Cloud Services — AWS (25 Questions)

Q1: Your production web app behind an ALB throws intermittent 502s during traffic spikes. Debug and stabilize without resizing instances.

Answer:
Start with ALB target health and response codes. Correlate ALB TargetResponseTime and HTTPCode_Target_5XX with EC2 CPU, connection counts, and app thread pool saturation. 502s usually mean backend closed or timed out; raise upstream timeouts (Nginx → app), bump ulimit/file-descriptors, tune keep-alive, and enable connection reuse. Offload TLS at ALB; ensure SG from ALB to instances is tight. Add an Auto Scaling policy on meaningful app metrics (e.g., queue depth), not CPU alone.

Sample Points:

- Confirm ALB health checks + backend port.
- Tune upstream/proxy timeouts and keep-alive.
- Scale on app backlog; cap max in-flight requests.

Example Code:

```
aws elbv2 describe-target-health --target-group-arn arn:...
aws cloudwatch get-metric-statistics --namespace AWS/ApplicationELB\
--metric-name TargetResponseTime --dimensions
Name=TargetGroup,Value=tg/...
```

Q2: Static site on S3 + CloudFront shows stale JS after a hotfix; users still see the buggy version. Push a safe, cost-aware fix.

Answer:
Use filename versioning (app.v2025-08-10.js) to avoid repeated invalidations. For this hotfix, do a targeted CloudFront invalidation for the changed assets; set short TTL only for

HTML (long TTL for hashed assets). Ensure `Cache-Control: no-cache` on HTML, immutable on versioned assets.

Sample Points:

- Versioned assets beat frequent invalidations.
- HTML short TTL; assets long TTL + immutable.
- Keep invalidation paths scoped.

Example Code:

```
aws cloudfront create-invalidation --distribution-id E123ABC --paths\
"/index.html" "/assets/app.v2025-08-10.js"
```

Q3: Lambda to RDS Postgres hits connection limits at peak. Fix without upscaling DB.

Answer:
Introduce **RDS Proxy** with IAM auth so concurrent Lambdas multiplex to fewer backend connections. Set sensible max_connections_percent, reuse connections, and tune Lambda concurrency (reserved concurrency) to cap surges. Rotate creds in Secrets Manager; block public access to DB.

Sample Points:

- RDS Proxy pools connections.
- Constrain Lambda concurrency.
- Use IAM auth + Secrets Manager.

Example Code:

```
aws rds create-db-proxy --db-proxy-name prod-proxy --engine-family
POSTGRESQL \
  --auth
'[{"IAMAuth":"REQUIRED","SecretArn":"arn:aws:secretsmanager:..."}]' \
--role-arn arn:aws:iam::123:role/RDSProxyRole
```

Q4: Private-subnet EC2s can’t reach the internet for OS updates, but egress must be restricted by domain.

Answer:
Use a **NATGateway** plus **VPCegress-onlycontrols** via **Route 53 Resolver + DNS Firewall** or proxy via **AWS Network Firewall** with FQDN-based rules. For strict domain egress, place a **Squid**/forward proxy in public subnet and point instances via http_proxy/https_proxy.
Endpoint policies for S3/ECR reduce public egress.
Sample Points:

- NAT for generic egress; endpoints for AWS APIs.
- DNS Firewall/Network Firewall to constrain FQDNs.
- Artifact pulls via VPC endpoints (S3/ECR).

Example Code:

```
aws ec2 create-route --route-table-id rtb-... \
  --destination-cidr-block 0.0.0.0/0 --nat-gateway-id nat-...
```

Q5: Multi-account org: a dev team must start/stop only EC2s tagged App=Billing in their account. Enforce via IAM.

Answer:
Use a permissions boundary or SCP for guardrails and attach an IAM policy with`Condition` on ec2:ResourceTag/App = Billing. Require the same tag on the instance **and** enforce `ec2:CreateTags on launch.`
Sample Points:

- Tag-based ABAC on actions.
- Combine IAM + SCP for least privilege.
- Enforce tagging at create-time.

Example Code:

```
{

"Effect":"Allow","Action":["ec2:StartInstances","ec2:StopInstances"],

"Resource":"*","Condition":{"StringEquals":{"ec2:ResourceTag/App":"Bil
ling"}}
}
```

Q6: EKS pods need S3 read and DynamoDB write with zero static creds. Design and wire it.

