

Day 7: Live AWS Project Using Shell Scripting for DevOps

Hello everyone, my name is Abhishek, and welcome back to my channel! Today is Day 7 of our full DevOps course, and in this class, we will be discussing a real-time shell script project that many DevOps engineers use in cloud infrastructure.

Why Move to Cloud Infrastructure?

Before we dive into the project, let's discuss why organizations choose to move to cloud infrastructure, such as AWS or Azure. There are two primary reasons:

1. **Manageability:** Managing physical servers requires significant effort. If you are a startup, you might need to build and maintain your own data center. This includes:
 - Regularly patching servers for security issues.
 - Upgrading hardware and software.
 - Managing backups and disaster recovery.

By moving to the cloud, organizations can reduce the overhead of managing physical servers because cloud providers handle much of this maintenance.

2. **Cost-Effectiveness:** Cloud providers operate on a "pay-as-you-go" model. This means you only pay for the resources you use. In contrast, if you purchase physical infrastructure, you pay for it regardless of whether you use it. This flexibility allows organizations to scale their resources according to their needs.

Tracking Resource Usage

As a DevOps engineer, one of your key responsibilities is to maintain cost-effectiveness in cloud usage. For example, if you work for an organization like `example.com`, you might have many developers creating resources in AWS.

Problem Scenario

Imagine Developer X creates 100 EC2 instances, but they are not actively using them. Similarly, they might create EBS volumes that are not attached to any instances. Even though these resources are not being used, AWS still charges for them. Therefore, tracking res

⊕ Ask follow-up

• ☒ Pro ↑

<https://chatgpt.com/c/a08a62b9-e077-490b-b33a-e050f41872e7>

Problem Scenario

Imagine Developer X creates 100 EC2 instances, but they are not actively using them. Similarly, they might create EBS volumes that are not attached to any instances. Even though these resources are not being used, AWS still charges for them. Therefore, tracking resource usage is essential to avoid unnecessary costs.

Approaches to Track Resource Usage

There are various ways to track resource usage in AWS:

- **AWS Management Console:** Manually check resources.
- **AWS CLI:** Use command-line tools to query resource usage.
- **CloudWatch:** Monitor and set alarms for resource usage.
- **Custom Scripts:** Automate the tracking process using shell scripts.

Shell Scripting Project Overview

In this session, we will create a shell script to monitor specific AWS resources like EC2 instances, S3 buckets, and Lambda functions. This script will help identify unused resources and provide insights into cost management.

Example Shell Script

Here's a simplified version of what our shell script might look like:

bash

 Explain 

```
#!/bin/bash
```



```
# AWS CLI must be configured with appropriate permissions
```

```
# Check for unused EC2 instances
```

```
echo "Checking for unused EC2 instances..."
```

```
unused_instances=$(aws ec2 describe-instances --filters  
"Name=instance-state-name,Values=stopped" --query
```



Ask follow-up



Pro



```
echo "No unused EC2 instances found."
```

```
else
```

bash

 Explain 

```
#!/bin/bash
```



```
# AWS CLI must be configured with appropriate permissions
```

```
# Check for unused EC2 instances
```

```
echo "Checking for unused EC2 instances ..."
```

```
unused_instances=$(aws ec2 describe-instances --filters  
"Name=instance-state-name,Values=stopped" --query  
"Reservations[*].Instances[*].InstanceId" --output text)
```

```
if [ -z "$unused_instances" ]; then
```

```
    echo "No unused EC2 instances found."
```

```
else
```

```
    echo "Unused EC2 instances: $unused_instances"
```

```
fi
```

```
# Check for unused EBS volumes
```

```
echo "Checking for unused EBS volumes ..."
```

```
unused_volumes=$(aws ec2 describe-volumes --filters  
"Name=status,Values=available" --query "Volumes[*].VolumeId" --  
output text)
```

```
if [ -z "$unused_volumes" ]; then
```

```
    echo "No unused EBS volumes found."
```

```
else
```

```
    echo "Unused EBS volumes: $unused_volumes"
```

```
fi
```

```
# Check for S3 buckets
```

```
echo "Checking for S3 buckets ..."
```

```
s3_buckets=$(aws s3api list-buckets --query "Buckets[*].Name" --  
-output text)
```



