AWS Certified Developer Associate 2020 (2)

DynamoDB Follow Along

- Getting Started
  - Create a Cloud 9 environment
  - Console > DynamoDB > Create Table
    - table name
    - partition key (hash) and sort key
    - as long as the combined partition and sort key results in a unique string it’s OK
    - choose data type for key as string, binary, or number (SBN)
    - configure capacity mode
    - if need an LSI, must be created now
- Create Table (1)
  - Create table -

```
## Create DynamoDB Table

aws dynamodb create-table \
  --table-name Starships \
  --attribute-definitions \
    AttributeName=ShipClass,AttributeType=S \
    AttributeName=Registry,AttributeType=S \
  --key-schema \
    AttributeName=ShipClass,KeyType=HASH \
    AttributeName=Registry,KeyType=RANGE \
  --provisioned-throughput \
    ReadCapacityUnits=5,WriteCapacityUnits=5 \
  --region us-east-1
```

  - Check status of table creation

```
## Describe Table

aws dynamodb describe-table --table-name Starships
```

- Batch Write Item
  - getting data into the new table
  - data format must be in JSON
  - the **batch-write-item** operation puts or deletes multiple items in 1/+ tables, a single call can write up to 16 MB of data which can comprise of up to 25 put or delete requests and individual items can be up to 40 KB

```ruby
require 'csv'
require 'json'

batches = []
count = 25
CSV.read("starfleet.csv","r:ISO-8859-1", headers: true).each do |line|
  if count == 25
    batches.push({"Starships" => []})
    count = 1
  end

  batches.last["Starships"].push({
    "PutRequest" => {
      "Item" => {
        "Registry"    => { "S" => line['registry']},
        "Name"        => { "S" => line['name']},
        "ShipClass"   => { "S" => line['ship_class']},
        "Description" => { "S" => line['description']}
      }
    }
  })
  count = count + 1
end

batches.each_with_index do |batch,i|
  json = JSON.pretty_generate batch
  File.write "batches/batch-#{i.to_s.rjust(3,'0')}.json", json
end
```

- this script transforms table data into JSON format, splits into multiple files containing 25 records each (max 16 MB)
- **aws dynamodb batch-write-item —request-items** file://items.json

- Get Item
  - returns items in JSON format

```
aws dynamodb get-item \
    --table-name MusicCollection \
    --key file://key.json
```

  - the key is the schema key to retrieve the records, basically looking for items

```
{
    "Registry": {"S": "NCC-13958"},
    "ShipClass": {"S": "Excelsior"}
}
```

```
ec2-user:~/environment/dynamodb-playground $ aws dynamodb get-item --table-name Starships --key file://key.json
{
    "Item": {
        "Description": {
            "S": "Ship commanded by Admiral James Leyton and on which Benjamin Sisko served as first officer."
        },
        "ShipClass": {
            "S": "Excelsior"
        },
        "Registry": {
            "S": "NCC-13958"
        },
        "Name": {
            "S": "USS Okinawa"
        }
    }
}
```

  - Projection Expression can be used to filter for specific attributes

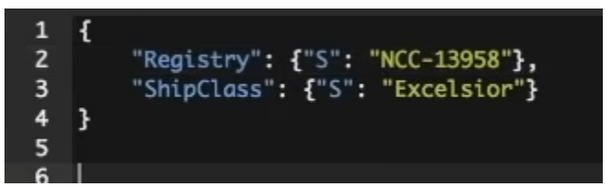
- Batch Get Item
  - getting multiple items in a single DynamoDB command
  - **aws dynamodb batch-get-item —request-items** file://items.json
  - use ProjectionExpression to filter out the fields from the output, but this is optional if you want to return all fields

```
aws dynamodb batch-get-item \
    --request-items file://request-items.json
```

Contents of request-items.json:

```
{
    "MusicCollection": {
        "Keys": [
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Call Me Today"}
            },
            {
                "Artist": {"S": "Acme Band"},
                "SongTitle": {"S": "Happy Day"}
            },
            {
                "Artist": {"S": "No One You Know"},
                "SongTitle": {"S": "Scared of My Shadow"}
            }
        ],
        "ProjectionExpression":"AlbumTitle"
    }
}
```

- Delete Item
  - **aws dynamodb delete-item —table-name** starships **—key** file://key.json



- Put Item
  - **aws dynamodb put-item —table-name** starships —**item** file://item.json — **return-consumed-capacity** TOTAL

- Update Item
  - **aws dynamodb update-item —table-name** starships —**key** file://items.json \

    —**update-expression **

    —**expression-attribute-names**
  - Create 2 files, one for expression-attribute-names.json, the other for expression-attribute-values.json

```
aws dynamodb update-item \
    --table-name MusicCollection \
    --key file://key.json \
    --update-expression "SET #Y = :y, #AT = :t" \
    --expression-attribute-names file://expression-attribute-names.json \
    --expression-attribute-values file://expression-attribute-values.json  \
    --return-values ALL_NEW
```

Contents of key.json:

```
{
    "Artist": {"S": "Acme Band"},
    "SongTitle": {"S": "Happy Day"}
}
```

Contents of expression-attribute-names.json:

```
{
    "#Y":"Year", "#AT":"AlbumTitle"
}
```

