**Route53**

* DNS manager, like name.com on AWS
* Fully managed DNS provider that works well with AWS services
* 4 types of records supported
  + A: hostname to IPv4
  + AAAA: hostname to IPv6
  + CNAME: hostname to hostname
  + Alias: hostname to AWS resource (AWS special)
* Global service (not region specific)
* For ELB use Alias
* TTL for DNS (URL to IP cache)
* Lower TTL before migration
* CNAME works only for non-root domains
* You can't create a CNAME record that has the same name as the hosted zone (the zone apex)
* https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/resource-record-sets-choosing-alias-non-alias.html
* Routing policies -
  + Simple (A record, can add more than 1 IP address)
  + Weighted distribution
  + Latency based
  + Failover
  + Geolocation
  + Geoproximity
* ELBs have no IP addresses
* Alias best used with ELB (free)
* When domain name is bought elsewhere, change nameservers to point to AWS ones

**IAM**

* Contol AWS resource access. Supports MFA, password policies, federation
* For humans - Users, Groups. For AWS resources / external users - Roles
* All access managed by policies (which is a JSON document)
* Users can be in multiple groups
* Use principle of "least privilege" always
* IAM is global
* Need access key + secret key for API/SDK access
* Use MFA for root account, never use it for API access
* Policy types -
  + Managed policies – Standalone identity-based policies that you can attach to multiple users, groups, and roles in your AWS account. There are two types of managed policies:
  + AWS managed policies – Managed policies that are created and managed by AWS.
  + Customer managed policies – Managed policies that you create and manage in your AWS account. Customer managed policies provide more precise control over your policies than AWS managed policies.
  + Inline policies – Policies that you add directly to a single user, group, or role. Inline policies maintain a strict one-to-one relationship between a policy and an identity. They are deleted when you delete the identity.
* Use IAM Policy Simulator to test Users, Groups, Roles and their Policies
* https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html

**EC2 / EBS / ELB**

* **EC2**
* Basically a VM on cloud
* Pricing
  + On-demand
  + Reserved (highly discounted)
  + Spot (cheap but un-reliable)
  + Dedicated (most expensive)
* Instance types based on CPU, memory, GPU, ML needs
* Security Group (SG) = Firewall
* Supports EBS volumes (like a USB stick)
* EBS is like a general purpose SSD attached / mounted
* Amazon EC2 uses an instance profile as a container for an IAM role
* UserData = initial setupo script
* curl http://169.254.169.254/latest/meta-data/ gives instance meta-data
* **EBS**
* EBS supports encryption
* Can be added with zero downtime
* IOPS special EBS (faster)
* IO2 latest generation
* IOPS upto 64000 IO Ops/sec
* Use HDD when there is lot of data and speed does not matter
* EBS can be snapshot-ted
* EFS (like a network file share) -> multi AZ but expensive
* **ELB**
* Distributes load
* 3 options - ALB (http/s, websocket, Layer 7), Network (faster, Layer 4, TCP), Classic (old)
* Use x-forwarded-for to get original IP of request
* Can do health checks
* Can do stickiness with cookies (can cause imbalance)
* Can setup across AZ
* Private ELB also supported
* Can restrict traffic only from ELB via SG
* Supports ECS, even Lambdas
* Uses "Target Groups" for load balancing
* Can manage TLS certs using ACM
* You can upload own certs
* Not sure about Let's Encrypt??
* Connection draining - Before deregistering, time to complete in-flight requests (1-3600 seconds)
* 50x errors when underlying resource is not available

**S3**

* Object storage
* Not a FS
* No OS/DB can run
* Objects upto 5 TB in size
* Universal namespace
* Like a key-value store for objects
* https://bucket-name.s3.region.amazonaws.com/key-name
* 99.95 - 99.99% availability
* 11 9's durability
* Supports versioning
  + At bucket level
  + Ability to restore to old version
* Encrypted
  + SSE-S3 (S3 managed)
  + SSE-KMS (KMS managed)
  + SSE-C (Customer provided keys)
  + Client side (user manages)
  + TLS for transport
  + In order to enforce object encryption, create an S3 bucket policy that denies any S3 Put request that does not include the x-amz-server-side-encryption header.
* Storage classes (keeps changing)
  + Standard (3 AZ)
  + Standard-IA
  + Standard-1Zone (lower cost)
  + S3 Glacier
  + S3 Glacier Deep Archive
  + Intelligent Tiering
  + Reduced Renundancy
* Security
  + By default all buckets private
  + Apply bucket policy
  + Bucket ACL (at object level)
* Can provide access logs
* Can be used for websites
  + Enable website
  + Un-block public access
  + Change bucket policy
  + Supports CORS
* AWS Athena -> Analytics queries on S3 (supports CSV, JSON, Avro etc.). Usecase e.g. logs on S3

