**1. How do you handle blue/green deployments in AWS?**

**Answer:**  
Blue/green deployment involves maintaining two separate environments—**blue (current)** and **green (new)**. Traffic is switched from blue to green once testing is complete.

**In AWS:**

* Use **Elastic Beanstalk**, **CodeDeploy**, or **ECS**.
* With **CodeDeploy**, define deployment group using **"blue/green deployment type"**.
* Use **Load Balancers** to shift traffic between target groups.
* Rollback is easy by redirecting traffic back to the blue environment.

**Real-World Tip:** Combine with **CloudWatch alarms** for auto-rollback on failure.

**2. Describe your end-to-end CI/CD pipeline in AWS.**

**Answer:**  
A typical pipeline includes:

* **Source Stage:** GitHub/CodeCommit trigger
* **Build Stage:** AWS CodeBuild compiles, runs tests
* **Test Stage:** Unit/Integration testing via CodeBuild or 3rd-party tools
* **Approval Stage:** Manual approval using CodePipeline
* **Deploy Stage:** Deploy via AWS CodeDeploy/ECS/EKS/CloudFormation

**Tools Used:** CodePipeline, CodeBuild, CodeDeploy, S3, Lambda (notifications), CloudWatch Logs.

**3. How do you secure sensitive data like environment variables or secrets in pipelines?**

**Answer:**

* Use **AWS Secrets Manager** or **SSM Parameter Store** (with encryption).
* Reference secrets in **CodeBuild via environment variables**.
* Avoid hardcoding secrets in code/repos.

**Best Practice:** Use IAM roles for least privilege and rotate secrets periodically.

**4. How do you achieve zero-downtime deployments?**

**Answer:**

* Use **blue/green** or **canary deployments** (e.g., CodeDeploy with linear/canary strategy).
* In **ECS**, use rolling updates with minHealthyPercent and maxPercent.
* Monitor with **CloudWatch** for issues and enable rollback policies.

**5. Explain how you integrate testing in a pipeline.**

**Answer:**

* Define test steps in **buildspec.yml** during CodeBuild.
* Example:

yaml

CopyEdit

phases:

build:

commands:

- npm install

- npm test

* Fail pipeline if tests fail.
* Use CodeCoverage and report to dashboard.

**6. How can you speed up a slow CodeBuild job?**

**Answer:**

* Use **custom Docker images** with pre-installed dependencies.
* Enable **caching** in buildspec.yml for dependencies like Maven/NPM.
* Use higher-performance instance types.
* Minimize unnecessary build steps.

**7. How do you design a high-availability solution using EC2 and Load Balancers?**

**Answer:**

* Deploy EC2 instances in **multiple Availability Zones** (AZs).
* Use **Auto Scaling Group (ASG)** to maintain instance count.
* Attach ASG to an **Application Load Balancer (ALB)**.
* Enable health checks on the ALB to remove unhealthy instances.

**Bonus Tip:** Use Route 53 for DNS failover and RDS Multi-AZ for database HA.

**8. What happens if one AZ goes down? How will you handle it?**

**Answer:**

* If you're using **Multi-AZ ASG**, traffic automatically routes to healthy instances in the other AZ.
* For databases, use **RDS Multi-AZ** which automatically fails over to the standby.
* Ensure **subnets, NAT gateways, and S3 access** are not AZ-restricted.

**9. How do you manage cost optimization in DevOps projects?**

**Answer:**

* Enable **AWS Trusted Advisor** and **Cost Explorer**.
* Set **CloudWatch billing alarms**.
* Use **S3 lifecycle rules** to move data to Glacier or delete.
* Right-size EC2 and RDS using **Compute Optimizer**.
* Use **Spot Instances** for non-production workloads.

**10. How do you ensure backups for EC2/RDS are in place?**

**Answer:**

* For EC2: Use **EBS snapshot policies** or **AWS Backup**.
* For RDS: Enable **automated backups** or use manual snapshots.
* Automate backups using Lambda + CloudWatch Events.

**11. How do you manage EC2 instance states in Dev, Test, and Prod environments?**

**Answer:**

* Use **tags** to identify environments (Environment=Dev).
* Use **AWS Systems Manager (SSM)** for automating start/stop tasks.
* Use **Lambda** functions on a schedule to stop unused Dev instances after hours.

**🧱 Infrastructure as Code (IaC) – Terraform & CloudFormation**

**12. How do you structure Terraform for multiple environments (Dev/Prod)?**

**Answer:**

* Use **modules** for reusable components (e.g., VPC, EC2).
* Create separate **workspace** or directories (e.g., env/dev, env/prod).
* Maintain different variable files for each environment (dev.tfvars, prod.tfvars).

**13. What is the difference between terraform plan and terraform apply?**

**Answer:**

* terraform plan: Shows what changes will be made (no changes applied).
* terraform apply: Executes the plan and applies changes to real infrastructure.

Use plan to review before applying in production.