Contents of expression-attribute-values.json:

```
{
    ":y":{"N": "2015"},
    ":t":{"S": "Louder Than Ever"}
}
```

- Scan
  - if you create a table without a sort key, you can only scan
  - a scan returns all records in a table, and then you can filter those results to find what you want
  - generally use query when you can
  - even though you filter out the results, it's already returned all records so you've already used up the read capacity units
  - project expression is optional to only return relevant attributes, otherwise returns all attributes, you can use operations on the sort key

```
# Scan

aws dynamodb scan \
    --table-name Starships \
    --filter-expression "begins_with(Description, :D)" \
    --expression-attribute-values "{":D": {"S": "Destroyed"}}"
```

- Query

```
aws dynamodb query --table-name Starships --projection-expression "Registry" --key-condition-expression "ShipClass = :C" --expression-attribute-values '{":C": {"S": "Galaxy"}}' --output text
```

  - must specify partition key, sort key optional
  - the difference is it only returns records that match the query filters, it doesn't return all records then hides them from display

- can filter further based on attributes
- Projection expression must match field case

```
aws dynamodb query \
    --table-name MusicCollection \
    --projection-expression "SongTitle" \
    --key-condition-expression "Artist = :v1" \
    --expression-attribute-values file://expression-attributes.json
```

- Transact
  - **transact-get-items**
  - **transact-write-items**
  - synchronous operations that retrieves items from 1/+ tables, if any errors happen the entire transaction is rolled back
  - all or none

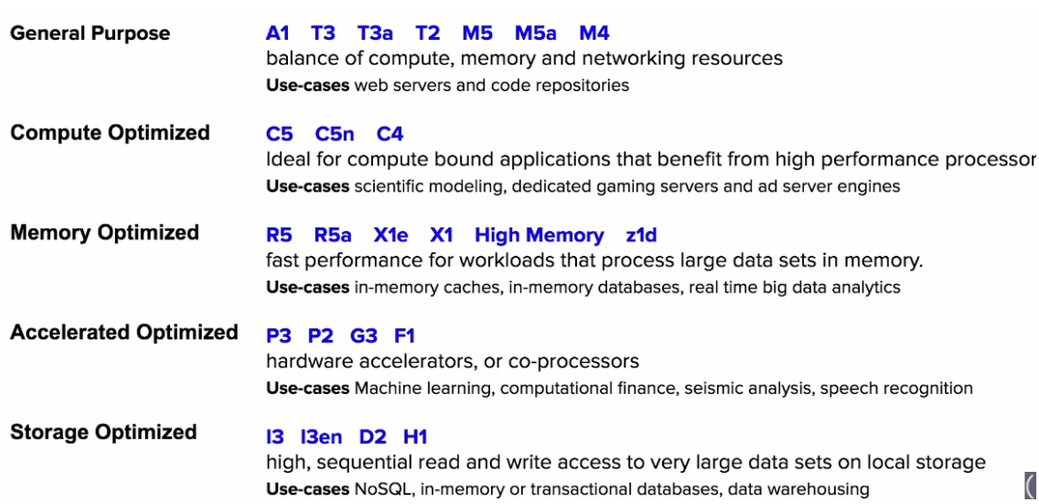
```
aws dynamodb transact-get-items \
    --transact-items file://transact-items.json
```

  - DynamoDB rejects the entire TransactWriteItems request if any of the following is true:
    - A condition in one of the condition expressions is not met.
    - An ongoing operation is in the process of updating the same item.
    - There is insufficient provisioned capacity for the transaction to be completed.
    - An item size becomes too large (bigger than 400 KB), a local secondary index (LSI) becomes too large, or a similar validation error occurs because of changes made by the transaction.
    - The aggregate size of the items in the transaction exceeds 4 MB.
    - There is a user error, such as an invalid data format.
- Delete Table
  - **aws dynamodb delete-table —table-name** 12345

Ec2

- Introduction
  - cloud computing service, choose your OS, storage, memory, network - launch and SSH to your server in mins
  - highly configurable server
  - resizable compute capacity, takes minutes to launch new instances

    - anything and everything on AWS uses an Ec2 instance underneath
    - Configuration Options
        - Choose your OS via AMIs
        - Choose your Instance type and size
        - Add Storage (EBS, EFS)
        - Configure your instance - security groups, key pairs, UserData, IAM Roles, Placement Groups
- Instance Types
    - **General Purpose -** for web servers and office servers
    - **Compute Optimized** - for compute-intensive apps like dedicated gaming servers, modelling, and ad servers
    - **Memory Optimized -** big data platform, in-mem caches and in-mem DBs
    - **Accelerated Optimized -** for ML, computational finance, seismic analysis
    - **Storage Optimized -** for high sequential RW to local storage - Data warehouses, SQL databases



General Purpose	**A1 T3 T3a T2 M5 M5a M4** balance of compute, memory and networking resources **Use-cases** web servers and code repositories
Compute Optimized	**C5 C5n C4** Ideal for compute bound applications that benefit from high performance processor **Use-cases** scientific modeling, dedicated gaming servers and ad server engines
Memory Optimized	**R5 R5a X1e X1 High Memory z1d** fast performance for workloads that process large data sets in memory. **Use-cases** in-memory caches, in-memory databases, real time big data analytics
Accelerated Optimized	**P3 P2 G3 F1** hardware accelerators, or co-processors **Use-cases** Machine learning, computational finance, seismic analysis, speech recognition
Storage Optimized	**I3 I3en D2 H1** high, sequential read and write access to very large data sets on local storage **Use-cases** NoSQL, in-memory or transactional databases, data warehousing

