | | | |
| Short Polling: returns a response immediately if message queue being polled is empty  - Can result in a lot of empty responses if nothing is in the queue  - Still pay for these responses  Long Polling: periodically polls the queue. – Saves money  - Doesn’t return a response until a message arrives in the message queue or long poll times out | | | |
| SQS Delay Queues: postpone delivery of new messages to a queue for a num of seconds  - Messages sent to Delay Queue remain invisible to consumers for duration of delay period  - Default delay is 0 second, max is 900 seconds = 15 min  - For standard queues, changing setting don’t affect delay of messages already in the queue, only new msgs  - For FIFO queues, this affects delay of msgs already in the queue  - Use Delay Queue for large distributed apps which may need to introduce a delay in processing (e.g. delay of few secs to allow updates to sales and stock DB before sending notification to customer confirming an online transaction) | | | |
| For large SQS messages: 256 KB up to 2GB in size, use S3 to store the messages  - Use Amazon SQS Extended Client Library for Java to manage them  -- Specify that msgs are always stored in S3 or only messages > 256 KB  -- Send a message which references a msg object in S3  -- Get a msg obj from S3. -- Delete a msg obj from S3  - Also need AWS SDK for Java: provides an API for S3 bucket and object operations  - AWS CLI, Management Console, SQS API, any other AWS SDK won’t allow SQS to store msg in S3 | | | |
| Cannot convert a standard queue to a FIFO queue. Must create new one | | | |
| SNS | | Notification service: push notifications to devices (Apple, Google, Fire OS, Windows, Android)  - SMS text message or email to SQS queues or any HTTP endpoint  - Can trigger Lambda functions to process info in msg, publish to another SNS topic or to another service  - Use a pub-sub model: apps publish/push messages to a TOPIC. Subscribers receives msg from a TOPIC  - Notifications are delivered using a push mechanism that eliminates need to periodically check or poll for new info and updates  Topic: access point, allowing recipients to subscribe to and receive identical copies of the same notification  - SNS delivers appropriately-formatted copies of the msg to each subscriber | | | |
| - Managed service. – Durable storage: prevents msgs from being lost  - All messages published to SNS are stored redundantly across multiple AZ  - Instantaneous: push-based deliver (no polling). - Pay as you go model w no up-front costs | | | |
| SES vs SNS | | SES: scalable and HA email service  - Help marketing teams and app devs send marketing, notification and transactional emails to their customers using a pay-as-you-go model  - Can use to receive emails w incoming mails delivered to an S3 bucket  - Incoming emails can be used to trigger Lambda and SNS | | | |
| SES: used for both incoming and outgoing email. Email addr required to start sending msgs  SNS: pub/sub message service for SMS, HTTP, SQS, email. Can fanout msgs to a large num of recipients (replicate and push messages to multiple endpoints and formats) | | | |
| Kinesis | | Family of services that enables you to collect, process, analyze streaming data in real time  Streaming data: data generated continuously by thousands of data sources that typically send in the data records simultaneously and in small sizes (kilobytes)  1) Kinesis Streams: stream data and video to allow you to build custom apps that processes data in real time. a) Data Streams. b) Video streams  2) Kinesis Data Firehose (KDF): capture, transform, and load data streams into AWS data stores to enable near real-time analytics w BI tools  3) Kinesis Data Analytics: analyze, query, transform streamed data in real time using standard SQL. Store results in an AWS data store | | | |
| KDS retain data by default for 24 hours, max 365 days retention. - Data is stored in shards.  - Kinesis stream is made up of ≥ 1 shard. Data capacity of stream determined by num of shards  - As data rate incr, need incr capacity of stream by incr num of shards  - Each shard is a seq of 1 or more data records and provides a fixed unit of capacity. Each shard gives 5 reads per sec (max total read rate is 2 MB per sec). 1000 writes per sec (max total write rate is 1 MB per sec)  - Each data record has a unique seq num. Order of records is maintained | | | |
| Kinesis Video Streams: securely stream video from connected devices to AWS | | | |
| KDF: no shards, capacity and sizing is automated. No need for consumers to consume the data  - Optionally, use Lambda function to analyze data in real time before sending data to S3  - No data retention. – Can save data to S3, OpenSearch | | | |
