1. **Write a shell script to check memory usage in linux server and send sns notification if it exceeds threshold .**

**#!/bin/bash**

**# Set threshold (e.g., 80% of total memory)**

**THRESHOLD=80**

**# Get total memory**

**TOTAL\_MEM=$(awk '/MemTotal/ {print $2}' /proc/meminfo)**

**# Get available memory**

**AVAILABLE\_MEM=$(awk '/MemAvailable/ {print $2}' /proc/meminfo)**

**# Calculate used memory**

**USED\_MEM=$((TOTAL\_MEM - AVAILABLE\_MEM))**

**# Calculate percentage of used memory**

**USED\_PERC=$((USED\_MEM \* 100 / TOTAL\_MEM))**

**# Check if used memory percentage exceeds threshold**

**if [ $USED\_PERC -gt $THRESHOLD ]; then**

**# Send SNS notification**

**aws sns publish --topic-arn <SNS\_TOPIC\_ARN> --message "Memory usage exceeded $THRESHOLD% threshold!"**

**fi**

**In this modified script:**

**1. We use two variables TOTAL\_MEM and AVAILABLE\_MEM to store the total and available memory respectively.**

**2. We calculate the used memory USED\_MEM by subtracting AVAILABLE\_MEM from TOTAL\_MEM.**

**3. We calculate the percentage of used memory USED\_PERC by dividing USED\_MEM by TOTAL\_MEM and multiplying by 100.**

**4. The rest of the script remains the same.**

**Open a terminal and run the following command to start a memory-intensive process:**

**bash**

**stress --vm 1 --vm-bytes 512M --vm-hang 0**

**This will start a process that allocates 512MB of memory and keeps it occupied.**

**In another terminal, run the script:**

**./memory\_threshold.sh**

**Replace ./memory\_threshold.sh with the actual path to your script.**

**Observe the output of the script. If the memory usage exceeds the threshold (80% in this case), the script should send an SNS notification.**

