AWS Cloud Engineer Interview Questions & Answers (2025)

1) How do you design a highly available application architecture using AWS?

- **Principles**
  - Remove single points of failure
  - Fault isolation and automatic failover
  - Automate recovery and testing (DR drills)
- **Compute & Traffic**
  - Deploy app instances across **multiple AZs** (2+)
  - Use **ALB/NLB** in front of stateless services
  - Place instances in **Auto Scaling Groups (ASG)** with min/desired/max
- **Data & Storage**
  - Use **RDS Multi-AZ** or **Aurora** (with replicas/global DB for cross-region)
  - Store static assets in **S3** (S3 + CloudFront for CDN & origin failover)
  - Use managed services with built-in replication (DynamoDB Global Tables, ElastiCache replication)
- **DNS & Routing**
  - Use **Route53 health checks** and routing policies (failover/weighted/latency)
- **Observability & Automation**
  - **CloudWatch** metrics & alarms, centralized logging, cross-account logs
  - CI/CD with blue/green or canary deployments for zero-downtime
- **Trade-offs**
  - Multi-region = higher cost & complexity, but lower RTO/RPO

---

2) Explain the difference between NLB, ALB, and CLB. When will you use each?

- **ALB (Application Load Balancer)**
  - Layer 7 (HTTP/HTTPS)
  - Host/path-based routing, WebSockets, HTTP/2, authentication (OIDC)

    - Use for microservices, content-based routing, WAF
- **NLB (Network Load Balancer)**
    - Layer 4 (TCP/UDP/TLS)
    - Ultra-low latency, millions of connections, preserves source IP, supports Elastic IPs
    - Use for non-HTTP, high-throughput TCP workloads, or static IP requirements
- **CLB (Classic Load Balancer)**
    - Legacy (L4 & L7 older model)
    - Use only for legacy apps that require it
- **Patterns**
    - Combine NLB + ALB when needed (e.g., NLB for TLS + ALB for L7)

---

3) What is the purpose of VPC Peering vs Transit Gateway?

- **VPC Peering**
    - One-to-one VPC connection
    - Low-latency, simple routing
    - No transitive routing (A↔B and B↔C does not imply A↔C)
    - Good for a small number of VPCs / bilateral trust
- **Transit Gateway (TGW)**
    - Hub-and-spoke for many VPCs and on-prem networks
    - Supports transitive routing, route propagation, central inspection
    - Scales to 1000s of VPCs
    - Higher cost (attachments & hourly charges) but simpler management at scale
- **When to use**
    - Peering = few VPCs, simple trust
    - TGW = multi-account, multi-VPC at scale, centralized egress/monitoring

---

4) How do you secure S3 buckets? Best practices

- **Access**
    - Block public access at account & bucket level (default deny)
    - Use least-privilege **IAM** policies; prefer roles over long-term keys
    - Enforce **Bucket Policies** (restrict by VPC endpoints, IP, principals)
    - Avoid ACLs; use Object Ownership (bucket owner enforced)
- **Encryption**
    - SSE-S3 or SSE-KMS (prefer KMS for key control & rotation)
    - Client-side encryption where needed

- **Data protection**
  - Enable **versioning** + **MFA Delete** (if operationally acceptable)
  - Use **Lifecycle policies** to archive/delete old versions
- **Network**
  - Use **Gateway VPC Endpoints** for S3 to keep traffic internal
- **Monitoring & Auditing**
  - Enable **S3 access logs**, **CloudTrail** data events, send logs to separate bucket
  - Use AWS Config rules & IAM Access Analyzer
- **Operational**
  - Tag, classify data, and apply stricter controls to sensitive objects

---

5) Explain how Auto Scaling works. What policies have you used?