| KDA: run SQL queries on data from KDS or KDF and store results in S3, Redshift, OpenSearch | | | |
| Kinesis Consumers | | | | Kinesis Client Library runs on consumer instances  - Tracks num of shards in stream. – Discovers new shards when you reshard  KCL ensures for every shard, there is a record processor  - Manages num of record processors relative to num of shards & consumers  - If only 1 consumer, KCL will create all record processors on a single consumer  - If 2 consumer, it will load balance and create half the processors on 1 instance, and half on another  - Ensure num of instances don’t exceed num of shards (except for failure or standby purposes)  - Never need multiple instances to handle processing load of 1 shard  - 1 worker can process multiple shards  - Use CPU utilisation to determine quantity of consumer instances, NOT num of shards in Kinesis stream  - Use Auto Scaling group, and base scaling decisions on CPU load on consumers | |
| Elastic Beanstalk | | | Deploy and scale web apps w/o having to manage infra/EC2 instances  Supported Languages: Java, .NET, PHP, Node.js, Python, Ruby, Go  Support Platforms: Apache Tomcat, Docker. – Handles:  1) Infra: provision infra, load balancing, auto scaling, app health monitoring, RDS, S3  2) App platform: installation and management of app stack, including patching and updates to OS and app platform. 3) You still have complete admin control of AWS resources  - Fastest and simplest way to deploy app to AWS | | |
| 1) All at once: deploys to all hosts concurrently  - Will experience total outage. Not ideal for mission-critical production systems  - If update fails, need roll back changes by re-deploying original version to all instances, resulting in another outage to get back the previous version  2) Rolling: deploys new version in batches  - Each batch is taken out of service while deployment takes place  - Env capacity will be reduced by num of instances in a batch while deployment takes place  - Not ideal for performance sensitive systems  - If update fails, need perform an additional rolling update to roll back changes  3) Rolling w additional batch: launches an additional batch of instances. Then deploys new version in batches. - Maintain full capacity throughout the deployment  - If update fails, need perform an additional rolling update to roll back changes  4) Immutable: deploys new version to a fresh group of instances before deleting the old instances (blue/green). – Only when new instances pass health checks, should old instances be terminated  - If deployment fails, just delete new instances. – Preferred approach for mission critical systems  5) Traffic Splitting: Installs new version on new set of instances (like Immutable), but forwards a % of incoming client traffic to new app version for a specified evaluation period  - Enables canary testing. – If new instances stay healthy, Elastic Beanstalk forwards 100% of traffic to them and terminates old ones | | |
| For Amazon Linux 2: use Buildfile, Procfile and platform hooks to configure and run custom code  Buildfile: for commands that run for short periods, and then exit upon task completion  - Create Buildfile in root directory of app source  - Format: <process\_name>: <command> (e.g. make: ./build.sh)  Procfile: long-running app processes (e.g. commands to start and run app)  - Create Procfile in root directory of app source - Format: <process\_name>: <command> (e.g. web: bin/myserver OR app: bin/myapp)  - Elastic Beanstalk expects processes defined in Procfile to run continuously. It monitors and restarts any processes that terminate  Platform Hooks: custom scripts or executable files that you want Elastic Beanstalk to run at a chosen stage of the EC2 provisioning process  - Stored in dedicated directories in app source code  - .platform/hooks/prebuild: files for Elastic Beanstalk to run before it builds, set up and configure app and web server  - .platform/hooks/predeploy: after sets up and configures app & web server but before deploy to final runtime location  - .platform/hooks/postdeploy: files to run after deployed of app. Last deployment workflow step | | |
| 1) Launch RDS within Elastic Beanstalk  - Created within Elastic Beanstalk env. – If terminate env, DB will be terminated  - Good for dev and test, not for Prod  2) Launch RDS outside of Elastic Beanstalk  - Use RDS console or AWS CLI to create RDS DB instead of Elastic Beanstalk  - Allows tear down of app env w/o affecting DB instance  - To connect to an outside DB: 1) Need to add additional Security Group to environments Auto-Scaling grp  2) Need provide connection string info to app servers using Elastic Beanstalk env properties | | |