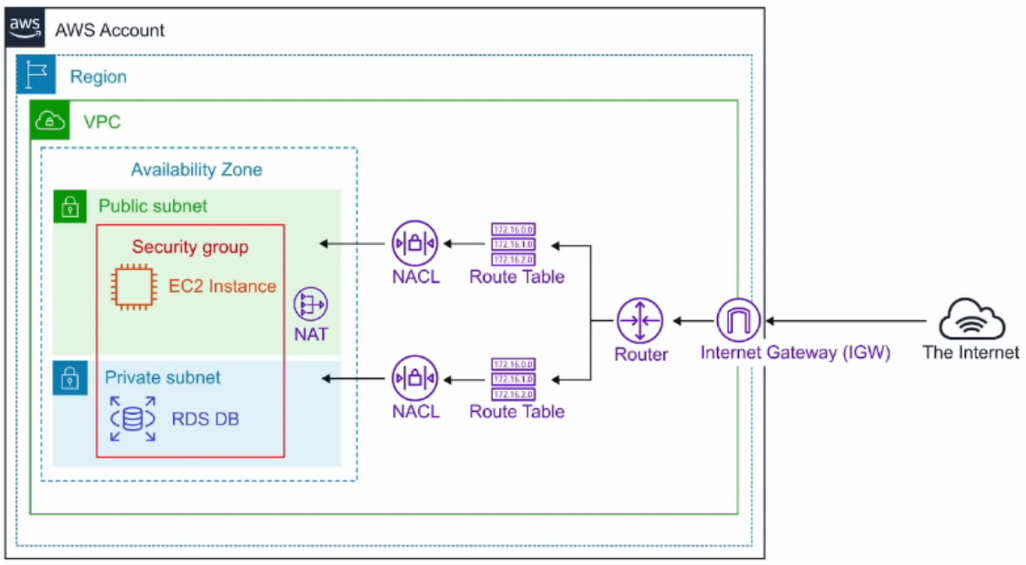
- Instance Sizes
    - Ec2 instance sizes generally double in price and key attributes (but specific types can vary)
    - generally if you start to need 2 instances it’s better to just upgrade the instance type/size to the next larger one
- Instance Profiles
    - instread of embedding AWS creds (access key + SAK) in code running on the instance, you attach a role to the instance via an instance profile
    - **always avoid embedding AWS creds whenever possible**

  - an instance profile holds a reference to a role, the Ec2 instance is associated with the profile.
  - Console - when attaching an IAM role to an Ec2 instance the instance profile is automatically created
  - CLI/CF template - must manually create an instance profile
- Placement Groups
  - lets you choose the logical placement of instances to optimize for comm., perf, or durability
  - free to use and configure
  - 3 types - cluster/partition/spread
  - Cluster PG (all in a group)
    - instances spun up closely together inside one AZ
    - low-latency network perf for tightly-coupled node-to-node comms
    - good for HPC apps
    - clusters can’t be multi-AZ
  - Partition PG (divide into large subgroups)
    - AZ carved out into partitions
    - instances spread across multiple partiitons
    - each partition does not share underlying h/w with each other (rack per partition)
    - for large distributed and replicated workloads (hadoop, cassandra, kafka)
  - Spread PG (spread each instance)
    - each instance on a diff. rack
    - critical instances kept separate from each other
    - up to 7 instances per spread PG and each spread PG can cover multi-AZ
- Userdata
  - a script that auto-runs when an instance is first launched
  - can be used to install packages, apply updates, or run any other script
  - can check running instance metadata to see if any scripts are assigned/run - http://169.254.169.254/latest/user-data
- Metadata
  - addt. info about Ec2 instance that you can get at runtime

    - run *locally* on the Ec2 instance - http://169.254.169.254/latest/meta-data
        - /public-ipv4
        - /ami-id
        - /instance-type
    - combine metadata + userdata scripts to perform advanced instance staging automation
- Cheat Sheet
    - Ec2 is a cloud computing service
    - configure Ec2 by choosing OS, storage, memory, network throughput
    - launch and SSH into server within minutes
    - 5 types of instances
        - GP - balance of compute, memory, networking (T, M)
        - Compute Optimized (C) - for CPU-intensive apps
        - Memory Optimized - for large data processing workloads (r, X, High Memory, z)
        - Accelerated Optimized - H/W accelerators and co-processors (P, DL, Trn, Inf, G, F, VT)
        - Storage Optimized - for sequential RW access to large datasets (Im4, Is4, I, D, H)
    - instance sizes generally double in price and key attributes
    - placement groups let you choose logical placement of instances to optimize for comm, perf, or durability. Free to use.
    - Userdata - script to autorun when Ec2 instance launches
    - Metadata - data about current instance - instance type, IO, size, IP add, etc. (accessed locally on instance)
    - Instance Profiles - container for an IAM role used to pass role info to an Ec2 instance when it launches

