EC2 + IAM

*Region - Us-east(Ohio) AZ: US-east-2 AZ are separate from other AZs to prevent disaster

*AWS consoles are region scoped except R53,IAM & S3)

*IAM (Identity & Access Mgt) - Root user never used,IAM policies are written in JSON,GLOBAL

*IAM - give users least privileges as possible, has predefined managed policies

*IAM Federation - for big companies' users, uses SAML method (similar to AD)

*IAM - Don't use in code, one IAM user per person, one IAM role per app

*EC2 - Virtual machine, ssh -i EC2tutorial.pem ec2-user@3.20.2-22.124

*EC2 security groups - Network security in AWS (Inbound/Outbound traffic), acts as firewall

*EC2 security groups - Can be attached to multiple instances,Locked down to region/VPC combo, If app times out then its a sec grp issue, if connection refused error then its a app error or some other issue,use separate one for ssh, all outbound traffic is allowed by default, all inbound are blocked by default. Security groups can have other security groups referenced.

*Public IP - unique across whole web, can be identified on internet

*Private IP - can be identified on company's network only

*Elastic IP - once the instance is stopped it loses its IP. So use ElasticIP as its fixed public IP.Attached to one instance at a time. It can be remapped to another instance. Max upto 5 elastic IPs. Poor arch decision.

*EC2 User data - set of things to run when the instance boots up. Runs only at startup. Runs with user's root user

*EC2 Launch Types - OnDemand, Reserved (1 to 3 yrs) for dbs, Convertible Reserved, Scheduled Reserved(every sunday football game), Spot Instances(bid price, will lose if outbid, 2 min notification to shutdown),Dedicated Instances (may share hardware),Dedicates hosts (will not share hardware,BYOLicense)

*Ec2 pricing - per hr, region in , instance type,OS. Also pay for factors like data trnsfer, storage, load balancers, fixed IP public addresses

*AMI - Amazon Image,Custom AMIs - no need for user data, packages/apps are already installed so faster deploy time. Built for specific region.

*EC2 characteristics - GPU,CPU,RAM,I/O,Network. M instances are balanced. R instances are heavy in RAM. T2/T3 are burstable instances which when need more CPU it will perform. You get burstable credits. T2 unlimited have unlimited burstable credits.

Route 53 + RDS + ElastiCache + VPC

Route 53 - Managed DNS.Redirects URLs. Common are A : Url to IPV4 AAAA: URL to IPV6. CNAME: URL to URL and Alias: AWS resource. Use alias over CNAME for AWS resources.

RDS - Relational Database Service. Managed DB service.use SQL as query language. Allows you to create dbs on the cloud that are managed by AWS. Oracle,PostGRE,MSSQL,Maria etc and Aurora (AWS) are supported.

RDS advantages - Managed service, OS patching, Continous backups and Restore to point in time, Monitoring dashboards, Read replicas, Multi AZ setup for disaster recovery,Scaling (vertical & Horizontal). BUT can't ssh into instances.

RDS Read replicas - Upto 5 replicas within same or cross AZ or cross region. Replication is ASYNC so reads are eventually consistent. Replicas can be promoted to own DB. Apps must update the connection string to leverage read replicas.

RDS Multi AZ (Disaster recovery) - SYNC replication. One DNS name - auto app failover to standby. Increase availability. Failover in case of loss of AZ, loss of network, instance or storage failure. Not used for scaling. No manual intervention in apps.

RDS backups - Auto-enabled. Daily full snapshot of the db.Capture transaction logs in real time.Ability to restore to any point in time. 7 days retention. Can be increased to 35.

DB Snapshots - Manually triggered by user. retention of backup as long as you want.

RDS Encryption - At rest with KMS. SSl certs to encrypt data in flight. To enforce SSL in postgre rds.force ssl in parameter groups. In MYSQL grant USAGE on '.' TO 'mysql'@' REQUIRE SSL.

RDS Security - Deployed in private subnets not in public. Security is enforced by security groups similar to EC2 instances.Sec grps controls who can communicate with RDS. IAM controls who can manage RDS. Traditional Username and password can be used to login to db. IAM users can now be used for MySQL/Aurora.

