#### **1. If a CloudFormation stack fails and rolls back after one hour, how can we troubleshoot and resolve the issue?**

1. **Check CloudFormation Events**:
   * Go to the AWS **CloudFormation console** → Select the stack → Click on the **Events tab** to find failure reasons.
2. **Inspect Logs in AWS Services**:
   * If the stack involves Lambda, **check CloudWatch Logs** for errors.
   * If it creates an EC2 instance, **check EC2 instance logs**.
3. **Validate IAM Permissions**:
   * Ensure that IAM roles and policies allow the necessary actions.
4. **Check Resource Quotas**:
   * Some resources (e.g., VPCs, ELBs) have **service limits** that may block creation.
5. **Use AWS CloudFormation Drift Detection**:
   * Run **drift detection** to see if the current state of resources differs from the expected CloudFormation configuration.
6. **Disable Rollback for Debugging**:
   * Create a stack with the **Rollback on failure** option disabled to retain failed resources for investigation.
7. **Manually Deploy in Stages**:
   * If troubleshooting a complex stack, deploy individual resources separately to identify issues.

### **Cost Optimization & Monitoring**

#### **2. How can data be secured in AWS services?**

1. **Encryption**:
   * **At rest**: Enable **S3 SSE, EBS encryption, and RDS encryption**.
   * **In transit**: Use **TLS (SSL) for secure communication**.
2. **IAM Policies & Access Control**:
   * Follow **the principle of least privilege**.
   * Use **IAM roles instead of IAM users** where possible.
3. **Enable AWS Security Services**:
   * Use **AWS Shield** for DDoS protection.
   * Use **AWS WAF** to filter malicious traffic.
   * Enable **AWS GuardDuty** for anomaly detection.
4. **Logging & Monitoring**:
   * Enable **AWS CloudTrail** for audit logging.
   * Use **Amazon CloudWatch** for monitoring security logs.

#### **3. How can I reduce the size of a logical disk on a Linux machine when it is sometimes unused, to save costs?**

**Identify the Current Disk Usage**:  
  
  
df -h

lvdisplay

**Unmount the Filesystem Temporarily**:  
  
umount /dev/mapper/volume-name

**Reduce the Logical Volume Size**:  
  
lvreduce -L 20G /dev/mapper/volume-name

**Resize the Filesystem**:  
  
resize2fs /dev/mapper/volume-name

**Remount the Filesystem**:  
  
mount /dev/mapper/volume-name /mountpoint

### **DevOps & Security**

#### **4. How can a 404 error in an application be resolved at the application level?**

1. **Check the Application Logs**:
   * Review logs in **CloudWatch Logs (Lambda)** or var/log/httpd (EC2).
2. **Verify Routes & URL Mappings**:
   * Check if the correct **API routes or UI paths** are defined.
3. **Ensure Static Files Exist**:
   * If serving a UI, verify that **CSS/JS/HTML files** exist in the correct location.
4. **Check Server Configuration**:
   * For **Nginx/Apache**, ensure correct settings in nginx.conf or .htaccess.
5. **API Gateway Mapping Issues**:
   * If using API Gateway, verify the **resource path and integration response settings**.

#### **5. If a UI-based application is deployed in Kubernetes and accessible via a URL, how can we deploy it in Kubernetes, and how will the backend API respond?**

1. **Deploy UI and Backend Separately**:
   * Create separate **Deployments & Services** for UI and backend.
   * Expose the backend API using **ClusterIP/LoadBalancer** service.
2. **Expose UI via Ingress**:
   * Use **Ingress Controller** (NGINX) to route traffic to the UI.

Example:  
  
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: ui-ingress

spec:

rules:

- host: app.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: ui-service

port:

number: 80

1. **Ensure Backend API Is Reachable**:

Define an **internal service** for the backend:  
  
apiVersion: v1

kind: Service

metadata:

name: backend-service

spec:

selector:

app: backend

ports:

- protocol: TCP

port: 8080

targetPort: 8080

1. **Configure UI to Call Backend**:
   * Use environment variables in the UI container to define the backend API URL.

#### **6. How can application vulnerabilities be checked in AWS, and which tools can be used for this?**

1. **AWS Native Tools**:
   * **AWS Inspector**: Scans EC2 instances for vulnerabilities.
   * **Amazon GuardDuty**: Detects security threats.
   * **AWS WAF**: Protects against web exploits.
2. **Third-Party Tools**:
   * **Snyk** (for dependencies security).
   * **Qualys** (for vulnerability scanning).
   * **OWASP ZAP** (for penetration testing).
3. **Best Practices**:
   * **Enable Logging**: Use **CloudTrail** and **CloudWatch**.
   * **Patch Regularly**: Keep software updated.

#### **7. If a web application is hosted, how can we secure it, and what security measures can we apply without using third-party tools?**

1. **Enable HTTPS (SSL/TLS)**:
   * Use **AWS Certificate Manager** to provision **SSL certificates**.
   * Configure **ELB or API Gateway** with HTTPS.
2. **Use AWS WAF for Web Security**:
   * Protect against **SQL Injection, XSS, and DDoS** attacks.

Example AWS WAF rule to block common exploits:  
  
{

"Name": "SQLInjectionRule",

"Priority": 1,

"Action": { "Block": {} },

"Statement": {

"SqliMatchStatement": {

"FieldToMatch": { "QueryString": {} },

"TextTransformations": [ { "Priority": 0, "Type": "NONE" } ]

}

}

}

1. **Secure IAM and API Access**:
   * Use **IAM roles** instead of credentials.
   * Restrict **API Gateway access** using IAM or Lambda Authorizer.
2. **Enable Security Headers** (in Nginx or Apache):
   * Enforce **CSP, X-Frame-Options, and HSTS**.

Example (Nginx):  
  
add\_header X-Frame-Options DENY;

add\_header X-XSS-Protection "1; mode=block";

add\_header Strict-Transport-Security "max-age=31536000; includeSubDomains";

1. **Monitor & Audit**:
   * Enable **CloudTrail** for API access logs.
   * Set up **CloudWatch Alarms** for anomaly detection.