Ask follow-up



Pro



```
echo "Checking for Lambda functions ..."
```

```
lambda_functions=$(aws lambda list-functions --query
```

```
fi

# Check for S3 buckets
echo "Checking for S3 buckets ..."
s3_buckets=$(aws s3api list-buckets --query "Buckets[*].Name" -
-output text)

echo "S3 Buckets: $s3_buckets"

# Check for Lambda functions
echo "Checking for Lambda functions ..."
lambda_functions=$(aws lambda list-functions --query
"Functions[*].FunctionName" --output text)

echo "Lambda Functions: $lambda_functions"

# Additional logic can be added to notify or delete unused
resources
```

Explanation of the Script

- **AWS CLI:** The script uses AWS CLI commands to check for unused resources.
- **EC2 Instances:** It checks for stopped instances, which are not currently in use.
- **EBS Volumes:** It checks for available volumes that are not attached to any instances.
- **S3 Buckets and Lambda Functions:** It lists all S3 buckets and Lambda functions.

Conclusion

In conclusion, understanding how to manage cloud resources effectively is crucial for any DevOps engineer. By automating the tracking of resource usage with shell scripts, you can help your organization save costs and optimize resource allocation. This project is just one example of how shell scripting can be applied in real-world scenarios.

If you have any questions or need further clarification on any topic, feel free to ask in the comments! Thank you for joining me today, and I look forward to seeing you in the next session!

[Share](#) [Rewrite](#)



Introduction

"Hello everyone, my name is Abhishek and welcome back to my channel. Today is day 7 of our full DevOps course. In this class, we'll talk about a real-time shell script project that DevOps engineers commonly use in cloud infrastructure."

Purpose of the Project

"First, I'll explain what this project is and why it's useful. Then, we'll look at how to create it. So, why would someone move to cloud infrastructure, like AWS or Azure? There are two main reasons:

1. Manageability:

- If you're a startup maintaining your own servers, it involves a lot of maintenance. You need to create and manage a data center, patch security issues, and constantly update servers.
- This requires a dedicated systems engineering team, which can be a hassle.

2. Cost:

- Cloud providers operate on a 'pay as you go' basis. If you don't use certain instances, you don't get billed for them.
- In contrast, with physical infrastructure, you have to pay for it whether you use it or not. So, moving to the cloud helps reduce maintenance overhead and is cost-effective."

Managing Cloud Costs

"To be cost-effective, an organization needs to monitor resource usage. For example, if example.com has 100 developers with access to AWS, they might create many resources.

- If a developer creates 100 EC2 instances but no one uses them, they are wasting money.
- Similarly, unused EBS volumes still incur costs.

As a DevOps engineer or AWS admin, one of your main responsibilities is to maintain cost-effectiveness by tracking resource usage."



Tracking Resource Usage

Managing Cloud Costs

"To be cost-effective, an organization needs to monitor resource usage. For example, if example.com has 100 developers with access to AWS, they might create many resources.

- If a developer creates 100 EC2 instances but no one uses them, they are wasting money.
- Similarly, unused EBS volumes still incur costs.

As a DevOps engineer or AWS admin, one of your main responsibilities is to maintain cost-effectiveness by tracking resource usage."

Tracking Resource Usage

"There are multiple ways to track resource usage. While this shell script I'm showing you is not the only way, it's a practical method. People also use Lambda functions and Python scripts for this purpose. The goal is to achieve effective resource tracking, regardless of the method."

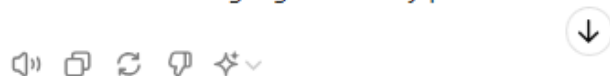
Example Project

"In today's topic, we'll assume there's an organization called example.com using resources like EC2, S3, Lambda functions, and IAM. We'll focus on monitoring these resources because they are commonly used across different organizations."