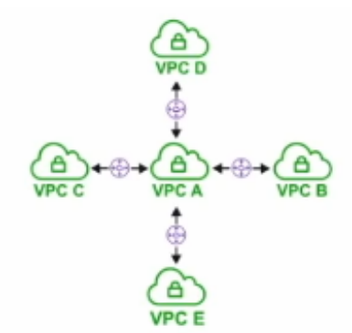
VPC

- Introduction
    - provision a logically isolated section of the AWS cloud where you can launch AWS resources in a virtual network you define
- Core Components
    - a VPC on AWS is your own personal datacenter
    - gives you complete control over your virtual networking environment



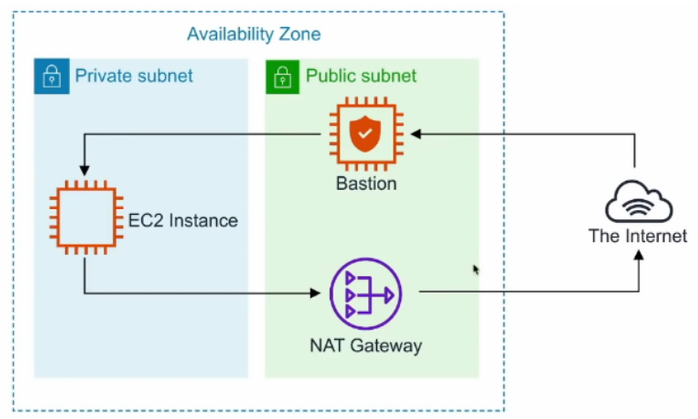
    - Core components of a VPC
        - internet gateway
        - Virtual private gateway
        - routing tables
        - NACLs
        - SGs
        - Public and Private subnets
        - NAT Gateway
        - Customer Gateway
        - VPC Endpoints
        - VPC Peering
- Key Features
    - VPCs are region specific and can’t span multiple regions
    - up to 5 VPCs per region
    - every region comes with a default VPC
    - up to 200 subnets per VPC
    - can assign both IPV4 and IPv6 CIDR blocks
    - Free to use - VPCs, route tables, NACL, IGW, SG, Subnets, Peering
    - Addt. components that cost money - NAT GW, VPC Endpoints, VPG, CG, TGW
    - DNS Hostnames - disabled by default, but can be turned on to automatically provide a unique public DNS name for the instance if enabled
- “Create Default VPC”
    - comes in every region for an AWS account so you can immediately deploy instances
    - creates a VPC with a /16 CIDR block (172.31.0.0/16)

    - creates a /20 subnet default subnet in each AZ
    - creates an IGW and connects it to the default VPC
    - creates a default sec group and associates it to the VPC
    - creates a default NACL and associates it to the VPC
    - associates the default DHCP options set for the AWS account
    - comes with 1 main route table used by every subnet
- Default Everywhere IP
    - 0.0.0.0/0 is the default route → internet access if pointed to IGW/NAT GW/Virtual appliance in routing table associated to a subnet
- VPC Peering
    - allows to connect a VPC to another over the AWS backbone using private IP addresses
    - allows 2 VPCs to behave as if they were on the same network infrastructure
    - VPC peers can be in diff. AWS accounts and regions
    - uses a star configuration - 1 hub VPC + spoke VPCs
    - **No transitive peering -** peering must take place directly btw. VPCs, needs a 1-to-1 connect to immediate VPC, you can’t flow traffic through an intermediate VPC to a 3rd VPC
    - **No overlapping CIDR blocks**



- Route Tables
    - determines where network traffic is directed
    - each subnet in the VPC must be associated with a route table
    - a subnet can only be associated to one route table but a route table can have many subnets associated
    - all route tables come with a local route
- IGW
    - allows VPC access to internet
    - does 2 things
        - provides a target in VPC route tables for internet-routable traffic

        - performs NAT for instances that have been assigned public IPv4 addresses
    - to route out to the internet, subnet route tables must have a default route pointing to the IGW
- Bastions/Jumpbox



    - an ec2 instance placed in a public subnet
    - should be hardened and locked down using SGs
    - systems manager sessions manager replaces the need for manual bastion instances and becoming more recommended
- Direct Connect
    - used to establish dedicated network connection from on-prem to AWS
    - very fast L2 network
        - lower bandwidth from 50M - 500M
        - higher bandwidth from 1GB - 10GB
    - helps reduce network cost while increasing bandwidth throughput
    - provides a more consistent network experience than a typical internet-based connection (reliable and secure)

Auto Scaling Groups

- Introduction
    - set scaling rules which will auto-launch or auto-shutdown instances to meet real time demand levels
    - AS Groups contain a collection of Ec2 instances that are treated as a group for the purposes of automatic scaling and mgmt
    - Scaling can occur via
        - capacity settings
        - health check replacements
        - scaling policies
- Capacity Settings

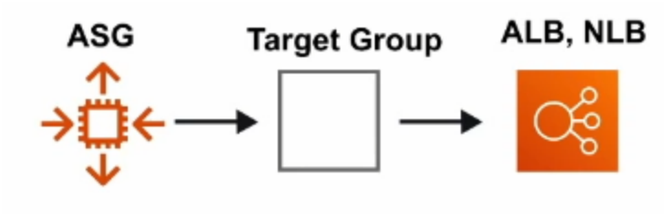
- size of an AS group based on min, max, and desired capacity
  - Min - how many instances should always be running at the minimum
  - Max - number of instances allowed to be running so prevent infinite number of instances running
  - Desired - how many instances to ideally be running
- ASG will always launch instances to meet min. capacity, and will auto-replace shutdown or failed instances