RDS vs Aurora - Aurora is expensive but efficient. Aurora can have 15 replicas and replication process is faster. Postgres and MySQL are supported as Aurora. Means drivers will work as if Aurora was a Postgre or MySQL db. Aurora is cloud optimizied.

ElastiCache - is for managing in-memory dbs Redis or Memcached as RDS is for Relational dbs.

Caches are in-memory dbs with high performance, low latency. Helps reduce load off read intensive workloads. makes app state less. Write scaling using sharding. Read scaling using replicas.

Route 53 + RDS + ElastiCache + VPC (cont)

Elasticache Architecture - Apps reads data from EC and if not present reads data from RDS and writes data to EC. Cache must have an invalidation strategy to make sure only most current data is used in there.
Redis - in-memory key value store. Super low latency. Cache survives reboot by default.(Persistence).Support for read replicas. Multi AZ with auto failover.
Memcached - Cache dpesnt survive reboots.
EC patterns - Lazy loading & Write through.
Laxy loading pros- load only when necessary. Cache isn't filled with unused data. Node failures are not fatal.
Lazy loading cons - Cache miss penalty results in 3 trips. Noticeable delay for that miss. Data can be updated in db but outdated in cache.
Write through pros - Data in cache is never stale. Write vs Read penalty (each write requires 2 calls).
Write through cons - Missing data unless it is added/updated in db. Mitigation for this is to implement lazy loading as well. Cache becomes too big.
VPC - Within region you are able to create Virtual Private Cloud. Each VPC contains subnets(networks). Each subnet must be mapped to AZ. Its common to have a private/public subnet.(private/public IP). Its common to have many subnets per AZ.
Public subnets contains LBs,Static websites,Files,public auth layers.
Private subnets contains web app servers, dbs.
Private and public subnets can communicate if they are in the same VPC.

Elastic Beanstalk

EB uses all base AWS components like EC2,ASG,IAM,LB,VPC,RDS etc. Dashboard to view all these.
EB is free but pay for underlying instances.
Instance config/OS is handled by EB and Deployment is handled by EB but can be configured.
3 arch models - Single instance (Dev) ,ASG + LB (prod/pre-prod) , ASG (non-web prod apps)
EB has Environments & Applications.Every deployment gets an application version. You can deploy application versions to Environments and can promote application version to next environment. Rollback feature to get to prev app version.
EB supports many platforms and Single/Multi docker containers.
Deployment Options - All at once,Rolling, Rolling with additional batches, Immutable

Elastic Beanstalk (cont)

All at once - Fastest Deployment, Downtime, Great for quick iterations in dev, No additional cost
Rolling - Application is running below capacity, can set bucket sizes, will run both apps simultaneously, No additional cost, Will take long time to deploy
Rolling with additional batches - App runs at capacity,Additional cost,will run both apps simultaneously,Longer deployment
Immutable - Zero downtime, New code is deployed to new instance on temp ASG,High cost double capacity,Longest deployment, Quick rollback feature (just terminate new ASG), great for prod.
Blue/Green Deployment - Not a direct feature of EB.Zero Downtime.Create stage environment and release new code there. Route 53 can direct little traffic to new version to test it.
EB Extensions - A zip file containing our code must be deployed to EB. All parameters set in UI can be configured using files. Requirements are that it should be in the .ebextesions/ directory in the root of source code. It should be in YAML/JSON format. .Config extensions. Ability to modify def settings and add resources like RDS,Dynamodb,ElastiCache etc. .ebextesnions will get deleted if environment gets deleted.
EB CLI - command eb create, eb status, eb deploy.
Under the hood EB relies on CloudFormation.
EB Deployment Mechanism - You have to describe dependencies e.g. req.txt fpr python or package.json for node.js. Pacakage code as zip. EC2 will resolve depencies which can be slow. To optmize this package dependencies with code to improve deployment performance and speed.
EB with https - SSL certs can be loaded to LB through EB console or LB config. Can be done by code through .ebextensions/securelistener-alb.config. It can be provisioned through ACM or CLI. Must configure security group rule to allow incoming port 443 (https port).
EB redirect from http to https - Configre instances to do redirect. Configure ALB to redirect with a rule. Make sure health checks are not redirected.
EB lIfecycle Policy - can store at most 1000 app versions.To remove old versions use lifecycle policy based on no of versions or days to retain version. Versions currently used wont be deleted. Option not to delete the source bundle in S3 to prevent loss of data.
Webserver vs worker environment - If your app performs long tasks then decouple the apps to two tiers. One tier is web app tier where work is requested and then using SQS queue it to second tier where the long task is procesed. Periodic Tasks can be defined in cron.yaml file.