**Cloudfront**

* AWS CDN
* Origin can be S3, EC2, ELB, R53
* Edge locations
* TTL of deefault 1 day
* Cached can be claered manually but charged
* Edge locations can be "write" also
* S3 can be configured to be only accessible by CF
  + OAI
  + Bucket policy
* Geo-restrictions are supported
* HTTPS can be forced
* First request, objects are not cached at edge locations so can be slow
* Seamlessly integrates with AWS Shield, AWS Web Application Firewall
* AWS Certificate Manager (ACM) can be used to easily create a custom SSL certificate and deploy to an CloudFront distribution for free.
* Signed URLs, signed cookies and token based authentication is supported to restrict access to only authenticated viewers.

**RDS**

* Basically fully-managed RDBMS on cloud
* Supports - MS SQL Server, Oracle, MySQL, Postgres, MariaDB and of-course Aurora
* Works well in VPC
* Automated backups (enabled by default, retention of 1-35 days)
* Backup window can be user defined
* Can take DB snapshot, store on S3
* Snapshot + Transaction log = point in time recovery
* Encryption -> at creation time only,
* Encryption can use AWS KMS (AES-256)
* If master is not encrypted, read-replicas cannot be (same for snapshot)
* To convert an unencrypted DB to encrypted -> Take shapshot, encrypt snapshot, create encrypted DB from new shapshot, delete old DB
* Snapshots can be used to copy data across regions
* 2 setups -
  + Multi-AZ
  + Read Replicas
* Multi-AZ
  + Exact copy in another AZ
  + Sync replication
  + Standby copy
  + Automatic failover
  + Only for disaster recovery, no positive impact on performance
  + Zero downtime switchover
* Read replicas
  + Read-only copy
  + Can be within an Availability Zone, Cross-AZ(?), or Cross-Region
  + Point to read-replica for read-only queries (lowers load)
  + Read-replicas can be used for OLAP
  + Upto 5 read-replicas (more for Aurora)
  + Reads are eventually consistent
  + Read-replica can be promoted to it's own separate DB (kind of DR then)

**Elasticache**

* AWS managed Memcached or Redis
* Helps in read-heavy operations
* Redis can be multi-AZ (but is single threaded)
* Not helpful in write heavy systems
* Amazon ElastiCache with Memcached includes sharding to scale in-memory cache with up to 20 nodes and 12.7 TiB per cluster
* Only Memcached has multithreaded architecture
* But Redis supports snapshots, transactions, pub-sub etc.

**AWS VPC**

* Virtual private cloud
* Great for isolation
* Dedicated VPCs are costly
* Regional AWS resource
* Create VPC -> Subnets (public / private)
* Internet Gateway (for public subnet, use routing table). 1 IGW per VPC. All internet traffic goes through it.
* NAT Gateway (for private subnet, lives in public subnet, AWS managed, routing for nodes in private subnet)
* NACL
  + Firewall for subnets
  + ALLOW and DENY rules
  + Works on IP addresses
* VPC flow logs - Network traffic (read https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html)
* VPC peering (read more on AWS)
  + Restriction: you cannot create a VPC peering connection between VPCs that have matching or overlapping IPv4 or IPv6 CIDR blocks.
* Best for traditional 3 tier apps (web, app, DB servers)

**X-Ray**

* Analyze and debug apps
* Provides visualization / service map
* End-to-End view of a request path
* Supports EC2, Lambda, ECS
* Java, Node.js etc. supported
* On EC2 install agent, in app install SDK
* Use annotations to add custom traces
* Lookup segment/sub-segment for analysis