- **Core components**
  - Launch template/configuration, Auto Scaling Group (ASG), health checks, scaling policies
- **Policy types**
  - **Target Tracking**: maintain metric target (e.g., CPU = 50%); simplest & recommended
  - **Step Scaling**: scale in steps based on how far metrics breach thresholds
  - **Scheduled Scaling**: scale at known times (business hours)
  - **Simple Scaling**: older pattern (alarm triggers action + cooldown)
- **Other features**
  - Lifecycle hooks (warm-up, draining)
  - Mixed instance policies (On-Demand + Spot)
- **Best practice**
  - Combine target tracking (baseline) + step scaling (spikes) + scheduled scaling for predictable loads

---

6) Difference between EC2 On-Demand, Reserved, and Spot Instances

- **On-Demand**
  - Pay-as-you-go, highest flexibility
  - Use for unpredictable workloads, dev/test
- **Reserved Instances / Savings Plans**
  - 1–3 year commitment, significant discounts
  - Use for steady-state predictable workloads

- Savings Plans offer more flexibility across instance families
- **Spot Instances**
  - Up to ~90% discount, interruptible (2-minute warning)
  - Use for fault-tolerant batch jobs, stateless processing
- **Pattern**
  - Mix RIs/SavingsPlans for baseline, On-Demand for bursts, Spot for cheap capacity

---

7) How do you create a Multi-AZ setup for RDS?

- **At creation**
  - Enable **Multi-AZ** option — AWS creates synchronous standby in another AZ
- **Post-creation**
  - Modify the instance to enable Multi-AZ
- **Read scale**
  - Create **read replicas** (optionally cross-region)
- **Aurora**
  - Use Aurora replicas + **Aurora Global Database** for multi-region reads & fast recovery
- **Operational**
  - Ensure subnet groups include multiple AZs, enable automated backups/snapshots, and setup connection pooling (RDS Proxy) for failover handling

---

8) Parameter Store vs Secrets Manager

- **Systems Manager Parameter Store**
  - Good for configs and simple secrets (secure string via KMS)
  - Free tier for standard parameters
  - Hierarchical paths & versioning
- **Secrets Manager**
  - Built for secrets (DB passwords, API keys)
  - Supports **automatic rotation** (Lambda-based)
  - Paid service with rotation & lifecycle features
- **Guidance**
  - Use Parameter Store for non-rotating config or small secrets
  - Use Secrets Manager for rotating credentials and more advanced secret workflows

---

9) Flow: Internet → VPC → Private Subnet

- **DNS**
    - Client -> Route53 (or external DNS) -> resolves to ALB / CloudFront static IPs
- **Ingress**
    - Traffic reaches **Internet Gateway (IGW)** -> public subnet containing **ALB/NLB**
- **LB to private**
    - ALB forwards to targets (EC2/ECS) in **private subnets**
- **Outbound from private**
    - Private instances use **NAT Gateway** in public subnet for outbound internet
- **Controls**
    - Route tables, Security Groups (instance-level, stateful), NACLs (subnet-level, stateless)
- **Observability**
    - VPC Flow Logs, ALB logs, CloudWatch

---

10) IAM: role vs policy vs user

- **IAM User**
    - Long-term credentials for individuals (avoid for applications)
- **IAM Role**
    - Temporary credentials; assumed by services (EC2, Lambda) or cross-account principals
- **IAM Policy**
    - JSON document that **allows/denies** actions on resources
- **Best practices**
    - Least privilege, use roles & temporary creds, identity federation for SSO

---

11) Use of Lifecycle Policies in S3

- **Automate data lifecycle**
    - **Transition** objects to cheaper classes (IA, Glacier)
    - **Expire**/delete objects or old versions
- **Use cases**
    - Cost optimization for large datasets, compliance retention, automated archiving
- **Combine**
    - With versioning and inventory for visibility