- Health Check Replacements
  - can be used to trigger autoscaling actions
  - 2 types of health checks - EC2 + ELB
  - EC2 Health Check
    - ASG performs HC on instances to see if it's healthy
    - 2 checks are performed
    - based on Ec2 status checks
    - unhealthy instances will cause ASG to terminate and relaunch the instance
  - ELB Health Check
    - ASG performs health check based on ELB health check
    - ELB performs HC by pinging an HTTP(S) endpoint with an expected response (i.e.: HTTP/200)
    - IF ELB determines instance is unhealthy it tells the ASG to terminate and replace the unhealthy instance

- Scaling Policies
  - 3 types of scaling policies
  - Target tracking - maintains a specific metric at a specific value i.e.: if avg. CPU utilization exceeds 75% then add an instance
  - Simple - scales when a CW alarm is breached, scale-out or scale-in, no longer recommended as legacy policy, use step scaling instead
  - Step - scale when an alarm is breached but also escalate based on alarm value changing

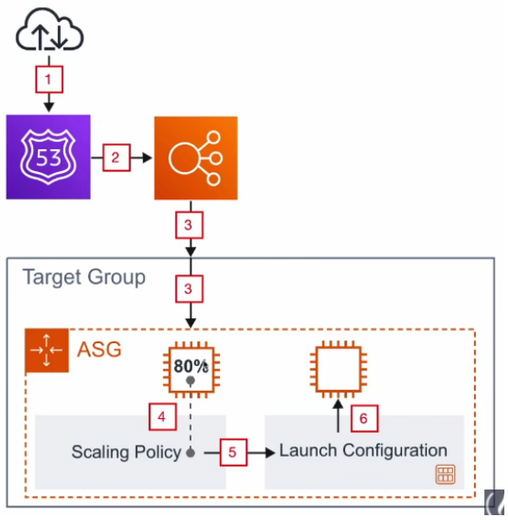
- ELB Integration

- ASGs can be associated with ELBs, when an ASG is associated with ELB, richer health checks can be set
- CLBs are associated directly to the ASG (legacy)
- ALBs + NLBs are configured with target groups which can contain the instances or ASGs



- Use Case

1. Burst of traffic from the internet hits our domain.
2. Route53 points that traffic to our load balancer.
3. Our load balancer passes the traffic to its target group.
4. The target group is associated with our ASG and sends the traffic to instances registered with our ASG
5. The ASG Scaling Policy will check if our instances are near capacity.
6. The Scaling Policy determines we need another instance, and it Launches an new EC2 instance with the associated Launch Configuration to our ASG



- Launch Configuration
  - an instance config template that an AS group uses to launch instances
  - a launch config is set up when a new AS group is created
  - same process to create as launching an instance except you save the config instead of launching a new instance right away
  - CANNOT be edited afterwards, if need update = create a new one or clone existing config then manually associate to te new launch config
  - Launch templates are an improved offering over launch config that supports versioning
- Cheat Sheet
  - an ASG is a collection of Ec2 instances grouped for scaling and mgmt
  - Scaling out - when you add servers
  - Scaling in - when you remove servers
  - Scaling up - when you increase size of an instance (i.e.: updating launch config with larger size)
  - size of an ASG is based on min, max, and desired capacity

    - Target scaling policy scales based on when a target value for a metric is breached (Avg CPU util)
    - Simple scaling policy triggers scaling when an alarm is triggered
    - Simple scaling policy with steps is a new version of the simple scaling policy and allows to create steps based on ec2 alarm values
    - Desired capacity = how many instances to ideally run
    - An ASG will always launch instances to meet min. capacity
    - Health Checks
        - determine current state of an instance in an ASG
        - can be run against ELB or Ec2 instances
    - When AS launches a new instance it uses a Launch config which holds the config values for that new instance i.e.: AMI, InstanceType, Role
    - Launch configs
        - can’t be changed after creation and must be cloned/new one created
        - must be manually updated by editing AS settings

VPC Endpoints

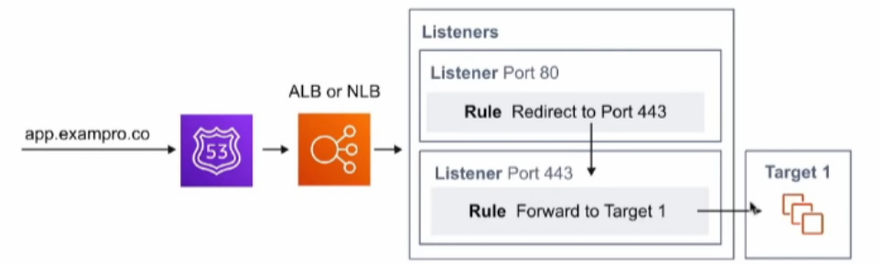
- Introduction
    - used to privately connect VPC to other AWS svcs, and VPC endpoint svcs
    - eliminates need for an IGW, NAT device, or VPN, or DirectConnect to AWS network
    - instances in VPC no longer need a public IP to talk to svcs
    - traffic btw. VPC and svcs does not leave the AWS network
    - horizontally scaled, redundant, highly available VPC component
    - allows secure comms btw instances and svcs w/o adding availability risks or bw constraints on traffic
    - 2 types - inf + gw endpoints
- Interface Endpoints
    - ENI with a private IP address that serve as an entry point for traffic going to a supported service
    - powered by AWS PrivateLink - access svcs hosted on AWS easily and securely by keeping network traffic within AWS network
    - $0.01 per hour per endpoint per AZ + $0.01 per GB data processed = $7.5/mo
    - Supports a lot of AWS svcs like API GW, Cloudformation, Cloudwatch, Kinesis, SageMaker, Codebuild, AWS Config, Ec2 API, ELB API, KMS, Secrets Mgr, STS, SNS, SQS, ...
    - can point to endpoint svcs in other AWS accounts