**Lambdas**

* Low cost (million requests free / month)
* Scale seamlessly
* No worry about server / OS etc.
* Pricing on number of req., duration & memory usage
* Event driven architecture, supports events from
  + S3
  + DynamoDB
  + API GW
  + Kinesis
  + SQS/SNS
  + More
* All lambdas by default have execution role with access to CloudWatch for logs
* Can use upto 128MB-3 GB RAM (or 10GB), 15 mins of execution time
* Can use /tmp upto 512MB
* Can be used with ELB (ALB) as well
  + ALB converts HTTP request to JSON event
  + Supports "multi-header values"
  + Also supports health check
* But for HTTP API GW is better choice
* Lambda@Edge (new service, lookup docs)
* Async invocation
  + S3, SNS, CWatch, SQS (uses polling)
  + Retry attempts upto 2
  + DLQ support
* Event source mapping
  + Kinesis
  + SQS
  + DynamoDB
* Batching of events supported (1-10)
* 1 Role for Lambda function is ideal
* Env. variables
  + Can use KMS for encryption
  + Key/Value pair
  + Upto 4 KB
* Supports X-Ray (called active tracing)
* Can be launched in VPC (needs more investigation)
* Deploying lambda in public subnet does not give internet access (use RT + NAT GW)
* vCPU given according to RAM (1792 MB = 1 CPU)
* Default timeout is 3 secs
* "Execution Context" for some sort of cached code (outside function handler)
* Limit of 1000 concurrent invocations
* See "Reserved Concurrency" vs "Provisioned Concurrency" in more detail
* Code can be 50MB compressed ZIP, or 250 MB unzipped
* Layers to support large bundle, can use custom runtime (e.g. Rust)
* Works well with CloudFormation - (define inline in CF template) or Store zip in S3 and reference in CF
* Lambda now supports Docker images (should use AWS base image)
* Versions & Aliases
  + $LATEST points to latest and is default
  + Versions are unique & immutable
  + ALIAS points to a version e.g. Prod -> v1
  + Use this for deployment (blue/green)
* HTTP 429 - Too many requests
* Can cron execute (with CW events setup)
* New feature - StepFunctions - can design / declare State Machines with serverless components

**DynamoDB**

* NoSQL, AWS managed DB
* Large scale, single-digit latency
* Speads data in partitions across DCs (max 10GB / partition)
* Row can be max 400 KB
* Eventually consistent reads by default
* Supports ACID txns.
* Supports JSON, HTML & XML docs
* Primary Key (also called Partition Key) hash used to determine partition
* 2 types of keys
  + Partition key
  + Composite key = Partiton Key + sort key (e.g. user id & timestamp)
* No need to define schema upfront
* Create table using API/SDK/CF etc.
* Access managed by IAM, can even control upto row level access
* Indexes
  + Secondary Indexes
    - Global secondary index (can create anytime) (like a view)
    - Local secondary index (at table creation time only) upto 5
* Query very efficient on PK
* Projection expression (only select few cols)
* Scan examines all items (can be slow on large tables)
  + Use parallel scans
  + avoid big scans
* Prebook Read Capacity Unit / Write Capacity Unit
  + 1 WCU = 1 write / sec for 1 KB of data
  + Strong RCU = 1 read / sec for upto 4 KB data
  + Eventual RCU = 2 reads / sec for upto 4 KB data
  + e.g. 10 strong reads/sec of 4 KB needs 10 RCU
  + 16 normal reads/sec of 12 KB needs (3 \* 8 = 24 RCU)
* Can also use on-demand capacity but then hard to predict cost
* For strongly consistent reads use "ConsistentRead" param in query
* Basic API
  + PutItem (create / full-update)
  + UpdateItem (partial update)
  + DeleteItem
  + DeleteTable
  + ConditionalWrites
  + BatchWrite (uto 25 items)
  + GetItem (PK based query)
  + BatchGetItem (upto 100 items)
* Scan upto 1MB/page, consumes RCUs
* Concurrent access
  + default is optimistic locking
  + Use conditional update / delete
  + set version field
* Txns.
  + All or nothing
  + API - TransactWriteItems or TransactGetItems
* If PK is bad, too few partions or imbalanced partitions (use random suffix etc.)
* Can encrypt data with KMS
* Can work in VPC (need to read more)
* DynamoDB accelerator (DAX) in memory cache
  + Improves read perf.
  + Write to DAX + DB
  + Read from DAX
  + Like ElastiCache for DynamoDB
* TTL
  + Delete old records
  + Expiry time for data
  + Expressed in Unix epoch
  + Use an attribute in the row
  + Deleted within 48 hours after expiry
* Streams
  + Time ordered changes in seq.
  + Item level changes
  + Like a log
  + Before/After image of item
  + Like a DB trigger
  + Encrypted / available for 24 hours after item change
* If provisioned throughput is exceeded
  + Exception thrown
  + Use SDK for retries
  + Or reduce requests + exponential backoff