---

12) Troubleshoot performance issues in EC2

- **Metrics & baseline**

    - Check CloudWatch (CPU, custom memory, disk IO, network)
    - Install CloudWatch agent for OS-level metrics
- **Live inspection**
    - Use SSM Session Manager to run `top`, `iotop`, `iostat`, `df -h`
- **I/O & disk**
    - Check EBS type (gp2/gp3/io1), IOPS/throughput limits, burst credits
- **Network**
    - Check NIC, ENI metrics, VPC Flow Logs, `ss -tna`
- **Application**
    - Review logs, slow queries, GC pauses, thread contention
- **Mitigation**
    - Right-size, scale out (ASG), add caching (ElastiCache), offload heavy jobs to queues

---

13) Explain Blue/Green Deployment on AWS

- **Concept**
    - Blue = current production; Green = new version
- **Steps**
    - Deploy to Green, test thoroughly, switch traffic (Route53/ALB weight shift)
- **AWS tools**
    - CodeDeploy (supports blue/green), ECS blue/green via CodeDeploy, ALB target-group swaps
- **DB considerations**
    - Backward-compatible migrations, dual-write strategies, or feature flags for gradual migration
- **Rollback**
    - Instant rollback by switching back to Blue

---

14) What is CloudFormation and how have you used it?

- **Definition**
    - AWS-native IaC (YAML/JSON) for declarative resource provisioning
- **Features**
    - Dependency management, rollback, nested stacks, drift detection, StackSets
- **Usage patterns**
    - Store templates in source control, use change sets, parameterize stacks, use StackSets for multi-account
- **When to use**
    - Use CloudFormation for deep AWS integration; Terraform if multi-cloud or preferred HCL

---

15) Steps to implement Disaster Recovery (DR) strategy in AWS

- **Planning**
  - Define RTO/RPO per application
- **Patterns**
  - Backup & Restore, Pilot Light, Warm Standby, Active-Active
- **Data replication**
  - S3 CRR, RDS cross-region replicas, DynamoDB global tables
- **Automation**
  - IaC templates for quick reprovisioning (CloudFormation/Terraform)
- **DNS & failover**
  - Route53 health checks + failover policies
- **Testing**
  - Regular DR drills & runbooks
- **Security**
  - Cross-region backups encrypted and IAM prepared for recovery

---

16) Difference between Stateful and Stateless Architecture

- **Stateless**
  - No session stored on server; easier to scale horizontally
  - Use JWT, cookies, or external store (Redis) for sessions
- **Stateful**
  - Server stores session/state; requires session replication or sticky sessions
  - Harder to scale; suitable when session locality is important
- **Cloud-native preference**
  - Prefer stateless microservices + external state stores

---

17) Security Groups vs NACLs

- **Security Groups**
  - Instance-level, stateful, allow rules only, return traffic allowed automatically
  - Use for micro-segmentation
- **NACLs**

    - Subnet-level, stateless, support allow/deny, rules evaluated by number
    - Use for coarse-grained subnet-level enforcement / extra layer of defense
- **Best practice**
    - Use Security Groups primarily; use NACLs only for additional boundary rules

---

18) Role of CloudWatch, CloudTrail, GuardDuty

- **CloudWatch**
    - Monitoring metrics, logs, dashboards, alerts, autoscaling triggers
- **CloudTrail**
    - Audit trail for API calls (who/what/when); crucial for investigations & compliance
- **GuardDuty**
    - Managed threat detection using VPC Flow Logs, CloudTrail, DNS logs; alerts suspicious activity
- **Integration**
    - Aggregate findings into Security Hub for centralized response

---

19) Explain PrivateLink, Endpoint, NAT Gateway

- **PrivateLink (Interface Endpoint)**
    - Private connectivity to AWS services or SaaS via ENIs in your VPC
    - Keeps traffic inside AWS network; good for SaaS access
- **Gateway Endpoint**
    - For S3/DynamoDB; route-based, keeps traffic internal without IGW
- **NAT Gateway**
    - Enables private subnet outbound internet access while blocking inbound internet-initiated traffic
    - Highly-available per-AZ (use one NAT per AZ for HA)

---

20) How do you optimize AWS cost for large workloads?

