Cost optimization is a crucial aspect of managing AWS infrastructure. You can automate various tasks to reduce costs, especially by using shell scripts with AWS CLI to manage resources efficiently. Here are some cost-cutting automation ideas and related shell scripts:

**1. Automated EC2 Instance Shutdown/Startup**

Automatically stopping non-production instances (e.g., dev/test) during non-working hours to save costs.

**Shell Script**: Schedule to stop instances at a specific time (e.g., at the end of the workday) and start them at the beginning.

bash

CopyEdit

# Stop all EC2 instances with a specific tag (e.g., Environment=Dev)

aws ec2 describe-instances \

--filters "Name=tag:Environment,Values=Dev" "Name=instance-state-name,Values=running" \

--query 'Reservations[\*].Instances[\*].InstanceId' \

--output text | while read instance; do

echo "Stopping instance: $instance"

aws ec2 stop-instances --instance-ids $instance

done

**To start instances**:

bash

CopyEdit

# Start all EC2 instances with a specific tag (e.g., Environment=Dev)

aws ec2 describe-instances \

--filters "Name=tag:Environment,Values=Dev" "Name=instance-state-name,Values=stopped" \

--query 'Reservations[\*].Instances[\*].InstanceId' \

--output text | while read instance; do

echo "Starting instance: $instance"

aws ec2 start-instances --instance-ids $instance

done

**Automation**: Use **AWS Lambda** and **CloudWatch Events** to schedule these scripts automatically (e.g., stop instances at 7 PM and start them at 9 AM).

**2. Unused EBS Volumes Cleanup**

Identify and delete unattached/unused Elastic Block Storage (EBS) volumes, which can accumulate costs.

**Shell Script**:

bash

CopyEdit

# List all unattached EBS volumes and delete them

aws ec2 describe-volumes \

--filters "Name=status,Values=available" \

--query 'Volumes[\*].VolumeId' \

--output text | while read volume; do

echo "Deleting volume: $volume"

aws ec2 delete-volume --volume-id $volume

done

**Automation**: Schedule this script to run periodically (e.g., daily or weekly) using **AWS Lambda** or a **cron job** on an EC2 instance.

**3. Snapshot Lifecycle Automation**

AWS snapshots can be costly if not managed properly. Automate the deletion of old snapshots to avoid high storage costs.

**Shell Script**: Delete EBS snapshots older than a specified number of days (e.g., 30 days).

bash

CopyEdit

# Define the retention period (e.g., 30 days)

RETENTION\_DAYS=30

# Find and delete snapshots older than the retention period

aws ec2 describe-snapshots --owner-ids self \

--query "Snapshots[?StartTime<=\`$(date -d "$RETENTION\_DAYS days ago" --utc +%Y-%m-%d)\`].SnapshotId" \

--output text | while read snapshot; do

echo "Deleting snapshot: $snapshot"

aws ec2 delete-snapshot --snapshot-id $snapshot

done

**Automation**: Schedule this with **cron** or use **Lambda** to periodically clean up old snapshots.

**4. Idle Load Balancer Cleanup**

Identify and delete idle Load Balancers (with no traffic) to avoid unnecessary costs.

**Shell Script**:

bash

CopyEdit

# List all ELBs and describe their request count

aws elb describe-load-balancers \

--query 'LoadBalancerDescriptions[\*].{Name:LoadBalancerName}' \

--output text | while read elb; do

request\_count=$(aws cloudwatch get-metric-statistics \

--metric-name RequestCount \

--start-time $(date -u -d '30 minutes ago' +%FT%T) \

--end-time $(date -u +%FT%T) \

--period 60 \

--namespace AWS/ELB \

--statistics Sum \

--dimensions Name=LoadBalancerName,Value=$elb \

--query 'Datapoints[0].Sum' \

--output text)

if [ "$request\_count" = "None" ] || [ "$request\_count" -eq 0 ]; then

echo "Deleting idle ELB: $elb"

aws elb delete-load-balancer --load-balancer-name $elb

fi

done

**Automation**: Set up a **CloudWatch** alert to trigger when ELBs have low or no traffic and then invoke this script through **Lambda**.

**5. S3 Object Lifecycle Policies**

Set up lifecycle policies to automatically move data from S3 Standard to cheaper storage classes (e.g., S3 Glacier) or delete objects after a certain period.

**Shell Script**: Apply a lifecycle policy to move objects older than 30 days to Glacier and delete objects older than 365 days.

bash

CopyEdit

# Set lifecycle policy on a bucket to transition objects to Glacier after 30 days and delete after 365 days

aws s3api put-bucket-lifecycle-configuration \

--bucket <bucket-name> \

--lifecycle-configuration '{

"Rules": [

{

"ID": "TransitionToGlacier",

"Filter": {},

"Status": "Enabled",

"Transitions": [

{

"Days": 30,

"StorageClass": "GLACIER"

}

],

"Expiration": {

"Days": 365

}

}

]

}'

**Automation**: This policy is automatically applied to the S3 bucket. No need for regular script execution, but review policies periodically.

**6. Rightsizing EC2 Instances**

Identify underutilized EC2 instances and recommend resizing to a smaller instance type.

**Shell Script**: Check CPU utilization for all instances over a period and suggest downsizing.

bash

CopyEdit

# Identify instances with low CPU usage (less than 10%) over the past 7 days

aws ec2 describe-instances --query 'Reservations[\*].Instances[\*].InstanceId' --output text | while read instance; do

avg\_cpu=$(aws cloudwatch get-metric-statistics \

--metric-name CPUUtilization \

--start-time $(date -d '7 days ago' --utc +%FT%T) \

--end-time $(date --utc +%FT%T) \

--period 86400 \

--namespace AWS/EC2 \

--statistics Average \

--dimensions Name=InstanceId,Value=$instance \

--query 'Datapoints[0].Average' \

--output text)

if (( $(echo "$avg\_cpu < 10.0" | bc -l) )); then

echo "Instance $instance has low CPU usage ($avg\_cpu%). Consider downsizing."

fi

done

**Automation**: Set up a **CloudWatch** alarm for underutilized instances and trigger an action (e.g., sending a report or auto-resizing).

**7. Delete Old Lambda Versions**

AWS Lambda stores multiple versions of a function, which can accumulate over time and increase costs.

**Shell Script**: Delete old Lambda versions, keeping only the latest N versions.

bash

CopyEdit

FUNCTION\_NAME="your-lambda-function-name"

KEEP\_VERSIONS=5

# List all Lambda versions

versions=$(aws lambda list-versions-by-function --function-name $FUNCTION\_NAME --query 'Versions[?Version!=`$LATEST`].[Version]' --output text)

# Count versions and delete the older ones, keeping only the latest N

count=0

for version in $(echo "$versions" | sort -nr); do

count=$((count+1))

if [ "$count" -gt "$KEEP\_VERSIONS" ]; then

echo "Deleting Lambda version: $version"

aws lambda delete-function --function-name $FUNCTION\_NAME --qualifier $version

fi

done

By automating these cost-saving strategies, you can significantly reduce your AWS bill while maintaining operational efficiency. Let me know if you need further assistance or customization on any of these scripts!