Elastic Beanstalk (cont)

EB with RDS - EBS can provision a RDS but this RDS is specific to environment. So its better to decouple and create RDS alone and then provide it to EB with connection string.
To migrate RDS coupled in EB to standalone RDS -
Take RDS snapshot
Enable deletion protection in RDS.
Create new environment without RDS and point to existing old RDS.
Perform blue/green deplyment and swap old/new environment.
Terminate old environment.
Delete cloud formation stack.

AWS Monitoring

CloudWatch
Metrics - Collect &track key metrics. EC2 detailed monitoring - default metrics every 5 mins. for extra cost you can have detailed monitoring at every 1 min. AWS free tier allows upto 10 detailed monitoing metrics. Dimension is an attribute of a metric.(instance id, environment etc). Upto 10 dimensions per metric. Custom metrics : use api call PutMetricData. Use exponential back-off in case of throttling errors.
Logs - Collect, monitor, store and analyze log files. Apps can send logs using SDK. Logs can go to S3 for archival or to ElasticSearchCluster for further analytics. Log storage architecture - Log groups (name rep application), Log streams (instances within app/log files/containers). Log expiration policies. Need IAM policies to write to logs. Encryption of logs using KMS at group level.
Events - Send notifications when certain events happen in AWS. Event pattern - event rules to react to a service doing something. e.g. code pipeline state change. Schedule cron jobs. Triggers to Lambda functions,SQS/SNS/Kinesis msgs. Creates JSON doc to give info about change.
Alarms - React in realtime to metrics/events . Alarm states are OK,INSUFFICIENT_DATA,ALARM. can trigger notifications for any metric.
X-ray - Troubleshooting app performance and errors. Distributed tracing of micro-services.
Visual analysis of apps.
Advantages :
Troubleshooting performance issues
Understand dependencies in microservice architecture.
CloudTrail - Internal monitoring of API calls. Audit changes to AWS resources by your users.

ECS,ECR,Fargate & Docker

*This section is equivalent to ELB/CloudFormation in running docker apps
*ECR - Elastic container registry to store private docker images
*To manage containers we have 3 choices. They are below
*ECS - Amazons own
*Fargate - Amazon's serverless
*EKS - Amazon's own Kubernetes (open source)
*ECS clusters - grp of EC2 instances. Instance run Ecs agent (Docker container) which registers to cluster. When a cluster is created it creates instances with Docker in it. We must configure the file **/etc/ecs/ecs.config** with the cluster name to register instance with the cluster.
*ECS tasks - containers running to create apps.
*ECS task definitions - Json form tells ECS how to run docker container (Env var,CPU,ports etc). Task Role is important as it gives permissions.
*ECS service -app definitions running on ECS cluster. tells how many tasks to run and how to run it. Can link to LB (Dynamic Port Mapping)
*ECS-->Clusters-->Task Def(Container def)/Service-->Tasks
ECR - aws ecr get-login --no include email --region us-east-1a -- login command to authenticate docker to push images to your registry
docker build
docker tag
docker push 3867.dkr.ecr.us-east-1.amazonaws.com/demo:latest
docker pull "imagename/tag" (Push and pull must be preceded by aws ecr login to get docker login credentials).
Fargate - Serverless. No need to add instances.Just create task definition and increase number of tasks running.
ECS + Xray - 1. ECS cluster X-ray daemon 2. ECS cluster X-ray container as sidecar 3. Fargate cluster X-ray container as sidecar .
Portmappings - Port -2000 Protocol - UDP
Environment variable - AWS-X-ray_Daemon_Address : x-ray_daemon_address :2000
Links : X-ray

ECS,ECR,Fargate & Docker (cont)

ECS + EB (Beanstalk) - You can run EB in Single/Multi Docker container mode.It will create ECS cluster ,EC2 instances, task definitions and execution, LB. Requires a config file Dockerrun.aws.json at the root of source code. EB has an option for containers under platform to make this posible. So this is configured when you create EB.