**To stop the memory-intensive process, run:**

**pkill stress**

**This will terminate the stress process and free up the allocated memory.**

**2) Script to convert temperature from Celsius to Fahrenheit:**

**#!/bin/bash**

**read -p "Enter temperature in Celsius: " temp**

**let fahrenheit=temp\*9/5+32**

**echo "Temperature in Fahrenheit: $fahrenheit"**

**3) Script to calculate the product of numbers from 1 to n:**

**#!/bin/bash**

**read -p "Enter a number: " num**

**let product=1**

**for ((i=1; i<=num; i++)); do**

**let product\*=i**

**done**

**echo "Product: $product"**

**4) Script to calculate the sum of even numbers from 1 to n:**

**#!/bin/bash**

**read -p "Enter a number: " num**

**let sum=0**

**for ((i=1; i<=num; i++)); do**

**if (( i % 2 == 0 )); then**

**let sum+=i**

**fi**

**done**

**echo "Sum of even numbers: $sum"**

**5) Script to calculate the sum of odd numbers from 1 to n:**

**#!/bin/bash**

**read -p "Enter a number: " num**

**let sum=0**

**for ((i=1; i<=num; i++)); do**

**if (( i % 2 != 0 )); then**

**let sum+=i**

**fi**

**done**

**echo "Sum of odd numbers: $sum"**

**6) Script to check if a number is prime:**

**#!/bin/bash**

**read -p "Enter a number: " num**

**if [ $num -le 1 ]; then**

**echo "Not prime"**

**else**

**for ((i=2; i\*i<=num; i++)); do**

**if [ $((num % i)) -eq 0 ]; then**

**echo "Not prime"**

**exit**

**fi**

**done**

**echo "Prime"**

**fi**

**7) Script to print the first n Fibonacci numbers:**

**#!/bin/bash**

**read -p "Enter a number: " num**

**a=0**

**b=1**

**echo "$a"**

**echo "$b"**

**for ((i=2; i<num; i++)); do**

**let c=a+b**

**echo "$c"**

**a=$b**

**b=$c**

**done**

**8) Script to check if a string contains only digits:**

**#!/bin/bash**

**read -p "Enter a string: " str**

**if [[ $str =~ ^[0-9]+$ ]]; then**

**echo "Contains only digits"**

**else**

**echo "Contains non-digit characters"**

**fi**

**9) Script to automate the process of listing all the resources in an AWS account.**

**#!/bin/bash**

**###############################################################################**

**# Author: Rahul Ranjan**

**# Version: v0.0.1**

**# Script to automate the process of listing all the resources in an AWS account**

**#**

**# Below are the services that are supported by this script:**

**# 1. EC2**

**# 2. RDS**

**# 3. S3**

**# 4. CloudFront**

**# 5. VPC**

**# 6. IAM**

**# 7. Route53**

**# 8. CloudWatch**

**# 9. CloudFormation**

**# 10. Lambda**

**# 11. SNS**

**# 12. SQS**

**# 13. DynamoDB**

**# 14. EBS**

**#**

**# The script will prompt the user to enter the AWS region and the service for which the resources need to be listed.**

**#**

**# Usage: ./aws\_resource\_list.sh <aws\_region> <aws\_service>**

**# Example: ./aws\_resource\_list.sh us-east-1 ec2**

**#############################################################################**

**# Check if the required number of arguments are passed**

**if [ $# -ne 2 ]; then**

**echo "Usage: ./aws\_resource\_list.sh <aws\_region> <aws\_service>"**

**echo "Example: ./aws\_resource\_list.sh us-east-1 ec2"**

**exit 1**

**fi**

**# Assign the arguments to variables and convert the service to lowercase**

**aws\_region=$1**

**#aws\_service=$2**

**#the user could input "EC2", "ec2", or "Ec2", and it would still work**

**aws\_service=$(echo "$2" | tr '[:upper:]' '[:lower:]')**

**# Check if the AWS CLI is installed**

**if ! command -v aws &> /dev/null; then**

**echo "AWS CLI is not installed. Please install the AWS CLI and try again."**

**exit 1**

**fi**

**# Check if the AWS CLI is configured**

**if [ ! -d ~/.aws ]; then**

**echo "AWS CLI is not configured. Please configure the AWS CLI and try again."**

**exit 1**

**fi**

**# List the resources based on the service**

**case $aws\_service in**

**ec2)**

**echo "Listing EC2 Instances in $aws\_region"**

**aws ec2 describe-instances --region $aws\_region**

**;;**

**rds)**

**echo "Listing RDS Instances in $aws\_region"**

**aws rds describe-db-instances --region $aws\_region**

**;;**

**s3)**

**echo "Listing S3 Buckets (Global)"**

**aws s3api list-buckets**

**;;**

**cloudfront)**

**echo "Listing CloudFront Distributions (Global)"**

**aws cloudfront list-distributions**

**;;**

**vpc)**

**echo "Listing VPCs in $aws\_region"**

**aws ec2 describe-vpcs --region $aws\_region**

**;;**

**iam)**

**echo "Listing IAM Users (Global)"**

**aws iam list-users**

**;;**

**route53)**

**echo "Listing Route53 Hosted Zones (Global)"**

**aws route53 list-hosted-zones**

**;;**

**cloudwatch)**

**echo "Listing CloudWatch Alarms in $aws\_region"**

**aws cloudwatch describe-alarms --region $aws\_region**

**;;**

**cloudformation)**

**echo "Listing CloudFormation Stacks in $aws\_region"**

**aws cloudformation describe-stacks --region $aws\_region**

**;;**

**lambda)**

**echo "Listing Lambda Functions in $aws\_region"**

**aws lambda list-functions --region $aws\_region**

**;;**

**sns)**

**echo "Listing SNS Topics in $aws\_region"**

**aws sns list-topics --region $aws\_region**

**;;**

**sqs)**

**echo "Listing SQS Queues in $aws\_region"**

**aws sqs list-queues --region $aws\_region**

**;;**

**dynamodb)**

**echo "Listing DynamoDB Tables in $aws\_region"**

**aws dynamodb list-tables --region $aws\_region**

**;;**

**ebs)**

**echo "Listing EBS Volumes in $aws\_region"**

**aws ec2 describe-volumes --region $aws\_region**

**;;**

**\*)**

**echo "Invalid service. Please enter a valid service."**

**exit 1**

**;;**

**esac**

**10) write a script to create a security group and iam roles using a function . while calling use parameters for dynamic .**