**14. How do you manage Terraform state securely and remotely?**

**Answer:**

* Store state in **S3** with encryption enabled.
* Enable **DynamoDB for state locking** to prevent concurrent changes.
* Use IAM policies to restrict access to state files.

**15. How do you handle Terraform drift?**

**Answer:**

* Run terraform plan regularly to detect drift.
* Use terraform refresh to sync state with real infra.
* Enable **AWS Config** to monitor drift outside of Terraform.

**16. What are lifecycle rules in Terraform?**

**Answer:**  
Used in the lifecycle block:

* prevent\_destroy: Avoid accidental deletion.
* create\_before\_destroy: Ensures replacement resources are created before destroying the old ones.

Example:

hcl

CopyEdit

lifecycle {

prevent\_destroy = true

}

**17. Explain a real issue you faced with Terraform and how you resolved it.**

**Answer:**  
*Example:* Accidentally deleted a resource because prevent\_destroy was missing. Fixed it by:

* Adding the lifecycle block.
* Restoring the resource using manual backup.
* Reviewing plan output thoroughly before applying changes in prod.

**📊 Monitoring & Logging**

**18. How do you monitor EC2, RDS, and Lambda in AWS?**

**Answer:**

* **EC2**: Use CloudWatch agent for disk/memory + native metrics.
* **RDS**: Enhanced monitoring + CloudWatch metrics.
* **Lambda**: Use built-in metrics and **AWS X-Ray** for tracing.
* Set **CloudWatch Alarms** to notify via SNS when thresholds are breached.

**19. What’s the difference between CloudTrail and CloudWatch?**

**Answer:**

| **Feature** | **CloudWatch** | **CloudTrail** |
| --- | --- | --- |
| Purpose | Monitors metrics/logs/performance | Audits API calls/events |
| Data Type | Metrics & logs | API activity |
| Used For | Operational monitoring | Security/audit |

**20. How do you set up centralized logging for a multi-account AWS setup?**

**Answer:**

* Use **CloudWatch cross-account log sharing**.
* Or forward logs from each account to a central **Kinesis Data Stream** or **S3 bucket**.
* Use **AWS Organizations**, **IAM roles**, and **resource policies** for access control.

**21. How do you implement least privilege using IAM?**

**Answer:**

* **Least privilege** means giving users/services only the permissions they need—nothing more.
* Use **IAM roles** instead of IAM users wherever possible.
* Define **fine-grained IAM policies** with **specific actions and resource ARNs**.
* Avoid using wildcards like "Action": "\*" or "Resource": "\*" in production policies.
* Regularly **audit IAM policies** using:
  + **IAM Access Analyzer**
  + **AWS Trusted Advisor (Security checks)**

**Real-Time Example:**  
Use a scoped-down policy allowing an EC2 instance to access only a specific S3 bucket:

json

CopyEdit

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::mybucket/data/\*"

}

]

}

**22. How do you rotate access keys securely for automation scripts?**

**Answer:**

* **Avoid static access keys** by using **IAM roles with instance profiles** or **Lambda execution roles**.
* If you must use keys:
  + Create a **new access key**, update your app/script.
  + Test the new key, then **deactivate and delete** the old one.
* Automate rotation using:
  + **AWS Secrets Manager** with Lambda rotation.
  + Schedule **AWS Config rules** or **SNS alerts** for old keys.

**Pro Tip:** Enable MFA and enforce periodic key rotation policies using **IAM Password Policy**.

**23. How do you audit and detect unusual activity in an AWS account?**

**Answer:**

* Enable **AWS CloudTrail** in all regions to capture API activity.
* Use **AWS Config** to track resource configuration changes.
* Set up **Amazon GuardDuty** to detect anomalies like:
  + Unusual logins
  + Port scanning
  + Unauthorized access
* Use **CloudWatch Alarms** for IAM policy changes, root account usage, etc.

**Real-Time Scenario:**  
Detect a new IAM user created outside of normal working hours:

* Trigger CloudTrail event → Send SNS alert or Lambda response.

**24. What are some best practices for S3 bucket security?**

**Answer:**

* **Block all public access** unless explicitly required.
* Use **Bucket Policies** and **IAM policies** to define fine-grained access.
* Enable **default encryption** (AES-256 or AWS KMS).
* Use **S3 Access Logs** to monitor access patterns.
* Enable **Versioning** and **Object Lock** (for compliance).

**Bonus:**  
You can use **S3 Macie** to scan for sensitive data like PII or credentials in buckets.

**25. Difference between ECS and EKS? Which one have you used and why?**

**Answer:**

| **Feature** | **ECS (Elastic Container Service)** | **EKS (Elastic Kubernetes Service)** |
| --- | --- | --- |
| Orchestration | AWS native | Kubernetes (open-source) |
| Complexity | Easier to set up and manage | Steeper learning curve |
| Control | AWS manages orchestration fully | You manage Kubernetes nodes, configs |
| Use case | Ideal for small to medium workloads | Large-scale, multi-cloud compatibility |

**Which one to choose?**

* Use **ECS** if you prefer AWS-native services, simple setup, and deep integration with Fargate.
* Use **EKS** if you're already using **Kubernetes**, or need portability across clouds.