- **Right-size**
    - Monitor & downsize idle resources
- **Commitments**
    - Reserved Instances / Savings Plans for baseline
- **Spot**
    - Use Spot for fault-tolerant workloads

- **Autoscaling**
  - Scale in/out to match demand
- **Storage**
  - S3 lifecycle & intelligent-tiering; compression
- **Design**
  - Use serverless where cost-effective; caching & CDNs; reduce cross-region egress
- **Governance**
  - Tags, Cost Explorer, Budgets, anomaly detection

---

21) Latency issues across regions — Global Accelerator vs Route53

- **Assess**
  - Gather metrics and RUM to identify client distribution & latency patterns
- **Route53**
  - Use latency-based or geo routing and CloudFront for HTTP content
- **Global Accelerator**
  - Static IPs, AWS global network (lower jitter), better for TCP/UDP & non-HTTP
- **Design**
  - Multi-region deployments + global routing + CDN + replicated data stores (DynamoDB global tables)

---

22) Migrate monolith → microservices on AWS

- **Plan**
  - Domain-driven decomposition / strangler pattern (incremental extraction)
- **Runtime**
  - EKS for full k8s control; ECS/Fargate for simpler container ops; Lambda for event-driven
- **API & messaging**
  - API Gateway for external APIs; EventBridge / SNS / SQS / Kinesis for async events
- **Data**
  - Move to per-service data stores or use dual-write patterns during transition
- **CI/CD**
  - Build container images, ECR, blue/green/canary deployments
- **Observability**
  - Tracing (X-Ray), metrics (Prometheus/CloudWatch), logging (CloudWatch/ELK)
- **Security**
  - VPC segmentation, private service endpoints, IAM roles per service

23) S3 bill increased suddenly — investigate & reduce cost

- **Investigate**
  - Use S3 Storage Lens, Cost Explorer, S3 Inventory (Athena) to identify buckets/objects
- **Common causes**
  - Accidental backups, version accumulation, logging growth, multipart uploads
- **Remediation**
  - Implement/adjust lifecycle rules, delete unused versions, enable intelligent-tiering, compress objects, fix scripts creating duplicates
- **Prevention**
  - Alerts for sudden growth, cost allocation tags, and regular audits

---

24) Zero downtime deployment: CodeDeploy, ECS, Blue/Green

- **Use Blue/Green**
  - Deploy to green env, test, shift traffic (weighted/ALB/Route53)
- **Tools**
  - CodeDeploy for EC2/Lambda; ECS integration for task set switching
- **DB migrations**
  - Make migrations backward-compatible; use expand-contract pattern
- **Feature flags**
  - Toggle new features without immediate DB changes
- **Validation**
  - Pre- and post-traffic hooks & automated smoke tests

---

25) Alert: EC2 CPU 90%+ for 10 minutes — steps to fix

- **Immediate**
  - Check CloudWatch scope (single vs fleet)
  - Use SSM Session Manager to inspect (`top` , `ps`)
- **Short-term mitigation**
  - Scale out via ASG, restart problematic services, terminate/recreate unhealthy instance
- **Root cause**

    - Investigate logs, recent deploys, runaway processes, memory leaks
- **Long-term**
    - Add autoscaling, caching, optimize app & queries, add runbook & alarms

---

26) Design multi-region failover strategy

- **Options**
    - Active-Passive (warm standby), Active-Active (multi-region writes)
- **Data replication**
    - Use cross-region read replicas, DynamoDB global tables, S3 CRR
- **DNS failover**
    - Route53 health checks + failover routing
- **IaC**
    - Maintain templates for both regions & automate failover runbooks
- **Testing**
    - Regular DR drills and validation

---

27) Secure public API with WAF, API Gateway, Cognito, Shield

- **Fronting**
    - API Gateway for throttling, validation, and auth
- **Authentication**
    - Cognito User Pools (OIDC/JWT) for user auth; integrate with API Gateway
- **Protection**
    - WAF for rule-based filtering (SQLi, XSS, rate limits)
    - Shield Standard (auto) or Shield Advanced for enterprise DDoS protection
- **Best practices**
    - TLS everywhere, logging, quotas, usage plans, secrets in Secrets Manager, CORS restrictions