- Gateway Endpoints
  - a gateway that’s a target for a specific route in the route table used for traffic destined for a supported AWS svc
  - only used for S3 and DynamoDB
  - free as you only need a route in the VPC route table
  - To create - specify the VPC to create the endpoint + the service to point the endpoint towards
- VPC Endpoints cheat sheet
  - VPC endpoints help keep traffic btw AWS svcs within the AWS network
  - 2 types - inf + gw endpoints
  - inf endpoints cost money vs. gw endpoints are free
  - Inf endpoints use an ENI with a private IP (powered by AWS PrivateLink)
  - Gw endpoints is a target for a specific route in the route table
  - Inf endpoints support many AWS svcs
  - Gw endpoints only support DynamoDB + S3

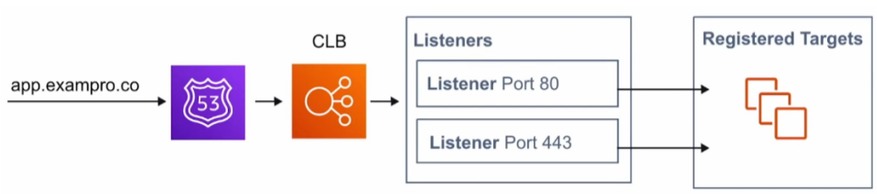
ELB

- Introduction
  - distributes incoming app traffic across multiple targets - EC2, containers, IP addresses, and Lambda functions
  - Load balancers
    - h/w or s/w that accepts incoming traffic then distributes it to multiple targets
    - load is balanced using rules
    - ELB is the AWS solution to load balancing - 3 types - ALB, NLB, CLB
- Rules of Traffic
  - listeners - incoming traffic evaluated against listeners to match the inbound port, for CLB the Ec2 instances are registered directly to the load balancer
  - rules - not available for CLB, rules are invoked by listeners to decide what to do with traffic, usually the next step is to fwd traffic to a target group
  - target group - not available for CLB, instances are registered as targets in a target group
  - ALB/NLB rules of traffic
    - traffic is sent to listeners
    - if port matches then rules are evaluated
    - rules tell the NLB/ALB to send traffic to a target group

        - target group will evenly distribute traffic to all instances registered to that group

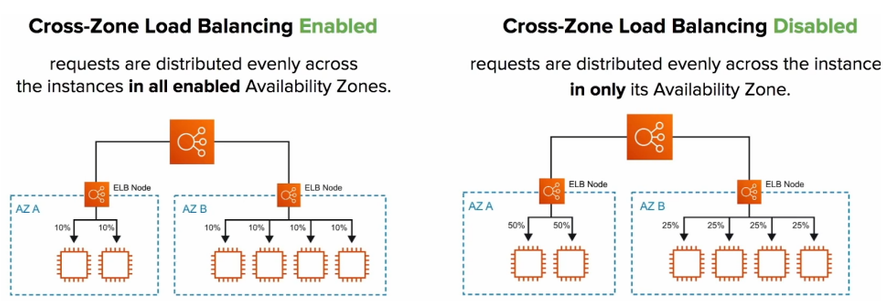


    - CLB rules of traffic
        - traffic sent to listeners, if port matches CLB fwd traffic to any Ec2 instance that are registered to it
        - doesn't allow to apply rules to listeners



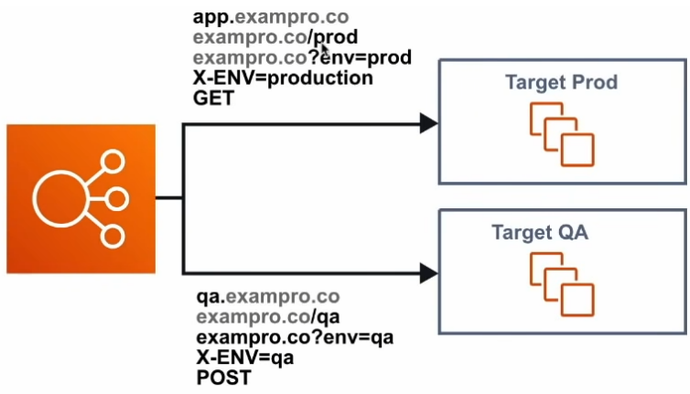
- Application Load Balancer ALB
    - for balance HTTP/HTTPS traffic
    - runs at OSI Layer 7
    - supports request routing feature - allows to add routing rules to listeners based on HTTP protocol
    - can attach a WAF to ALB
    - good for web apps
- Network Load Balancer NLB
    - designed to balance TCP/UDP traffic
    - rules at OSI layer 4
    - very high throughput at very low latency - millions of requests per sec
    - can do cross-zone load-balancing
    - use for gaming and/or when network performance is critical
- Classic Load Balancer CLB
    - AWS' first load balancer, legacy
    - can balance HTTP, HTTPS, or TCP traffic but not at the same time
    - can use layer 7 specific features such as sticky sessions
    - but can also use strict layer 4 balancing for pure TCP apps
    - can do cross-zone LB