| Parameters: input custom values to CloudFormation template  Outputs: export selected values from CloudFormation stack. Can import the values into other stacks or display values after you create/update a stack  Mappings: matches a key to a corresponding set of named values (e.g. specify AMI ID for ea Region)  Resources: defines stack resources and properties | | |
| Migrating Apps to Elastic Beanstalk | | | | | Windows Web Application Migration Assistant (formerly known as .NET Migration Assistant)  - Interactive PowerShell Utility: enables migration of .NET app, or entire website from Windows server to Elastic Beanstalk. - Open-source |
| Extra | SQS actions:  - SetQueueAttributes and ReceiveMessage: can configure short polling  - PurgeQueue: delete msg. – SendMessageBatch: send msg. – ChangeMessageVisibility  To decouple DB instance from already deployed Elastic Beanstalk env  1) Create RDS DB snapshot. 2) Protect RDS DB instance from deletion. 3) Create new Elastic Beanstalk env  4) Perform a blue/green deployment. 5) Update DB deletion policy for beanstalk env A.  6) Decouple RDS instance from beanstalk env A. 7) Terminate old Elastic Beanstalk env  SQS btw producers and Lambda function. Alert report some “OverLimit” error messages  - “OverLimit” when too many inflight messages. In-flight msg = msg received by consumer and not deleted yet from queue  - To solve: delete processed messages from queue and raise quota of inflight msg limit from AWS support  Config specs for Elastic Beanstalk specify health checks, but not being run  - Config file need to be saved in the .ebextensions folder w the .config extension.  Dev runs CLI command on EC2 instance. Command fails w response, *A client error (UnauthorizedOperation) occurred when calling the …. Encoded authorization failure message: mVqaktia1h2cD…*. How to decode response  - Use AWS Security Token Service (AWS STS) decode-authorization-message command to decode contents of response. Certain AWS operations return an encoded authorization message, so need STS | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Developer Theory | | | | |
| CI/CD | Continuous Integration, Continuous Delivery/Deployment  - Small incremental changes. Automate as much as possible (code integration, build, test, deploy)  Continuous delivery: human decision to deploy code  Continuous deployment: system auto deploys code  1) CodeCommit: source & version control  2) CodeBuild: automated build. Compiles source code, runs test and produces packages ready to deploy  3) CodeDeploy: Automates code deployments to any instance (EC2, Lambda, on-prem)  4) CodePipeline: manages the workflow. End-to-end solution, build, test, deploy app every time there is code change | | | |
| Code-Commit | Central code repo. Manages updates from multiple sources. Enables collaboration  - Tracks and manages code changes. Maintain version history | | | |
| Code-Deploy | 1) In-Place/Rolling Update: app is stopped on each instance and new release is installed  - Instance will be out of service during deployment so capacity reduced. Configure ELB to stop sending requests to the instance  - CodeDeploy installs new version, known as Revision  - Need to re-deploy previous version for roll back which can be time-consuming. – Less ex  - Run order of hooks: App stop -> Before install -> After install -> App start  2) Blue/Green: New instances provisioned and new release installed on new instance. Blue = active deployment, green = new release  - New Revision deployed to Green env. – Green instances registered w ELB  - Blue env terminated. – To roll back, just set ELB to divert traffic back to Blue env. – More ex | | | |
| AppSpec File: config file to define params used during a CodeDeploy deployment. – Put in root directory  - For EC2 and on-prem systems: YAML  - For Lambda: YAML and JSON supported. File structure depends on deploy to Lambda or EC2  EC2 AppSpec File Structure:  1) Version: reserved for future use. Currently only allowed value is 0.0  2) OS version: e.g. linux, windows  3) Files: Location of any app/config files/packages that needs to be copied & where they should be copied to  4) Lifecycle event hooks: scripts which needs to run at set points in deployment lifecycle (unzip files, run functional tests, de-register and re-register instances w ELB) | | | |