Answer:
Enable EKS OIDC provider, create **IRSA** roles with least-privilege policies, and annotate service accounts per workload. Validate `aws sts get-caller-identity` inside pod.
Separate roles per namespace to limit blast radius.

Sample Points:

- OIDC + IRSA > node IAM roles.
- One role per SA for least privilege.
- Rotate by redeploying SA annotations.

Example Code:

```
eksctl utils associate-iam-oidc-provider --cluster prod --approve
eksctl create iamserviceaccount --cluster prod --namespace orders\
--name ddb-writer --attach-policy-arn
arn:aws:iam::123:policy/DdbWrite --approve
```

Q7: Kinesis consumer lag spikes; records arrive late to downstream Lambda. Triage and fix throughput without overpaying.

Answer:
Increase shard count (or enable on-demand), raise Lambda batch size/parallelization factor, and enable enhanced fan-out if many consumers. Monitor iterator age. Prefer on-demand for

bursty traffic; scale down shards post-peak if using provisioned.
Sample Points:

- `Watch GetRecords.IteratorAge.`
- Batch/parallelization tuning first.
- Shard scaling vs on-demand cost trade-off.

Example Code:

```
aws kinesis update-shard-count --stream-name orders
--target-shard-count 8 --scaling-type UNIFORM_SCALING
```

Q8: API Gateway + Lambda returns 429 during promotions. Smooth it out without lifting limits org-wide.

Answer:
Use **usage plans** and **API keys** to shape traffic per client, enable **API cache** for idempotent reads, and offload async work to **SQS**. Add **reserved concurrency** per Lambda to isolate tenants.
Sample Points:

- Usage plans for fair sharing.
- Cache hot GETs at API Gateway.
- Queue write-heavy paths.

Example Code:

```
aws apigateway create-usage-plan --name prod-plan --throttle
burstLimit=200,rateLimit=100
```

Q9: RDS MySQL monthly reporting kills OLTP. Keep reports fast without hurting prod.

Answer:
Create a **read replica** (or Aurora reader), offload reporting there, and schedule data exports to

S3 + Athena when feasible. Tune long-running queries with proper indexes. If locking is the issue, consider **Aurora Serverless v2** for elastic readers.
Sample Points:

- Separate OLTP and reporting traffic.
- Tune queries before scaling.
- Consider Athena for ad-hoc analytics.
  Example Code:

```
aws rds create-db-instance-read-replica
--source-db-instance-identifier prod-db --db-instance-identifier
prod-db-ro
```

Q10: DynamoDB hot partition due to skewed `tenant_id`. Stabilize latency.

Answer:
Adopt **adaptive partitioning**: use a composite key with a **sharded** partition key (e.g., tenant_id#shard_0..N) and distribute writes with a client-side hashing strategy. Enable **DAX** only if read-latency is the main issue; use **on-demand** capacity during spikes.
Sample Points:

- Prevent hot partitions via key sharding.
- On-demand for unpredictable traffic.
- DAX helps read latency, not writes.
  Example Code (write key):

```
pk = f"{tenant_id}#{hash(order_id)%8}"
table.put_item(Item={"pk": pk, "sk": order_id, ...})
```

Q11: Cross-region, active-active API with Route 53. Ensure session affinity and graceful failover for a stateful app.

Answer:
Store session/state in **DynamoDB Global Tables** or **ElastiCache Global Datastore**; front with

Route 53 latency-based routing + health checks. Use sticky sessions only at the regional ALB; primary solution is **statelessfrontends** . Enable circuit breakers in clients for brownouts.
Sample Points:

- Externalize state for multi-region.
- Latency routing + health checks.
- Prefer stateless; sticky is a crutch.

Example Code:

```
aws route53 change-resource-record-sets --hosted-zone-id Z123
--change-batch file://latency.json
```

Q12: S3 bucket hosts PII exports. Data must be readable only from two VPCs and by a specific AWS account's role.

Answer:
Use **S3 bucketpolicy** with aws:PrincipalOrgID/Principal restriction, **VPC endpoint policy** requiring aws:sourceVpce, and enforce **SSE-KMS** with a key policy granting only the role and bucket access. Block public access at account+bucket level.
Sample Points:

- Combine bucket + endpoint policies.
- Tight KMS key policy; no wildcard principals.
- Enable Access Analyzer to validate.

Example Code (bucket policy excerpt):

```
{"Condition":{"StringEquals":{"aws:sourceVpce":"vpce-abc"}}}
```

Q13: You must rotate ECR images across accounts with provenance and immutability.