Daily Monitoring

"Our goal is to run a script every day at 6 pm to monitor the usage of these resources."

This breakdown highlights the key points and makes the information more digestible.





every day at 6 pm, your shell script will run automatically to check the usage of these AWS resources. The goal is to find any unused or underutilized resources and report them so that they can be managed or terminated to save costs. Here's how we can go about creating this project step by step:

1. Set Up AWS CLI

Ensure that the AWS CLI is installed and configured on the machine where the script will run. You can install the AWS CLI by following the instructions [here](#).

2. Create the Shell Script

Let's start by creating a basic shell script named `resource_monitor.sh`:

```
bash Copy code

#!/bin/bash

# Enable debug mode
set -x

# AWS profile to use (make sure this is configured)
AWS_PROFILE=default

# Function to check unused EC2 instances
check_ec2_instances() {
    echo "Checking EC2 instances..."
    ec2_instances=$(aws ec2 describe-instances --query 'Reservations[*].Instances[*].[InstanceId, State.Name, Tags[*].Name]' --output text)

    echo "InstanceId    State    Name"
    echo "$ec2_instances"

    # Check for instances that are stopped
    stopped_instances=$(echo "$ec2_instances" | awk '$2 == "stopped" { print $1 }')
    if [[ -n $stopped_instances ]]; then
        echo "Stopped EC2 instances found:"
    fi
}
```



Message ChatGPT



```

check_ec2_instances() {
    echo "Checking EC2 instances..."
    ec2_instances=$(aws ec2 describe-instances --query 'Reservations[*].Instances[*].[InstanceId,State,Name]' --text)

    echo "InstanceId    State    Name"
    echo "$ec2_instances"

    # Check for instances that are stopped
    stopped_instances=$(echo "$ec2_instances" | awk '$2 == "stopped" { print $1 }')
    if [[ -n $stopped_instances ]]; then
        echo "Stopped EC2 instances found:"
        echo "$stopped_instances"
    else
        echo "No stopped EC2 instances found."
    fi
}

# Function to check unused EBS volumes
check_ebs_volumes() {
    echo "Checking EBS volumes..."
    ebs_volumes=$(aws ec2 describe-volumes --query 'Volumes[*].[VolumeId,State,Attachments]' --text)

    echo "VolumeId    State    Attachments"
    echo "$ebs_volumes"

    # Check for volumes that are available (unused)
    available_volumes=$(echo "$ebs_volumes" | awk '$2 == "available" { print $1 }')
    if [[ -n $available_volumes ]]; then
        echo "Unused EBS volumes found:"
        echo "$available_volumes"
    else
        echo "No unused EBS volumes found."
    fi
}

```




```

# Function to check S3 buckets
check_s3_buckets() {
    echo "Checking S3 buckets..."
    s3_buckets=$(aws s3api list-buckets --query 'Buckets[*].[Name,CreationDate]' --output te

    echo "BucketName    CreationDate"
    echo "$s3_buckets"

    # This script does not check the usage of S3 buckets as it can be complex and requires m
}

# Function to check Lambda functions
check_lambda_functions() {
    echo "Checking Lambda functions..."
    lambda_functions=$(aws lambda list-functions --query 'Functions[*].[FunctionName,LastMod

    echo "FunctionName    LastModified"
    echo "$lambda_functions"

    # This script does not check the usage of Lambda functions as it can be complex and requ
}

# Function to check IAM users
check_iam_users() {
    echo "Checking IAM users..."
    iam_users=$(aws iam list-users --query 'Users[*].[UserName,CreateDate]' --output text --

    echo "UserName    CreateDate"
    echo "$iam_users"

    # This script does not check the activity of IAM users as it can be complex and requires
}

# Main function to call other functions