**#!/bin/bash**

**# Variables**

**aws\_region="us-east-1" # Update with your AWS region**

**trust\_policy\_file="trust-policy.json" # Path to your IAM trust policy JSON file**

**# Function to create a security group**

**create\_security\_group() {**

**local sg\_name=$1**

**local sg\_description=$2**

**echo "Creating Security Group $sg\_name in region $aws\_region..."**

**# Create the security group**

**local sg\_id=$(aws ec2 create-security-group \**

**--region $aws\_region \**

**--group-name $sg\_name \**

**--description "$sg\_description" \**

**--query 'GroupId' \**

**--output text)**

**if [ $? -eq 0 ]; then**

**echo "Security Group created with ID: $sg\_id"**

**else**

**echo "Failed to create Security Group."**

**exit 1**

**fi**

**# Example: Add a rule to allow inbound SSH traffic**

**aws ec2 authorize-security-group-ingress \**

**--region $aws\_region \**

**--group-id $sg\_id \**

**--protocol tcp \**

**--port 22 \**

**--cidr 0.0.0.0/0**

**if [ $? -eq 0 ]; then**

**echo "Inbound SSH rule added to Security Group."**

**else**

**echo "Failed to add inbound SSH rule to Security Group."**

**exit 1**

**fi**

**}**

**# Function to create an IAM role and attach a policy**

**create\_iam\_role() {**

**local role\_name=$1**

**local assume\_role\_policy=$2**

**local policy\_arn=$3**

**echo "Creating IAM role $role\_name..."**

**# Create the IAM role**

**aws iam create-role \**

**--role-name $role\_name \**

**--assume-role-policy-document file://$assume\_role\_policy**

**if [ $? -eq 0 ]; then**

**echo "IAM role created: $role\_name"**

**else**

**echo "Failed to create IAM role."**

**exit 1**

**fi**

**# Attach the IAM policy to the role**

**echo "Attaching policy to IAM role..."**

**aws iam attach-role-policy \**

**--role-name $role\_name \**

**--policy-arn $policy\_arn**

**if [ $? -eq 0 ]; then**

**echo "Policy attached to role: $role\_name"**

**else**

**echo "Failed to attach policy to IAM role."**

**exit 1**

**fi**

**}**

**# Example function calls with parameters**

**create\_security\_group "sg-web" "Security group for web servers"**

**create\_security\_group "sg-db" "Security group for database servers"**

**create\_iam\_role "iamRoleWeb" "trust-policy-web.json" "arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"**

**create\_iam\_role "iamRoleDB" "trust-policy-db.json" "arn:aws:iam::aws:policy/AmazonEC2ReadOnlyAccess"**

**11) The create\_resources.sh script creates an AWS security group and IAM role, outputs their IDs and names for further use. The create\_ec2\_instance.sh script executes create\_resources.sh to retrieve these resources, then uses them to launch an EC2 instance with the specified configuration, including security group and IAM role. It waits for the EC2 instance to reach the running state before completing.**