Answer:
Enable **ECR image scanning**, **image tag immutability** , push with **Sigstore cosign** or ECR pull-through cache. Replicate with **ECR replication rules** to target accounts/regions. Enforce

deployment on **signed** digests only via CI and admission controllers (EKS).
Sample Points:

- Immutable tags prevent “latest” drift.
- Replication rules automate fan-out.
- Verify signatures at deploy time.

Example Code:

```
aws ecr put-image-tag-mutability --repository-name app
--image-tag-mutability IMMUTABLE
aws ecr put-replication-configuration --replication-configuration
file://replication.json
```

Q14: SQS queue backlog grows; consumers on ECS Fargate occasionally crash on poison messages.

Answer:
Use **DLQ** with redrive policy; set **visibilitytimeout** > max processing time. Implement idempotency and retries with backoff. Scale ECS service by **ApproximateNumberOfMessagesVisible** and cap with service autoscaling. Add message validation before processing.
Sample Points:

- DLQ protects consumers.
- Visibility timeout must exceed processing.
- Scale consumers on queue depth.

Example Code:

```
aws sqs set-queue-attributes --queue-url $Q \
--attributes
RedrivePolicy='{"deadLetterTargetArn":"arn:...:dlq","maxReceiveCount":
"5"}'
```

Q15: Event-driven ETL: S3 → Lambda → Glue. Random timeouts on big CSVs. Make it resilient.

Answer:

Use **S3 event** to enqueue to **SQS**, trigger a **Lambda** that launches **Gluejob** asynchronously; increase Lambda timeout or split big files with **S3 Multipart + manifest**. For very large transforms, move logic into Glue (Spark) instead of Lambda. Add **idempotency token** to avoid duplicate processing.

Sample Points:

- Decouple with SQS.
- Use Glue for heavy transforms.
- Idempotency + retries.

Example Code:

```
aws glue start-job-run --job-name etl
--arguments='--s3key=s3://bucket/key.csv'
```

Q16: You need fine-grained access for CI runners to deploy to ECS, but prohibit console access.

Answer:

Create a **role for CI** with **OIDC federation** (GitHub/GitLab) and scoped policies: `ecs:UpdateService`, `ecr:GetAuthorizationToken`, `logs:CreateLogStream`. Deny `aws:ViaAWSService`=false to block console usage, and condition on repo/branch claims in OIDC token.

SamplePoints:

- Web identity federation for CI.
- Least-privilege ECS/ECR/Logs actions.
- Explicit deny for console paths.

Example Code:

```
"Condition":{"StringEquals":{"token.actions.githubusercontent.com:sub"
:"repo:org/app:ref:refs/heads/main"}}
```

Q17: Blue/Green for ECS with zero-downtime + HTTP checks + gradual traffic shift.

Answer:
Use **CodeDeploy** with an **ALB** target group pair. Configure canary or linear traffic shifting and pre/post hooks for smoke tests. Roll back on health check failures automatically.
SamplePoints:

- Two target groups, one service.
- Canary/linear shift patterns.
- Hooks for validation + rollback.

ExampleCode(AppSpecexcerpt):

```
{"Resources":[{"TargetService":{"Type":"AWS::ECS::Service","Properties
":{"TaskDefinition":"<td>"}}}]}
```

Q18: CloudWatch costs are growing due to verbose app logs. Keep observability but reduce spend.

Answer:

Add **log filters** at agent (Fluent Bit) to drop debug in prod, route **metrics** with EMF instead of logs, enable **log retention** (14–30 days) + **subscription filters** to S3 for archival, and compress in S3 IA. Use **metric filters** for alerts instead of scanning logs.
SamplePoints:

- Reduce at source; don't ship noise.
- Short retention, archive to S3.
- Metrics > logs for steady alerts.

Example Code:

```
aws logs put-retention-policy --log-group-name /app/prod
--retention-in-days 14
```

Q19: You must expose an internal ALB via API Gateway for standardized auth and throttling.

Answer:

Deploy **API Gateway HTTP API** with **VPC Link** to the ALB target group. Attach **JWT authorizer**/Cognito, apply **throttle** per route, and restrict ALB SG to only the VPC Link ENIs. Keep idle timeouts aligned.

Sample Points:

- HTTP API + VPC Link to ALB.
- Centralized auth + throttling.
- Lock down SG to VPC Link ENIs.

