Let's **customize the backup automation script with Docker, Ansible, and Kubernetes integration** to make it more robust, scalable, and ready for a production DevOps environment.

**Step 1: Dockerizing the Backup Script**

Dockerizing the script allows it to run in isolated, consistent environments across development and production.

**Dockerfile**

Dockerfile

Copy code

# Base image

FROM amazonlinux:2

# Install dependencies

RUN yum install -y tar aws-cli crontabs && yum clean all

# Set working directory

WORKDIR /app

# Copy the script

COPY backup\_script.sh /app/backup\_script.sh

# Make the script executable

RUN chmod +x /app/backup\_script.sh

# Run the backup script

ENTRYPOINT ["/app/backup\_script.sh"]

**Steps to Build and Run**

1. Build the Docker image:

bash

Copy code

docker build -t backup-script .

1. Run the container:

bash

Copy code

docker run --rm -v /var/log/app:/var/log/app -v /backup:/backup backup-script

**Key Features in the Dockerized Solution**

* Uses **bind mounts** (-v) to map host directories to the container.
* Ensures AWS CLI is pre-installed for S3 operations.
* Cleans unnecessary files to reduce image size.

**Step 2: Automating with Ansible**

Ansible ensures the deployment of the backup solution across multiple servers.

**Ansible Playbook**

yaml

Copy code

---

- name: Deploy Backup Automation

hosts: all

become: true

tasks:

- name: Install Docker

yum:

name: docker

state: present

- name: Start Docker service

service:

name: docker

state: started

enabled: true

- name: Pull backup Docker image

docker\_image:

name: backup-script

tag: latest

source: build

build:

path: /path/to/dockerfile

- name: Run backup container

docker\_container:

name: backup-container

image: backup-script:latest

state: started

volumes:

- /var/log/app:/var/log/app

- /backup:/backup

**Run the Playbook**

1. Update the inventory file with your server IPs.
2. Execute the playbook:

bash

Copy code

ansible-playbook -i inventory backup-playbook.yml

**Step 3: Running the Backup on Kubernetes**

Kubernetes enables deployment and scaling in a cloud-native environment.

**Kubernetes Manifest**

yaml

Copy code

apiVersion: batch/v1

kind: CronJob

metadata:

name: backup-cronjob

spec:

schedule: "0 2 \* \* \*" # Run daily at 2 AM

jobTemplate:

spec:

template:

spec:

containers:

- name: backup

image: backup-script:latest

volumeMounts:

- name: app-logs

mountPath: /var/log/app

- name: backup-data

mountPath: /backup

env:

- name: AWS\_ACCESS\_KEY\_ID

valueFrom:

secretKeyRef:

name: aws-secret

key: access-key-id

- name: AWS\_SECRET\_ACCESS\_KEY

valueFrom:

secretKeyRef:

name: aws-secret

key: secret-access-key

restartPolicy: OnFailure

volumes:

- name: app-logs

hostPath:

path: /var/log/app

- name: backup-data

hostPath:

path: /backup

**Key Features**

1. **CronJob:** Automates backups using Kubernetes' native scheduler.
2. **Secrets Management:** AWS credentials are stored securely using Kubernetes Secrets.
3. **HostPath Volumes:** Maps host directories to the container.

**Deployment Steps**

1. Create a Kubernetes secret for AWS credentials:

bash

Copy code

kubectl create secret generic aws-secret \

--from-literal=access-key-id=<your-access-key-id> \

--from-literal=secret-access-key=<your-secret-access-key>

1. Apply the CronJob manifest:

bash

Copy code

kubectl apply -f backup-cronjob.yml

1. Check CronJob logs:

bash

Copy code

kubectl logs job/<backup-job-name>

**Mock Interview Questions**

**Docker:**

1. **Q:** How does the ENTRYPOINT in the Dockerfile differ from CMD?  
   **A:** ENTRYPOINT sets the command that always runs in the container, while CMD provides default arguments.
2. **Q:** How would you handle secret management in Docker?  
   **A:** Use Docker secrets or environment variables, but ensure they’re sourced securely.

**Ansible:**

1. **Q:** How do you ensure idempotency in Ansible tasks?  
   **A:** Use Ansible modules like yum and docker\_image, which only apply changes when needed.
2. **Q:** What’s the advantage of using Ansible for this use case?  
   **A:** It simplifies deployment across multiple servers and ensures consistency.

**Kubernetes:**

1. **Q:** Why use a CronJob over a manual job in Kubernetes?  
   **A:** A CronJob automates repetitive tasks on a schedule, reducing manual effort.
2. **Q:** How would you scale this solution to multiple regions?  
   **A:** Use Helm charts for templating and deploy CronJob instances across clusters in different regions.