For EC2 instances to run multiple containers you must NOT specify host port. Enable LB dynamic port mapping feature.Security grp should provide LB with access. Sec grps should only work at instance level and not at task level. Task def has IAM roles.

ECS integrations - Can integrate with X-ray. X-ray must run as 2nd container. ECS integrates with cloudwatch logs. You need to set up logging at task def level. Each container will have separate log stream.

CLI command to create ECS service - aws ecs create-service

Integration & Messaging : SQS,SNS & Kinesis

DynamoDb

ELB + ASG + EBS

LB - Spread Load, Single point of access (DNS) to the app, Handle failures of instances, Health checks , SSl (Https) , stickiness (cache), high availability across zones, separate public/private traffic

ELB - takes care of upgrades, maintenance & high availability

ELB Types - ALB(http/https/websockets),NLB (TCP/IP) , Internal (private) and External (public) ELBs

Health Check - They make LBs know if it can forward data to instances. Response 200 - OK.

ALB - awesome for micro services and for containers (apps run on same machine). Has port mapping feature to redirect to dynamic port. Stickiness is enabled by ALB and not app. App servers don't see the client ip directly but see only ALB ip. The client IP is stored in X-Forwarded-For.

LB Target Groups - Instances where LB should direct its traffic.

NLB - TCP traffic. Less latency. Support for static /Elastic IP. Used for extreme performance. Default should be ALB. Both ALB & NLB has static host name. Always use host name and don't resolve underlying IP.

LBs can scale but not instantaneously,

4xx errors are client induced.

ELB + ASG + EBS (cont)

5xx errors are app induced.

LB error 503 means at capacity.

If LBs cant connect to your apps then check security groups.

ASG - Scale out or Scale in instances based on increased/decreased load. Ensure we have minimum/max number of instances running. Automatically register new instances to a LB.

ASG Launch configuration - AMI + Ins type, user data, security grp, EBS volume, SSH key pair

Other ASG attributes - Min/max/initial capacity, Scaling policies, LB info, Network/subnet info.

ASG alarms - Based on cloud watch alarms we can write ASG scaling policies

ASG new rules - New scaling rules based on CPU usage, avg network in/out. Easy to set up and makes more sense.

ASG metric - Integration with cloud watch metrics to get custom metric from EC2 instance to alarm an ASG scaling policy. PUTMetric API.

EBS - Elastic Block Store. Network drive not physical drive. Store instance data here so that you don't lose it when instances are terminated. Can be detached from instance and attached to another. Can be attached to only one instance at a time.Locked to AZ. Provisioned capacity (GB and IOPS) which can be increased over time.

EBS 4 Types - GP2 (SDD General purpose. Balanced) , IOI - Mission critical low latency high performance ,STI (HDD frequently accessed) and SCI (HDD less frequency accessed).

You can resize EBS volumes but need to repartition it.

EBS snapshot - backups. Snapshots backups only actual data. So if 100 GB EBS has only 5GB of data then only that 5 GB is snapshotted. Snapshots can be used for disaster recovery.

EBS encryption - Rest & inflight. Snapshots are also encrypted. All volumes created from snapshot is also encrypted. We don't have to do encryption/decryption.

Instance Store - Instances without EBS volumes. Better I/O.

Migrating EBS across AZ means snapshot and then recreate in another AZ.

EBS backups use high I/O so don't run them when your application is handling lot of traffic.

Root EBS volumes (one that comes with instance) get deleted once instances are deleted. This can be disabled.