Example Code:

```
aws apigatewayv2 create-vpc-link --name prod-link --subnet-ids
subnet-...
```

Q20: A single AZ outage took down your stateful workload on EBS. Add HA with minimal refactor.

Answer:

Convert to **Auto Scaling group** across **3 AZs** with **EFS** for shared state (if POSIX fits) or move to **Aurora** for database. Use **multi-AZ** RDS/EFS and distribute ALB targets across AZs. For caches, use **ElastiCache Multi-AZ** with automatic failover.

Sample Points:

- Spread across 3 AZs by default.
- Replace EBS single-AZ state with EFS/Aurora.
- Health checks per AZ for fast failover.

Example Code:

```
aws autoscaling create-auto-scaling-group --availability-zones a,b,c
...
```

Q21: SSM Session Manager to private EC2 fails; SSH bastion is forbidden. Fix secure access quickly.

Answer:
Attach **AmazonSSMManagedInstanceCore** role to the instance, ensure SSM agent is running, allow egress to SSM endpoints via **VPC endpoints** (ssm, ec2messages, ssmmessages), and block SSH in SG. Verify IAM user permissions for Session Manager.
Sample Points:

- Instance role + SSM agent required.
- Use interface endpoints in private VPC.
- No inbound SSH needed.
  Example Code:

```
aws ec2 create-vpc-endpoint --vpc-endpoint-type Interface
--service-name com.amazonaws.ap-south-1.ssm ...
```

Q22: WAF needed only for `/login` and `/api/*` paths on CloudFront. Keep perf high and rules cheap.

Answer:
Attach **AWS WAF** to CloudFront and scope rules with **rate-based** + **managedrulegroups**; use **scope-down statements** to match the specific URIs so other paths aren't inspected. Monitor with sampled requests before enforcing.
SamplePoints:

- Scope-down limits inspection cost/latency.
- Start in count mode; then block.
- Rate-based to resist credential stuffing.
  ExampleCode(scope-downJSONexcerpt):

```
"Statement":{"ByteMatchStatement":{"SearchString":"/login","FieldToMat
ch":{"UriPath":{}},"TextTransformations":[{"Priority":0,"Type":"NONE"}
],"PositionalConstraint":"STARTS_WITH"}}
```

Q23: S3 inventory team needs Athena queries over JSON logs with partitioning and lifecycle control.

Answer:
Land logs in S3 with **Hive-style partitions** (`dt=YYYY-MM-DD`), create an **Athena external table** with `partitioned_by`, **MSCK REPAIR TABLE** or use **Glue crawler**. Lifecycle: 30 days in Standard → IA; 180 days to Glacier. Use **CTAS** to write optimized Parquet for faster/cheaper scans.
Sample Points:

- Partition on date or tenant.
- Convert to Parquet to cut cost.
- Lifecycle transitions for cold data.

Example Code (DDL):

```
CREATE EXTERNAL TABLE logs(
  level string, msg string, ts string
) PARTITIONED BY (dt string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://bucket/logs/';
```

Q24: Cost anomaly: NAT Gateway data processing exploded last week. Contain and prevent repeat.

Answer:
Check VPC Flow Logs to see which subnets/instances egressed most. Move S3/ECR/DynamoDB to **VPC endpoints**, route third-party pulls through a **proxy** with caching, and consolidate outbound to a single NAT per AZ (no cross-AZ). Educate teams on avoiding public package mirrors when private mirrors exist.
Sample Points:

- Replace public AWS egress with endpoints.
- AZ-aligned NAT to avoid cross-AZ charges.
- Cache/proxy dependency downloads.

Example Code:

```
aws ec2 create-vpc-endpoint --vpc-endpoint-type Gateway --service-name
com.amazonaws.ap-south-1.s3 ...
```

Q25: You're asked to enforce org-wide "no public S3 buckets" and detect drift automatically.

Answer:
Use **SCPs** denying `PutBucketAcl/PutBucketPolicy` that grant public access, enable **Block PublicAccess** at the **account** level via **Organization** defaults, and set **Config rules** (`s3-bucket-public-read-prohibited`, `s3-bucket-public-write-prohibited`) with **Security Hub** aggregation. Auto-remediate with **SSM Automation**.
Sample Points:

- SCP + Block Public Access = hard guardrail.
- AWS Config for continuous detection.
- Automation doc for remediation.

Example Code:

```
aws organizations attach-policy --policy-id p-abcdefgh --target-id
<account-id>
aws config put-config-rule --config-rule
file://s3-public-read-prohibited.json
```