| Lifecycle event hooks run in a specific order known as the Run Order. For In-place deployment:  BeforeBlockTraffic: tasks to run before de-register from Load Balancer  BlockTraffic: de-register instances from LB  AfterBlockTraffic: tasks to run after de-register from LB  ApplicationStop: gracefully stop app  DownloadBundle: CodeDeploy agent copies app revision files to temp location  BeforeInstall: pre-installation scripts, e.g. backing up curr version, decrypting files  Install: copy app revision files to final location  AfterInstall: post-install scripts, e.g. config, file permissions  ApplicationStart: start any services that were stopped during ApplicationStop  ValidateService: run tests to validate service  BeforeAllowTraffic: tasks to run on instances before registered w LB  AllowTraffic: register instances w LB  AfterAllowTraffic: tasks to run on instances after registered w LB | | | |
| Code-Pipeline | Fully managed CI/CD services: 1) Orchestrates build, test & deployment (triggered when change in code)  2) Automated Release process: fast, consistent, fewer mistakes. Quick release of new features & bug fixes  - Integrates w CodeCommit, CodeBuild, CodeDeploy, Github, Jenkins, Elastic Beanstalk, CloudFormation, Lambda, ECS | | | |
| Code-Artifact | Artifact repo for devs to find software packages they need  - Securely store, publish and share software packages used in software dev process  - Package = bundle of software. – Can have open-source software and in-house software  - Integrates w npm registry, Python Package Index, Maven Central, CodeBuild  - Artifacts = documentation, compiled apps, deployable packages, libraries  - IT leaders can make approved packages available. E.g. version that engineering team is willing to support | | | |
| 1) Create a CodeArtifact domain. Domain = group repos to manage them  2) Create repo in domain. 3) Create an upstream repo. 4) Add external connection to upstream repo to pull packages from an external public repo. 5) Associate the upstream repo w your repo | | | |
| ECS | Containers: similar to virtual machine  - Standardized unit w everything the software needs to run (e.g. libraries, system tools, code, runtime)  - Use Docker for Linux containers and Windows Containers for Windows workloads  ECS: container orchestration service which supports Docker and Windows Containers  - Quickly deploy and scale containerized workloads w/o having to install, configure, manage and scale own orchestration platform | | | |
| 1) Clusters of Virtual Machines: run containers on clusters of virtual machines (EC2). More control  2) Fargate for Serverless: don’t need to worry about EC2 instances | | | |
| Elastic Container Registry (ECR) : store container images to be use for ECS | | | |
| Cloud-Formation | | Manage, configure, provision AWS infra as code. – Supports YAML or JSON  - Resources defined using CloudFormation template. CF makes appropriate API calls to create resource you defines. – Consistent. – Quick and Efficient. – Version Control.  – Free to use, only charged for AWS resources you create. – Manage updates. – Roll back  - To prevent CF from deleting successfully provisioned resources during a stack create operation, while allowing resources in failed state to be updated or deleted upon the next stack operation: In CF console, for Stack failure options, select “Preserve successfully resources” | | |
| YAML or JSON Template -> Upload to S3 -> CF makes API calls -> Set of resources CF builds = Stack | | |
| Template: AWSTemplateFormatVersion, Description, Metadata, Parameters (input values), Conditions,  Mappings, Transform (snippets of code outside main template), Resources (mandatory section), Outputs  - Use !Ref <parameter name> | | |
| E.g. Can import (parameter) SecurityGroupID and SubnetID (Fn::ImportValue) that were output (Outputs) from the NetworkStack into new stack by referencing them in CF template  Or can self-populate predefined parameters by AWS CloudFormation  - E.g. AWS::Region, AWS::StackId, AWS::AccountId | | |
| Serverless Application Model (SAM) | | | Extension to CF used to define serverless apps  Simplified syntax for defining serverless resources: APIs, Lambda, DynamoDB  Use SAM CLI to package deployment code, upload to S3, deploy app  *sam package --template-file <template.yml> --output-template-file <output.yml> --s3-bucket <bucket>*  *sam deploy --template-file < output.yml> --stack-name <stack name> --capabilities CAPABILITY\_IAM*  *sam sync --watch # Only redeploy changes made locally to cloud* | |
| CF Nested Stacks | Enable re-use of CF code for common use cases, e.g. standard config for load balance, web server  - Instead of copying out code each time, create a standard template for each common use case and reference from within your CF template  Resources:  Type: AWS::CloudFormation::Stack  TemplateURL: https://s3.amazon.aws.com/.../template.yml | | | |