**Real-Time Example:**  
In a microservices project, ECS was chosen for faster setup and less operational overhead, using **Fargate** to remove the need to manage EC2.

**26. How do you deploy a Dockerized app on ECS Fargate?**

**Answer:**

**Steps:**

1. **Build Docker Image** and push to **Amazon ECR**.
2. Define an **ECS Task Definition**:
   * Container name, image URI, ports, CPU/memory.
3. Create an **ECS Cluster** and a **Service**.
4. Choose **Fargate** launch type (serverless).
5. Attach a **Load Balancer** if needed.
6. Configure **IAM roles** for execution and logging.
7. Monitor via **CloudWatch** and ECS console.

**Example Command:**

bash

CopyEdit

aws ecs run-task \

--launch-type FARGATE \

--network-configuration 'awsvpcConfiguration={subnets=[subnet-abc],securityGroups=[sg-xyz],assignPublicIp="ENABLED"}' \

--cluster my-cluster \

--task-definition my-task

**27. How do you handle logging and monitoring in containers?**

**Answer:**

* Use **AWS CloudWatch Logs**:
  + Configure log drivers like awslogs in ECS.
* For EKS:
  + Use **Fluent Bit**, **CloudWatch Agent**, or **Prometheus + Grafana**.
* Enable **container insights** in ECS/EKS to collect CPU, memory, network, and disk metrics.

**Example ECS Task Logging Configuration:**

json

CopyEdit

"logConfiguration": {

"logDriver": "awslogs",

"options": {

"awslogs-group": "/ecs/my-app",

"awslogs-region": "us-east-1",

"awslogs-stream-prefix": "ecs"

}

}

**Pro Tip:** Set log retention policies and create alerts for log patterns (e.g., 500 errors).

**28. What is the purpose of Task Roles in ECS?**

**Answer:**

* ECS **Task Role** grants permissions to the application inside the container to access AWS services (e.g., S3, DynamoDB).
* Better security practice than embedding credentials in code.
* It’s an **IAM Role assumed by the task**, not the ECS service.

**Example Use Case:**  
A container running in ECS needs to read/write data to S3. You attach a Task Role with s3:PutObject permissions to the ECS Task Definition.

**29. You deployed a new app version, but users report errors. What’s your debugging approach?**

**Answer:**

**Step-by-step approach:**

1. **Check Load Balancer Health:**
   * Verify ALB/NLB target group health checks.
   * Are new instances marked as healthy?
2. **Check Logs:**
   * **CloudWatch Logs**: Look for application errors.
   * **ECS logs** (if containerized) or EC2 instance logs.
3. **Roll Back Deployment:**
   * Use **CodeDeploy automatic rollback**.
   * Or revert to the previous container/task revision in ECS/EKS.
4. **Check Monitoring Dashboards:**
   * CloudWatch metrics (e.g., CPU, memory, latency).
   * Look for spikes in 5xx errors.
5. **Audit Code & Configs:**
   * Check if recent commits introduced breaking changes.
   * Validate env variables, IAM roles, and security groups.

**Pro Tip:** Use canary/blue-green deployments to test before full rollout.

**30. How would you automate AMI creation and deployment?**

**Answer:**

**Option 1: Packer**

* Define a JSON/HashiCorp config to build AMIs.
* Integrate with **Jenkins**, **GitHub Actions**, or **CodeBuild**.

**Option 2: EC2 Image Builder**

* AWS-managed tool to automate AMI pipelines.
* Supports image hardening, patching, testing, and tagging.

**Automation Workflow:**

1. Packer/EC2 Image Builder creates AMI.
2. Update **Auto Scaling Launch Template**.
3. Use **Terraform or CI/CD** to deploy new instances with the new AMI.

**31. How would you implement auto-scaling based on memory or custom metrics?**

**Answer:**

**EC2 Auto Scaling (custom metric):**

1. Install **CloudWatch Agent** on EC2.
2. Publish memory usage as custom metric.
3. Create a **CloudWatch Alarm** for threshold (e.g., memory > 80%).
4. Attach this to a scaling policy in **Auto Scaling Group**.

**ECS Auto Scaling (Service level):**

* Configure ECS Service Auto Scaling using **target tracking policy** on:
  + CPUUtilization
  + MemoryUtilization (Fargate/ECS)
  + Custom metrics (e.g., queue length, request count)

**EKS:**

* Use **Horizontal Pod Autoscaler (HPA)** based on metrics-server or custom metrics via Prometheus Adapter.

**Example Scaling Policy (CLI):**

bash

CopyEdit

aws application-autoscaling put-scaling-policy \

--service-namespace ecs \

--scalable-dimension ecs:service:DesiredCount \

--resource-id service/my-cluster/my-service \

--policy-name memory-scale-out \

--policy-type TargetTrackingScaling \

--target-tracking-scaling-policy-configuration file://policy.json