    - responds with a 504 timeout if underlying app is not responding
    - **not recommended for use, use NLB or ALB**
- Sticky Sessions
    - advanced LB method that allows to bind a user's session to an instance
    - ensures all requests from that session are sent to the same instance
    - used with a CLB, can be enabled for ALB but can only be set on target group not on individual instances
    - uses cookies to remember instance
    - used when specific info is only stored on a single instance
- X-Forwarded-For (XFF) Header
    - used when you need to know the IPv4 address of the requester
    - usually when requests come into the backend server the source IP is the load balancer, but if the server needs the IP of the original requester this is when the header is used
    - XFF is a standardized header used to identify originating IP add of a client connecting to a web server through a HTTP proxy or LB
- Health Checks
    - used to route traffic away from unhealthy instances to healthy ones
    - instances monitored by ELB which reports back health check status as InService or OutOfServices
    - ELB health checks talk directly with the instance to determine state
    - ELB **does not** terminate unhealthy instances, just routes around them
- Cross-Zone Load Balancing
    - only for CLB and NLB
    - when enabled, requests distributed evenly across instances in all enabled AZ
    - when disabled, requests are distributed evenly across instances in only its AZ



- Request Routing
    - apply rules to incoming requests then fwd or redirect that traffic
    - can use
        - host header

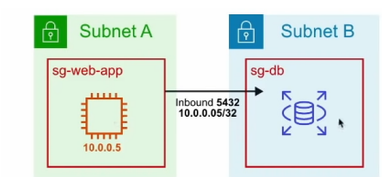
        - http header
        - src ip
        - http header method
        - path
        - query string
    - use to route traffic based on subdomains



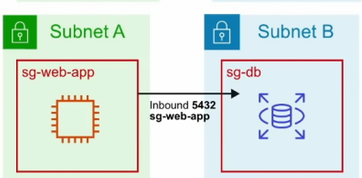
- Cheat Sheet
    - 3 ELB types - Network, application, Classic
    - an ELB must have at least 2 AZ to work
    - ELB cannot go cross region, create 1 per region
    - ALBs has listeners, rules, and target groups to route traffic
    - NLBs use listeners and target groups to route traffic
    - CLBs use listeners and Ec2 instances are registered directly as targets
    - use-cases
        - ALB is for HTTP(S) traffic and good for web apps
        - NLB for TCP/UDP and good for high throughput apps like games
        - CLB is legacy and not recommended
    - use X-Forwarded-For (XFF) to get original IP of incoming request passing through ELB
    - can attach WAF to ALB but not NLB/CLB
    - can attach Amazon Cert Mgr SSL to any ELB types for SSL
    - ALB has advanced request routing rules - route based on subdomain header, path, and other HTTP(S) info
    - use sticky sessions on CLB or ALB which allows it to be sent to a specific host which remembers using a cookie

Security Groups

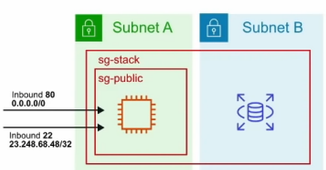
- Introduction
  - virtual fw that controls traffic to/from an instance
  - deployed at/associated to the instance level (host fw)
  - each SG contains a set of rules that filter inbound and outbound traffic to/from instances
  - provides security at protocol/port level
  - no deny rules, all traffic blocked by default
  - multiple instances across multiple subnets can belong to a security group
  - rules evaluated in totality (no order)
- Use Cases



You can specify the source to be an IP range or
A specific ip (/32 is a specific IP Address)



You can specify the source to be another security group



An instance can **belong to multiple Security Groups**, and rules are **permissive** (instead of restrictive). Meaning if you have one security group which has no Allow and you add an allow to another than it will Allow.

- Limits
  - up to 10k SG in a region (default 2,500 but can extend with service limit increase request)
  - 60 inbound + 60 outbound rules per SG
  - 16 SG per ENI (default is 5) - how many SG per instances = SG * number of ENI for that instance
- Cheat Sheet
  - SGs act as a stateful firewall at the instance level
  - all inbound traffic blocked by default
  - all outbound traffic allowed by default
  - can specify source to the an IP range, a single IP, or another SG
  - is a stateful firewall
  - any changes to SG rules take effect immediately
  - instances can belong to multiple SGs
  - SGs can contain many instances