**ECS / Fargate**

* Traditional ECS needs a EC2 cluster
* Works well with VPC
* Creates an ASG internally
* Each EC2 machine needs ECS agent (config in /etc/ecs/ecs.config)
* Agent itself runs in a Docker container
* ECS -> Cluster -> Service -> Task definition -> Container defintion(s)
* Task definition
  + JSON
  + Docker image
  + Ports mapping
  + Env. vars
  + Memory / CPU
  + Networking info
  + IAM role
  + Add volume
* Service
  + Add task defns.
  + Deployment strategy (AWS managed)
    - Rolling
    - Blue green
  + Code Deploy can be used
  + Choose ALB
* Scale number of tasks
  + But what about ports
  + With ALB leave host port mapping blank
  + ALB will manage port mapping
  + Can only add ALB on service creation
  + Using ALB good practice
* ECR
  + Private docker registry
  + Manage with IAM
  + $(aws ecr get-login) or aws get-login-password
* Fargate
  + Best option
  + No need to manage EC2 machines / Cluster
  + Fargate with ALB simple scaling without managing EC2
* ECS -> IAM roles at 2 level - EC2 or task definition
* ECS task placement
  + where to place tasks when scaling
  + Binpack: place with least CPU/memory usage (cheap)
  + Random
  + Spread (AZ, InstanceID)
* ECS constraints
  + DistinctInstance
  + memberOf (Cluster Query Language e.g. type M4)
* Services can auto scale like ASG
* Cluster auto scales with Capacity Providers (?)
* ECS has EBS on EC2
* Docker can use EBS
* Use EFS to share volumes
* Fargate + EFS = high-scale, serverless data storage
* Sidecar pattern = 2 containers in a task defn. + volume bind-mount(?)
* EKS = Managed K8 on AWS (if you really want to use K8)

**SQS / SNS**

* SQS = AWS managed MQ service
* SQS is Pull based SNS is push based
* At-least 1 read guaranteed
* 256 KB of text
* 1 million messages / month free
* Default retention of 4 days / max. 14 days
* SQS
  + Standard (unlimited msgs/sec)
  + FIFO (ordered but max 300 msgs./sec)
* FIFO uses
  + MessageGroupId (for grouping consumers)
  + MessagingDeduplicationId (can use message SHA). 5 mins same message not repeated
* API
  + CreateQueue
  + DeleteQueue
  + PurgeQueue
  + SendMessage(DelaySecs param etc.)
* Visibility
  + Time message is not visible to others once picked up
  + default 30 secs
  + max 12 hours
* Short / long polling
  + Long polling is better 1-20 secs
  + Set in queue read API (WaitTimeSeconds param)
* Sometimes delay is needed. max of 900 secs
* Large messages use S3, feature available in Java SDK
* Best to keep consumers idempotent
* SNS
  + Push notifications (Apple, Google)
  + SMS
  + Email
  + Trigger lambda
* SNS is like a topic
* SNS can filter msgs.
* SES (Simple Email Service) - Old, used for emails only. Can trigger Lambdas

**API GW**

* Powerful but complex feature
* Great for internet facing APIs
* Works will with lambdas but also any AWS service with HTTP like ELB
* Supports auth, websockets, versioning, throtting, rate limiting
* Edge optimized like CFront
* Can also be used in private in VPC
* Create API -> Resources -> Actions -> Handler
* Supports proxy mode, no req./response transformation
* Max timeout 29 secs
* For deployment, create a stage e.g. dev, test, prod
* Stages can also be v1, v2
* Stage variables supported (look into more details). Works great with Lambda Alias
* Use stage vars. + aliases for blue/green deployment (see docs/demo)
* At stage level
  + Define caching policy (default 300 sec)
  + Throttling
  + Logging
  + X-Ray integration
* Can also use with canary deployment strategy (see hands-on)
* Types
  + Mock (for testing)
  + AWS (Lambda etc.)
* Mapping templates
  + Req/Resp transformation
  + Can use with SOAP services
  + Can change headers, query params etc.
* Swagger
  + Can import/export
* Usage plans can be setup for throttling, setting up API keys, quotas
* 10000 rps across an API
* Cloudwatch integration at stage level
* Supports CORS
* Setup policy to integrate with other AWS services