Amazon S3

S3 - Simple Service Storage
S3 buckets - directories which stores files. Defined at region level.
S3 objects - are files. They have Key which is the path of the file. Max 5TB. More than 5GB then it should be multi-part upload.
S3 Versioning - Enabled at bucket level. Any file that is not versioned prior to enabling versioning will have version "null".
S3 Encryption - SSE-S3,SSE-KMS,SSE-C,Client Side Encryption
SSE-S3 - encrypt using keys handled by S3 & AWS, Server-side enc,AES-256, Set Header **"x-amz-server-side-encryption" : "AES-256"**
SSE-KMS - encrypt using keys handled by KMS , server-side , KMS gives user control + audit trail , Set Header **"x-amz-server-side-encryption" : "aws : KMS"**
SSE-C - server-side enc by keys managed by customer outside of AWS. S3 does not store enc keys. HTTPS must be used and data key must be provided in every Https header.
Client -side - used SDK such as S3 encryption client library. Encr and Decr happens at clients place.
Encryption in transit is also called SSl/TLS.
S3 security - user-based - IAM policies resource-based - Bucket Policies, Object ACL, Bucket ACL
S3 Bucket Policies - used for providing access, force encryption, cross-account access. Written in JSON it can have Resources (buckets & objects),Actions (set of API to allow/deny), Principal (user/account the policy applies to)
S3 access logs can be stored in another bucket.
Can be integrated with Cloud trail for API calls. Supports VPC endpoints.
S3 websites - allows static websites. Url is **<bucket-name>.s3-website-<AWS region>.amazonaws.com** . If you get 403 error then check bucket policies for public read.
S3 CORS - If you request data from another S3 bucket you need to enable Cross Origin Resource Sharing. This allows to limit the number of websites that can request your files in S3 (and limit costs).**Access-Control-Allow-Origin : <domain>**
S3 Consistency Model - Read after write consistency for PUTS (you can read imm after write) , Eventually consistent for PUTS and DELETES
S3 can send notifications on changes to SQS,SNS & Lambda. S3 has cross-region replication feature.

Amazon S3 (cont)

S3 performance - upto 3500 RPS for PUTS & 5500 RPS for GETS. Faster upload of >100MB use multi-part. Must use multi part if >5GB. Use cloudfront to cache S3 reads. S3 transfer acceleration-(uses edge location) - just need to change the endpoint u write to. If using KMS encryption then AWS limits applies.
S3 & Glacier Select - Glacier is for long term archival.Its another 'tier' within S3.If you retrieve from glacier you might need only a subset. Otherwise costs might be high. So SQL select queries can be used. No subqueries or joins are supported. Works in CSV,JSON or parquet format.

CICD (Continuous Integration Continuous Delivery)

CICD - automating deployment with additional safety using CodeCommit,CodeBuild,CodePipeline,CodeDeploy
CodeCommit - storing code. Version Control.Colloborate with other developers. Code is backed up. Fully viewable and auditable. Provides private repositories. No size limit on repositories. Secure (encryption,ACL).Integrated with Jenkins/CodeBuild and other CI tools.
CodeBuild - building and testing code
CodePipeline - automating pipeline from code to EB
CodeDeploy - deploying code to EC2 fleets.
CI - Find bugs early and fix, Deliver faster as the code is tested,Deploy often
CD - Ensure that sw can be delivered reliably, Ensures deployments happen often and quick
Code Commit security - Authentication using SSH,HTTPS using AWS cli or https cred,MFA can be enabled.IAM to repositories.ENcryption at rest using KMS and in flight through https/ssh.
Code commit notifications - SNS,cloud watch events or lambda. To automatically trigger a code analysis when code is commited and to check if there are no secrets there use AWS SNS/Lambda integ in code commit.
Code Pipeline - Visual workflow, source - S3/GIT/Code commit. Made of stages. Each stage can have seq or parallel actions. Manual approval can be define at any stage.
CodePipeline Artifacts - Each stage in pipeline can produce artifacts which are stored in S3.
Pipeline troubleshooting - Stage changes happen in CloudWatch events. Events can create SNS notifications. If stage fails it stops and info can be seen in console. CloudTrail can be used to audit API calls. If pipeline cant perform actions make sure IAM service role has enough permissions. (IAM policies).