| Cloud Development Kit (CDK) | | | | Open-source framework to build apps, define and deploy AWS resources using a programming language of your choice (TypeScript, Python, Java, .NET, Go)  - Use CDK to transform code into CF template, then CDK can provision resources using CF |
| App: container for 1 or more stacks. Stack: Unit of deployment. Construct: Defines the AWS resources  CDK Toolkit aka CDK CLI |
| cdk init -> Compile app -> cdk synth (create CF template) -> cdk deploy |
| Amplify | Allow frontend developers to quickly and easily create full-stack web and mobile apps using AWS  - Amplify takes care of creating a stable and reliable backend to support apps (authentication, storage, Lambda functions, …)  - Client libraries and CLI. Console that provides continuous deployment and hosting of apps  - Frontend devs can focus on developing web frontend of app  - Amplify libraries integrate app w services like Cognito, S3, Lambda, API Gateway  - To add end-to-end testing before deploy to prod: add test phase to amplify.yml build settings for app | | | |
| 1) Amplify Hosting: fully managed web hosting service  - Web app and static website hosting. – CI/CD functionality. - Integrates w code repo | | | |
| 2) Amplify Studio: simple visual interface, separate from AWS Management Console  - Create frontend UI. – Configure and maintain app backend. – Add features like authentication, integration w data services, serverless functions | | | |
| Extra | Dev is testing AWS Lambda function by using AWS SAM local CLI. App that is implemented by the Lambda function makes several AWS API calls by using AWS SDK. Dev wants to allow the function to make AWS API calls in a test AWS acct from the dev laptop  - Add a test profile by using the aws configure command with the --profile option. Run AWS SAM by using the *sam local invoke* command with the --profile option.  Dev want to use a feature of AWS SAM to set up deployments to multiple envs:  - Add config file in TOML format to group config entries to every env. Add a table for each testing and staging env. Deploy updates to envs by using the sam deploy command and the --config-env flag that corresponds to each env  Reference new S3 bucket from another AWS CloudFormation template:  - Add Export declaration to Outputs section of original template and use ImportValue in other templates  CodeDeploy deployment fails w error message, *InvalidSignatureException – Signature expired: [time] is now earlier than [time]*  - Date/time setting on EC2 instance is not correct. CodeDeploy require accurate time references to perform its operations. If not set correctly, might not match signature timestamp of CodeDeploy deployment request  CodeBuild, downloads of large dependency files consume most of the time on each build. Downloaded dependency files stored in a /.m2 folder. Builds run frequently. How to improve build performance?  - Modify CodeBuild project to implement local caching. Modify the buildspec’s cache settings to include a path to the /.m2 folder | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Advanced IAM | | | |
| Web Identity Federation | | | Simplifies authentication and authorization for web apps  1) Users access AWS resources after successfully authenticating w a web-based identity provider like Facebook, Amazon, Google  2) Users receive an authentication code from web ID provider (JWT token)  3) Users can trade this authentication code for temp AWS security credentials from Cognito, authorizing access to AWS resources  - Cognito works w ALB, not CloudFront |
| AWS Cognito: provides web ID federation, including sign-up and sign-in functionality for app and access for guest users  - Acts as identity broker: manages authentication btw app and web ID provides, so don’t need extra code  - Synchronizes user data for multiple devices  - Recommended for all mobile apps that call AWS services  - Temp credentials map to an IAM role, to allow access  1) User Pools: User directories used to manage sign-up and sign-in functionality for mobile and web app  - Users can sign in directly to User Pool, OR using Facebook, Amazon, Google  2) Identity Pools: enable you to provide temp AWS credentials |
| Cognito Push Synchronization (across devices)  - Cognito tracks association btw user identity and various diff devices they sign-in from  - Cognito uses Push Synchronization to push updates and synchronize user data across multiple devices  - Underlying uses SNS silent notification to all devices associated w a given user identity whenever data stored in cloud changes |