**KMS**

* AWS managed service for encryption keys
* Can use across AWS services like S3, RDS, etc.
* Customer master keys are the primary resources in AWS KMS.
* A customer master key (CMK) is a logical representation of a master key.
* The CMK includes metadata, such as the key ID, creation date, description, and key state. The CMK also contains the key material used to encrypt and decrypt data.
* AWS KMS supports symmetric and asymmetric CMKs. A symmetric CMK represents a 256-bit key that is used for encryption and decryption. An asymmetric CMK represents an RSA key pair that is used for encryption and decryption or signing and verification (but not both), or an elliptic curve (ECC) key pair that is used for signing and verification.
* Types
  + Customer managed CMK
  + AWS managed CMK
  + AWS owned CMK
* Lookup CloudHSM (single tenant model)
* Region specific service
* Use CMK to generate data key to encrypt large amounts of data (envelope encryption)
* KMS handles rotation, audit (CloudTrail)
* Roles - administrators / users
* GenerateDataKey API

**CloudFormation**

* Infra as code
* JSON/YAML
* Types
  + Resources
  + Parameters (provide input)
  + Mappings (static declared variables)
  + Outputs
  + Conditionals
  + Metadata
  + Reference
  + Functions
* Outputs can be imported / exported
* Fn:GetAtt
* Fn:Ref => !Ref
* All rollback by default if stack creation fails
* Can disable rollback
* Changeset - tells us what will change
* Read more: Cross vs NestedStack vs StackSet (important)
* Drift: when resources changes outside CF
* Create template -> S3 -> API calls -> create stack -> delete stack

**SAM**

* Extension to CF for serverless
* Simplified CF
* Separate CLI
* sam package
* sam deploy
* Main thing - Transform:AWS::Serverless-2016-10-31
* Supports
  + Function
  + API
  + SimpleTable
* Look into AWS FAQ

**CDK**

* Use the CDK to define your cloud resources in a familiar programming languages.
* See Contructs/Patterns

**Kinesis**

* Collect, process, analyze streaming data in real time
* Supports
  + Data Stream
  + Data Firehose
  + Data Analytics
  + Video Streams
* Data Stream
  + Made up of shards
  + e.g. a stream of 10 shards
  + Shards divide volume
* Kinesis producers push records -> PK determines shard -> 1000 msg/sec/shard or 1 MB/sec
* As many shards as you want
* Retention 1 day - 1 year
* Can replay data
* Kinesis Producer Library
* Consumer
  + Lambdas
  + SDK
  + KCL
  + Enhances fan-out consumer (higer cost) higher throughput
  + max EC2 instances = number of shards

**Cognito**

* Like Auth0 + IAM integration
* Web identity federation
* User pool - Authentication, Identity pool - Authorization (integrates with IAM)
* Supports triggers (lambda on pre-sign up/post auth etc.)
* "LeadingKey" for policy at the resource level (e.g DynamoDB)
* STS
  + Security Token Service
  + Grant temp. access to AWS
  + API AssumeRole/AssumeRoleWithSAML/AssumeRoleWithWebIdentity

**AppSync**

* New service for GraphQL
* Supports websockets + MQTT
* Old service: Cognito sync
* Integrates well with Lambda + DynamoDB
* e.g. chat service

**ASG**

* Auto scale
* Can specify min/max size
* Needs a launch config
  + AMI + Instance type
  + EC2 UserData
  + EBS volume
  + SG
  + KeyPair
* Can be setup based on CloudWatch alarms e.g. CPU usage or custom metrics
* Can also schedule scale up/down

**Cloudwatch / CloudTrail**

* Monitoring service
* Used for logs / metrics / perf. checks
* Has a default reporting time (5 mins?)
* Can setup alarms
* Needs CW agent on EC2 instance
* See CloudWatchAgentServerPloicy
* Configure the agent
* CloudTrail is at AWS level, audit AWS actions / accounts
* AWS EventBridge - see events from SaaS like DataDog