CICD (Continuous Integration Continuous Delivery) (cont)

Code Build - Continous scaling,Fully managed, Leverages docker. Secure as it integrates KMS for secure build artifacts,IAM for build permissions, VPC for nw security, CloudTrail for API calls logging.
Code build - Source code from code commit/S3/github/Codepipeline. Build inst can be defined in code (buildspec.yaml). Output logs to S3 & Cloudwatch logs. Cloudwatch events to trigger notifications if build fails. Cloudwatch alarms to notify if threshold failures. Can reproduce locally to troubleshoot using code build agent.
Buildspec - buildspec.yaml must be at the root of the code. Environment variables - Secure secrets using SSM parameter store. Phases - Install (dependencies), Prebuild, Build ,postbuild (output). Cache dependencies to S3 to help speed up build.
CodeDeploy - Each EC2 instance must run CodeDeploy agent. Agent continously polls CD for work. CD send appsec.yaml. App is pulled from github or S3. EC2 will run deployment instructions.
Code Deploy components - Application (must have unique name), platform (EC2/on-prem,Lanbda),dep group (group of ec2 inst)
App Sec - File section (how to copy from git/S3) , Hooks (ApplicationStop,DownloadBundle,BeforeInstall,AfterInstall,Applicationstart,Validate service)
Deployment Config - Configs : one a time(1 inst a time), Half at time,all at once, Custom
Deploy targets - set of EC2 instances with tags, ASG,Mix of ASG/Tags, Customization in scripts using DEPLOYMENT_GROUP_NAME env variable.
Codestar - Integrates solution that regroups Github, CodeCommit, Codebuild, Codepipeline, Codedeploy, CloudFormation & Cloudwatch.
Codestar helps create projects for Lambda,EC2,Beanstalk.

CloudFormation

CloudFormation - Infra as code.

Benefits

Infra as code - No manual creation of resources.Code is version controlled. Changes to infra are reviewed in code.
Cost - you can see how much all resources in stack costs. You can estimate costs of your resources using template. Savings strategy : You can automate deletion of template at 5pm and creation at 8PM.
Productivity

CloudFormation (cont)

Separation of concern - VPC stack,Network Stack, App stacks
Reusable - use templates on web
How CF works - Templates have to be uploaded in S3.To update a template re-upload a new version. Stacks are identified by name. Deleting stack deletes all artifacts.
Deploying CF - Manually : Edit templates in CF designer, Using console to input parameters.
Automated : Editing templates in YAML File, AWS CLI to deploy templates
CF Building blocks - Templates Components : Resources,Parameters,Mappings,Outputs,Conditionals,Metadata.
Resources - AWS resources declared in template. Resource type identifier **AWS:: aws-product-name::data-type-name** e.g. AWS::EC2::instance
Parameters -dynamic inputs. If CF resource config is going to change then use parameters. To reference a parameter use **Fn::Ref** function. In YAML it is !Ref.
Pseudo Parameters - AWS provides these params by default. Some are **AWS::AccountId , AWS::Region , AWS::StackName**
Mappings - static variables, used to differentiate environments(dev,prod) or Regions (us-east-1,eu-central-1). To reference mappings use **Fn::FindInMap**. In YAML it is !FindInMap[Mapname, Toplevelkey, secondlevelkey].
Outputs - References to what has been created. Can export to other stack and used for cross stack reference using **Fn::ImportValue**. Cant delete the stack until the underlying cross-ref stack is also deleted.
Conditions - list of condition to perform resource creation
CF intrinsic functions - Ref , Fn::GetAtt, Fn::FindInMap, Fn::ImportValue, Fn::Join, Fn::Sub (substitute),Condition functions
YAML & JSON are used to create templates.
CF Rollbacks - When stack creation fails - default it will roll back (gets deleted). Optional is to stop rollback and troubleshoot what happened. When stack update fails - The stack will rollback to previous known working state. You can see the logs what happened.

Other Services

CloudFront - Works with Content Delivery network. Improves read performance.Cached at edge.Prevents network attacks. Provides encryption.
Simple Email Service (SES) - integrates with SNS,Lambda,S3
Step Functions - Server less workflow to orchestrate Lambda functions,JSON state machine, max execution of 1 yr, integrate with EC2/ECS/API gateway
Simple Workflow Service - not server less, Human approval possible, if you need external intervention or child processes then go with this otherwise Step functions.
Other Dbs - RDS, Dynamo, Inmemory(memcached/Redis),Neptune(graph),OLAP(Redshift),DMS
ACM - Certificate manager. Loads SSL certificates to LBs,API gateway,Cloud front distributions

CLI , SDK , IAM Roles & Policies

Lambda

API Gateway

Serverless App Model (SAM)

AWS Security & Encryption : KMS, Parameter store