### **Script 1: create\_resources.sh**

**This script will create security groups and IAM roles, and output their IDs and names.**

**#!/bin/bash**

**# Variables**

**aws\_region="us-east-1" # Update with your AWS region**

**security\_group\_name="my-security-group"**

**iam\_role\_name="my-iam-role"**

**iam\_policy\_arn="arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess" # Example policy ARN**

**key\_name="my-key-pair" # Replace with your key pair name for SSH access**

**# Function to create a security group**

**create\_security\_group() {**

**echo "Creating security group $security\_group\_name in region $aws\_region..."**

**sg\_id=$(aws ec2 create-security-group \**

**--region $aws\_region \**

**--group-name $security\_group\_name \**

**--description "Security group for EC2 instance" \**

**--query 'GroupId' \**

**--output text)**

**if [ $? -eq 0 ]; then**

**echo "Security group created with ID: $sg\_id"**

**echo "$sg\_id"**

**else**

**echo "Failed to create security group."**

**exit 1**

**fi**

**}**

**# Function to create an IAM role and attach a policy**

**create\_iam\_role() {**

**echo "Creating IAM role $iam\_role\_name..."**

**aws iam create-role \**

**--role-name $iam\_role\_name \**

**--assume-role-policy-document file://trust-policy.json**

**if [ $? -eq 0 ]; then**

**echo "IAM role created: $iam\_role\_name"**

**else**

**echo "Failed to create IAM role."**

**exit 1**

**fi**

**echo "Attaching policy to IAM role..."**

**aws iam attach-role-policy \**

**--role-name $iam\_role\_name \**

**--policy-arn $iam\_policy\_arn**

**if [ $? -eq 0 ]; then**

**echo "Policy attached to role: $iam\_role\_name"**

**else**

**echo "Failed to attach policy to IAM role."**

**exit 1**

**fi**

**echo "$iam\_role\_name"**

**}**

**# Create security group and IAM role**

**sg\_id=$(create\_security\_group)**

**iam\_role=$(create\_iam\_role)**

**# Output the results to use in the next script**

**echo "Security Group ID: $sg\_id"**

**echo "IAM Role Name: $iam\_role"**

### **Script 2: create\_ec2\_instance.sh**

**This script will use the output from create\_resources.sh to create an EC2 instance.**

**#!/bin/bash**

**# Variables**

**aws\_region="us-east-1" # Update with your AWS region**

**ec2\_instance\_type="t2.micro" # EC2 instance type**

**ec2\_ami\_id="ami-0c55b159cbfafe1f0" # Replace with your desired AMI ID**

**key\_name="my-key-pair" # Replace with your key pair name for SSH access**

**# Execute the resource creation script and capture output**

**resource\_output=$(./create\_resources.sh)**

**if [ $? -ne 0 ]; then**

**echo "Failed to create resources. Exiting."**

**exit 1**

**fi**

**# Extract Security Group ID and IAM Role Name from the output**

**sg\_id=$(echo "$resource\_output" | grep "Security Group ID" | awk '{print $4}')**

**iam\_role=$(echo "$resource\_output" | grep "IAM Role Name" | awk '{print $4}')**

**# Create EC2 instance**

**echo "Creating EC2 instance in region $aws\_region..."**

**instance\_id=$(aws ec2 run-instances \**

**--region $aws\_region \**

**--image-id $ec2\_ami\_id \**

**--instance-type $ec2\_instance\_type \**

**--security-group-ids $sg\_id \**

**--iam-instance-profile Name=$iam\_role \**

**--key-name $key\_name \**

**--query 'Instances[0].InstanceId' \**

**--output text)**

**if [ $? -eq 0 ]; then**

**echo "EC2 instance created with ID: $instance\_id"**

**else**

**echo "Failed to create EC2 instance."**

**exit 1**

**fi**

**echo "Waiting for EC2 instance to be in running state..."**

**aws ec2 wait instance-running --region $aws\_region --instance-ids $instance\_id**

**if [ $? -eq 0 ]; then**

**echo "EC2 instance is running."**

**else**

**echo "Failed to wait for EC2 instance to be running."**

**exit 1**

**fi**

**echo "EC2 instance setup complete."**

### **Explanation**

1. **create\_resources.sh Script:**
   * **Creates the security group and IAM role.**
   * **Outputs the IDs and names of the created resources.**
2. **create\_ec2\_instance.sh Script:**
   * **Calls create\_resources.sh to create the necessary resources and capture its output.**
   * **Extracts the Security Group ID and IAM Role Name from the output.**
   * **Uses these values to create an EC2 instance.**