---

28) RDS hitting connection limits — scale & optimize

- **Connection pooling**
    - Use RDS Proxy or PgBouncer/ProxySQL to pool connections
- **Read scaling**

    - Offload reads to read replicas
- **Scale**
    - Vertical scale (bigger instance) or optimized instance class
- **Query optimization**
    - Indexes, slow-query analysis, optimize transactions
- **Cache**
    - Use ElastiCache to reduce DB hits
- **Batching**
    - Reduce connection churn, use async processing

---

29) Process 1M messages/hour — use SQS/SNS/Kinesis

- **Throughput**
    - ~278 msgs/sec — moderate throughput
- **Service choice**
    - **SQS Standard**: high throughput, at-least-once, scale consumers horizontally
    - **SQS FIFO**: for ordering & exactly-once semantics (throughput limits)
    - **SNS + SQS**: fan-out to multiple consumers
    - **Kinesis**: ordered stream, replayability, real-time analytics; scale via shards
- **Design**
    - Producers -> SNS (if fan-out) -> SQS -> autoscaling consumers (ECS/Lambda)
    - Use batching, DLQs, monitor queue depth & consumer scaling

---

30) IaC: CloudFormation vs Terraform — which to choose?

- **CloudFormation**
    - AWS native, deep service integrations, drift detection, StackSets
    - Use when AWS-only and tight native features are needed
- **Terraform**
    - Multi-cloud, modular, HCL, large module ecosystem, flexible state management
    - Use when multi-cloud or cross-provider infra is required
- **Decision**
    - Choose based on team skills, portability needs, and governance
    - Hybrid approach possible (Terraform for multi-cloud, CloudFormation for AWS-native features)

---

31) Rotate secrets automatically with Secrets Manager

- **Use Secrets Manager**
  - Store secrets and enable automatic rotation
- **Rotation flow**
  - Secrets Manager invokes Lambda that runs rotate workflow (create/set/test/finish)
- **Implementation**
  - Use built-in rotation templates for RDS or custom Lambda for other services
  - Grant Lambda KMS & service-specific permissions
  - Update apps to fetch secrets via Secrets Manager and cache them minimally

---

32) EC2 instances under DDoS attack — mitigation

- **Immediate**
  - WAF with rate-based rules, IP blocking, and security group updates
- **AWS protections**
  - Shield Standard (auto) or Shield Advanced for enterprise support
  - CloudFront + WAF to absorb at edge; offload traffic & caching
- **Scaling**
  - Use ASG to absorb burst (with caution — costs)
- **Network**
  - Use NACLs for coarse IP deny rules, analyze VPC Flow Logs
- **Post-mortem**
  - Update WAF rules, contact AWS DDoS Response Team (if Shield Advanced), and consider scrubbing services

---

33) Bonus — Useful troubleshooting commands & snippets (bullets)

- HTTP requests & headers:
  - `curl -I https://example.com`
  - `curl -X POST -d @payload.json https://example.com/api`
- DNS:
  - `dig +short example.com`
  - `dig example.com ANY`
- Routing:
  - `traceroute example.com` (Linux/macOS)

- `tracert example.com` (Windows)

- Networking / sockets:
  - `ss -tna | grep ESTAB`
  - `netstat -anp` (or `ss`)
- System metrics:
  - `top` , `htop` , `vmstat` , `iostat` , `iotop` , `free -m` , `df -h`
- AWS CLI:
  - `aws s3 ls s3://bucket --summarize --human-readable`
  - `aws cloudwatch get-metric-statistics --namespace AWS/EC2 ...`
  - `aws rds describe-db-instances`
- Security scanning & probing (with permission):
  - `nmap -sT -p 1-1000 host`
- SSM (no open SSH):
  - `aws ssm start-session --target <instance-id>`