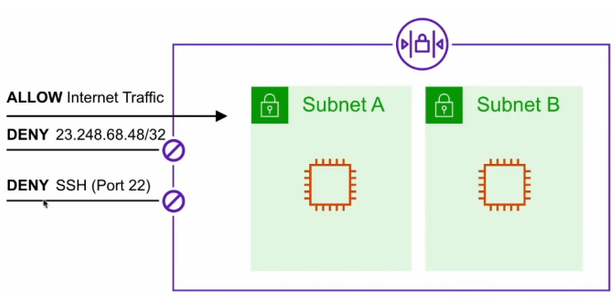
- no deny rules or ability to block specific IP addresses, use NACLs instead
- up to 10k SG per region (default 2500)
- 60 inbound + 60 outbound rules per SG
- up to 16 SG per ENI (default 5)

NACLs

- Introduction
  - optional layer of security that acts as a firewall for controlling traffic in/out of subnets
  - acts as a virtual fw at the subnet level
  - VPCs automatically get a default NACL
  - NACL rules can allow or deny traffic in/out of subnets
  - NACL rules have a # which determines order of evaluation starting from lowest to highest rule #
  - NACLs are associated to subnets
  - subnets can only be associated to a single NACL
  - can block single IP addresses
- NACL Use Case

We determine there is a malicious actor at a specific IP address is trying to access our instances so we block their IP

We never need to SSH into instances so we add a DENY for these subnets. This is just an additional measure in case our Security Groups SSH port was left open.



- Cheat Sheet
  - VPCs are automatically given a default NACL which allows in/out traffic
  - each subnet in a VPC must be associated with a NACL
  - subnets can only be associated with 1 NACL at a time, associating a subnet to a new NACL will remove the previous association
  - if a NACL is not explicitly associated with a subnet, the subnet is auto assigned to the default NACL
  - has inbound + outbound rules like SGs
  - rules can either allow or deny traffic (unlike SGs which only gives allow rules)
  - NACLs are STATELESS

    - denies all traffic by default
    - rules checked from lowest to highest
    - can be used to block single IP addresses (can’t with SGs)

VPC Flow Logs

- Create VPC IGW Route Tables and Subnets
    - 7:58:21
- Launch an EC2 Instance
- Groups NACL and Bastion
- NAT
- Flow Logs
- Cleanup

IAM

- Introduction
- Core Components
- Types of Policies
- Policy Structure
- Password Policy
- Access Keys
- Multi Factor Authenication
- Temporary Security Credentials
- Indentity Federation
- STS
- AssumeRoleWithWebIdentity
- Cross Account Roles
- Follow Along
- Cheatsheet

CloudFront

- Introduction
- Core Components
- Distributions
- Lambda Edge

- Protection
- Follow Along Create
- Follow Along Serve

CloudTrail

- Invalidate
- Cheat Sheet
- Introduction
- Event History
- Trail Options
- CloudTrail to CloudWatch
- Management vs Data Events
- Follow Along Overview
- Follow Along Create A Trail
- Follow Along CloudWatch to CloudTrail
- Follow Along CloudTrail to Athena
- Cheat Sheet

CloudFormation

- Introduction
- Template Formats
- Template Anatomy
- Quick Starts
- Stack Updates
- Prevent Stack Updates
- Nested Stacks
- Drift Detection
- Rollbacks
- Pseudo Parameters
- Resource Attributes
- Intrinstic Functions
- Ref And Get Attr
- Wait Conditions
- Follow Along

- Cheat Sheet

CDK

- Introduction

SAM

- Introduction
- Sam Vs CloudFormation
- SAM CLI

CI / CD

- Introduction
- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- Cheat Sheet

CodeCommit

- Introduction
- Key Features

Docker

- Introduction
- Dockerfile
- Docker Commands

CodeBuild

- Introduction
- Workflow
- BuildEnviroments
- Use Cases
- Buildspec yml
- CheatSheet

CodeDeploy

- Introduction
- Core Components
- In Place
- Blue Green
- appspec
- Lifecycle Hooks
- CodeDeployAgent ServiceRole
- Follow Along - Part 1

Code Pipeline

CodeStar

RDS

- Introduction
- RDS Encryption
- RDS Backup
- Restoring Backups
- Multi-AZ
- Read Replicas
- Multi-AZ vs Read Replica
- Follow Along / Hands-On
- Cheat Sheet

S3

- Introduction
- Storage Classes
- Storage Class Comparsion
- S3 Security
- S3 Encryption
- Data Consistency
- Cross-Region Replication
- Versioning

- Lifecycle Management
- Transfer Acceleration
- Presigned URLs
- MFA Delete
- Create and Delete Bucket - Follow Along
- Upload Files and Make Public - Follow Along
- Versioning - Follow Along
- Encryption - Follow Along
- S3 CLI - Follow Along
- Lifecycle Policies - Follow Along
- Cross-Region Replication - Follow Along
- Bucket Policies - Follow Along
- S3 Cheatsheet

ElastiCache

- Introduction
- Caching Comparison
- Cheatsheet

AWS Lambda

- Introduction
- Lambda Use Cases
- Triggers
- Pricing
- Interface
- Defaults and Limits
- Cold Starts
- Versioning
- Aliases
- Layers
- Cheatsheet

API Gateway

- Introduction

- Key Features
- Configuration
- Caching
- CORS
- Same Origin Policy
- CheatSheet

Step Function

- Introduction
- Use Cases
- States
- Follow Along
- Cheatsheet

Developer Associate CheatSheet