**ParameterStore / SecretsManager**

* ParameterStore is secure storage for config / secrets
* Works with KMS
* Supports versioning
* Can create tree
* TTL supported
* SecretsManager similar but also supports
  + Rotation
  + RDS integration

**BeanStalk**

* BeanStalk - Like heroku for AWS
* Handles ELB, EC2, S3, RDS etc.
* Just "upload" code
* AWS managed platform / OS
* Can keep 1000 versions / delete old ones
* App -> Environment -> Version
* You can create a load-balanced (min-max instances), scalable environment or a single-instance environment
* Now also supports Docker, even multi-container
* Elastic Beanstalk creates an Amazon S3 bucket named elasticbeanstalk-region-account-id for each region in which you create environments.
* Supports worker environment with SQS support to handle async tasks
* You can add AWS Elastic Beanstalk configuration files (.ebextensions) to your web application's source code to configure your environment and customize the AWS resources that it contains. Configuration files are YAML- or JSON-formatted documents with a .config file extension that you place in a folder named .ebextensions and deploy in your application source bundle.
* example .ebextensions/network-load-balancer.config
* Has a CLI
* AWS Elastic Beanstalk provides several options for how deployments are processed, including deployment policies
  + All at once
  + Rolling
  + Rolling with additional batch
  + Immutable
  + Traffic splitting
* By default, your environment uses all-at-once deployments
* Blue/Green also supported (but not out-of-box, use clone + SWAP URL)
* Good practice to create RDS separately so that data is not lost when EB is terminated (e.g. .ebextensions/db-instance-options.config)
* Can setup HTTPS

**CI/CD**

* CodeCommit
  + Like GitHub
  + SNS Integration - delete branch / master branch changes
  + CloudWatch events - PRs / Comments / Commits -> Can notify SNS
* CodeBuild
  + Like CircleCI/Jenkins
  + Pay per use
  + No build queue
  + Uses buildspec.yml
  + Cannot SSH into build servers/containers
  + Can build Docker images + push them to ECR
* CodeDeployment
  + Most complex, can setup all sorts of deployments
  + Uses appspec.yml file
  + Like Ansible / Chef
  + Provides hooks
    - Before + After for -> Install / AllowTestTraffic / AllowTraffic
    - Also DownloadBundle, ApplicationStart, ValidateService
* CodePipeline
  + Umbrella service
  + Made up of stages -> Action groups
  + If stage fails, pipeline fails
* CodeStar
  + New service

### Management Tools

* [CloudWatch](https://jayendrapatil.com/aws-cloudwatch-overview/)
  + monitoring to provide operational transparency
  + is extendable with custom metrics
  + CloudWatch -> (Subscription filter) -> Kinesis Data Firehose -> S3
* [CloudTrail](https://jayendrapatil.com/aws-cloudtrail/)  
  + helps enable governance, compliance, and operational and risk auditing of the AWS account.
  + helps to get a history of AWS API calls and related events for the AWS account.
* [CloudFormation](https://jayendrapatil.com/aws-cloudformation/)
  + easy way to create and manage a collection of related AWS resources, and provision and update them in an orderly and predictable fashion.
* [AWS Config](https://jayendrapatil.com/aws-config/)
  + fully managed service that provides AWS resource inventory, configuration history, and configuration change notifications to enable security, compliance, and governance.

### Integration Tools

* [Simple Queue Service](https://jayendrapatil.com/aws-sqs-simple-queue-service/)
  + as message queuing service and SNS as pub/sub notification service
  + as a decoupling service and provide resiliency
  + SQS features like visibility, and long poll vs short poll
  + provide scaling for the Auto Scaling group based on the SQS size.
  + [SQS Standard vs SQS FIFO](https://jayendrapatil.com/aws-sqs-standard-vs-fifo-queue/) difference
    - FIFO provides exactly-once delivery but with low throughput
* [Simple Notification Service – SNS](https://jayendrapatil.com/aws-sns-simple-notification-service/)
  + is a web service that coordinates and manages the delivery or sending of messages to subscribing endpoints or clients
  + Fanout pattern can be used to push messages to multiple subscribers