| Inline vs Managed vs Customer Managed Policies | | 1) AWS Managed Policies: IAM policy created and administered by AWS  - E.g. AmazonDynamoDBFullAccess, AWSCodeCommitPowerUser, AmazonEC2ReadOnlyAccess  - Assign appropriate permissions to users w/o having to write policy yourself  - Attach to multiple users, groups, roles in same AWS acct or diff accts  - Cannot change permissions defined in an AWS managed policy  2) Customer Managed Policies: standalone policy you create and administer inside OWN acct  - Can attach this policy to multiple users, groups, roles within own acct  - To create a customer managed policy, can copy an existing AWS managed policy and customize it to fit requirements of your org  3) Inline Policies: strict 1:1 r/s btw entity and policy. – Not recommended to use  - When you delete user, group, or role in which inline policy is embedded, policy will also be deleted | |
| Security Token Service (STS) | STS AssumeRoleWithWebIdentity: API that returns temp security credentials for users authenticated by a mobile or web app or Federated identity provider  - Cognito calls the STS assume-role-with-web-identity API underlying  - Sample response contain AssumedRoleUser (the ARN and AssumedRoleID to programmatically reference the temp credentials, not an IAM role or user), credentials (access + secret key) | | |
| Cross Acct access | E.g. Prod acct need give access to S3 to dev acct group  1) Create user in dev acct group  2) In prod acct, create IAM policy that has access  3) In prod acct, create IAM role w trust policy to another AWS acct, and put in acct ID of dev acct  4) Attach created IAM policy to IAM role in prod acct  5) In dev acct, create IAM inline policy to assume role created in Prod acct (Action: sts:AssumeRole, Resource: arn:aws:iam::<prod acct ID>:role/<prod role>)  6) Sign in to dev acct user. Click top right, and click switch role (provide prod acct ID and prod role) | | |
| Extra | Cognito can email users for certain events in user pool’s client app  - When user log in, sign up, reset password, can auto send email  Cognito identity pool attribute-based access control to limit user access to S3 objects by department. Company uses an Cognito user pool to authenticate all users. Company adds a “department” attribute and populates attribute for each user. Company tags each S3 object w a “department” tag.  - Define 1 IAM role on the identity pool. Associate the role w a policy that allows S3 permissions to any resources w the “s3:ExistingObjectTag/department”: “${aws:PrincipalTag/department}” condition  New mobile app uses Cognito web identity federation. After a user logs in, error *AccessDenied -- Not authorized to perform sts:AssumeRoleWithWebIdentity*  - Developer incorrectly defined the authenticated principal role trust policy  Cognito user pools need add social identity providers (IdPs) to user pool. Need provide what info?  - App client ID and app client secret. Social IdPs are based on OpenID. All OpenID providers require client apps to register. Result of registration are an app client ID and app client secret, which must be provided  Mobile app obtains an OpenID token by using web identity provider’s login process. Code needs AWS credentials to make call to DynamoDB  - Use Cognito identity pool. Call the GetId API, then call the GetCredentialsForIdentity API  - GetId API is called w an OpenId token and returns a Cognito ID  - GetCredentialsForIdentity API called w an Cognito ID and returns temp credentials | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Monitoring | | | | | |
| Cloud-Watch | Monitoring service to monitor health and performance of AWS resources, as well as apps in AWS or on-prem  1) Compute: EC2, Auto Scaling groups, ELB, Route53 health checks, Lambda  2) Storage & Content Delivery: EBS volumes, Storage Gateway, CloudFront  3) DB & Analytics: RDS, DynamoDB tables, Elasticache nodes, Redshift, EMR  4) Other: SNS topics, SQS queues, API Gateway, Estimated AWS charges  - Can use CloudWatch Agent to define custom metrics  - CloudWatch Logs allows you to monitor OS and app logs. - CloudWatch metrics stored indefinitely.  – Can retrieve data from any EC2 or ELB even after it has been terminated | | | | |
| By default, all EC2 send key health & performance metrics to CloudWatch: CPU, network, disk, status check  EC2 don’t send OS level metrics to CloudWatch. To do so, need install CloudWatch agent to collect and send  - OS Metrics: memory usage, processes running on instance, amt of free disk space, CPU idle time, …  By default, EC2 send metrics to CloudWatch at 5-min intervals  - Enable detailed monitoring to send every 1 min (default, can send every 1s), but need pay extra | | | | |
| Can monitor system and app logs: - Monitor and troubleshoot apps using existing system and app logs files  - Monitor logs in near-real time for specific phrases, values, patterns. Requires CloudWatch agent | | | | |
| CloudWatch Alarms: create alarm to monitor CloudWatch metric  - Set thresholds to trigger alarms and actions to be taken if alarm state reached  - Can set to send to SNS topic and email | | | | |
| CloudWatch Dashboards: create custom view of metrics that is relevant to u  - Can display metrics for any Region | | | | |
| Cloud-Watch Metrics | | Metric = variable to monitor. - Metrics uniquely defined by a name, namespace, and ≥ 0 dimensions  - Time-ordered seq of values or data points  - Each data point in a metric has a timestamp, and optionally a unit of measurement | | | |
| Namespace = container for CloudWatch metrics (e.g. EC2 use AWS/EC2 namespace)  - Create own namespace to publish custom metric data  - Must specify namespace for each data point or value you publish to CloudWatch  - Specify namespace name when you create a metric  - Metrics in diff namespaces are isolated from each other  - Custom metrics appear under Custom Namespaces, while default AWS metrics under AWS Namespaces  - Can send custom metrics from diff apps to diff namespaces, so can segregate based on app | | | |
| CloudWatch Dimensions: Dimension is like a filter  - Name/value pair: can be used to filter CloudWatch data  - CloudWatch can aggregate data across dimensions for some services (e.g. EC2) | | | |
| CloudTrail | | | Records user activity in your AWS acct  - Records events related to creation, modification, or deletion of resources  - By default, can view last 90 days of acct activity  - Can deliver log files to an S3 bucket. – Can be integrated w CloudWatch Logs | | |
| CloudWatch Actions | | | | | CloudWatch API supports diff actions which allows to publish, monitor and alert on variety of metrics  PutMetricData: publish metric data points to CloudWatch  aws cloudwatch put-metric-data --metric-name <metric name> --namespace <Namespace> \  --value <value> --timestamp 2022-01-01T00:00:00.000Z  PutMetricAlarm: creates an alarm associated w a metric to alert if threshold has been reached  aws cloudwatch put-metric-alarm --alarm-name <alarm name> --alarm-description <description> \  --metric-name <metric name> --namespace <Namespace> --statistic Average --period 300  --threshold 50 --comparison-operator GreaterThanThreshold --evaluation-periods 1 |
| CloudWatch Logs Insights | | | | | Interactive query and analysis of data stored in CloudWatch Logs  - Bespoke query language. – Generate visualizations  - E,g, create lambda function, and new log group is auto created |
| Common HTTP Error Codes | | Client-Side Errors: 4XX  400 – Access Denied Exception: you don’t have required access  403 – Missing Authentication Token: request didn’t contain a valid X.509 cert or AWS access key ID 404 – Malformed Query string: query string contains a syntax error. Obj don’t exist or file not found | | | |
| Server-Side errors: 5XX. – Request was valid but server not able to fulfil it  500 – Internal Failure: request failed due to unknown error, exception, or failure  503 – Service Unavailable: Request failed due to a temp failure of server (can be caused by high traffic to website and server is not responding, or internal failure) | | | |
| Common SDK Exceptions | | | | SDK Exception: response to an error that has occurred when processing an SDK or API request  1) BatchGetItems: returns details of ≥ 1 items from DynamoDB table  - Limited to up to 16MB of data and up to 100 items  - If DynamoDB cannot return all items, it returns a partial result, along w an exception  - Causes of partial result: requested > 100 items (ValidationException) OR  request > 16 MB of data (ValidationException) OR  exceed provisioned throughput of table (ProvisionedThroughputExceededException) OR  UnprocessedKeys (some items not successfully processed during BatchGetItem operatioin) | |
| 2) BatchWriteItem: puts or deletes ≥ 1 item in ≥ 1 DynamoDB tables  - Limited to up to 16 MB of data and up to 25 put or delete operations  - If any put or deletes fail, DynamoDB returns list of UnprocessedItems  - Or ProvisionedThroughputExceededException | |
| EventBridge | | | | EventBridge all about event-driven architecture. Event = change in state | |
| Scheduled events: EventBridge rules that run on a schedule Or Rule w an event pattern | |
| CloudWatch Events and EventBridge are same underlying service and API but EventBridge provides more features | |