```



```
# Function to check IAM users
check_iam_users() {
    echo "Checking IAM users..."
    iam_users=$(aws iam list-users --query 'Users[*].[UserName,CreateDate]' --output text --)

    echo "UserName    CreateDate"
    echo "$iam_users"

    # This script does not check the activity of IAM users as it can be complex and requires
}


# Main function to call other functions
main() {
    check_ec2_instances
    check_ebs_volumes
    check_s3_buckets
    check_lambda_functions
    check_iam_users
}

# Run the main function
main
```

3. Make the Script Executable

Ensure the script is executable by running:

```
bash
```

 Copy code

```
chmod +x resource_monitor.sh
```

4. Schedule the Script Using Cron



4. Schedule the Script Using Cron

To automate this script to run every day at 6 pm, add a cron job:

```
bash
```

[Copy code](#)

```
crontab -e
```

Then add the following line to the crontab file:

```
bash
```

[Copy code](#)

```
0 18 * * * /path/to/resource_monitor.sh
```

5. Output the Results

Modify the script to log results to a file or send an email notification if required. Here's an example of how to log to a file:

```
bash
```

[Copy code](#)

```
LOG_FILE="/var/log/resource_monitor.log"
exec > >(tee -a $LOG_FILE) 2>&1
```

Add this line at the beginning of your script to log both stdout and stderr to the log file.

Summary

This script will check for unused EC2 instances and EBS volumes. You can extend it to check for other resources or refine the logic to better suit your needs. By scheduling it with cron, it will run automatically at the specified time, helping you keep track of resource usage and manage costs effectively.



Proper Code which hide in image:

```
1  #!/bin/bash
2
3  # Enable debug mode
4  ```bash
5  set -x
6  ```
7  # AWS profile to use (make sure this is configured)
8  ```bash
9  AWS_PROFILE=default
10 ```
```

```
1  # Function to check unused EC2 instances
2  ```bash
3  check_ec2_instances() {
4  echo "Checking EC2 instances..."
5  ec2_instances=$(aws ec2 describe-instances --query 'Reservations[*].Instances[*].[InstanceId,State.Name,Tags[?Key=='Name''].Value]' --output text --profile $AWS_PROFILE)
6
7  echo "InstanceId    State    Name"
8  echo "$ec2_instances"
9
10 # Check for instances that are stopped
11 stopped_instances=$(echo "$ec2_instances" | awk '$2 == "stopped" { print $1 }')
12 if [[ -n $stopped_instances ]]; then
13 echo "Stopped EC2 instances found:"
14 echo "$stopped_instances"
15 else
16 echo "No stopped EC2 instances found."
17 fi
18 }
19 ```
```

```

1  '''
2  # Function to check unused EBS volumes
3  '''bash
4  check_ebs_volumes() {
5      echo "Checking EBS volumes..."
6      ebs_volumes=$(aws ec2 describe-volumes --query 'Volumes[*].[VolumeId,State,Attachments]' --output text --profile $AWS_PROFILE)
7
8      echo "VolumeId    State    Attachments"
9      echo "$ebs_volumes"
10
11     # Check for volumes that are available (unused)
12     available_volumes=$(echo "$ebs_volumes" | awk '$2 == "available" { print $1 }')
13     if [[ -n $available_volumes ]]; then
14         echo "Unused EBS volumes found:"
15         echo "$available_volumes"
16     else
17         echo "No unused EBS volumes found."
18     fi
19 }
20 '''

```

```

1  '''
2  # Function to check S3 buckets
3  '''bash
4  check_s3_buckets() {
5      echo "Checking S3 buckets..."
6      s3_buckets=$(aws s3api list-buckets --query 'Buckets[*].[Name,CreationDate]' --output text --profile $AWS_PROFILE)
7
8      echo "BucketName    CreationDate"
9      echo "$s3_buckets"
10
11     # This script does not check the usage of S3 buckets as it can be complex and requires more specific logic
12 }
13 '''

```

```

1  # Function to check Lambda functions
2  '''bash
3  check_lambda_functions() {
4      echo "Checking Lambda functions..."
5      lambda_functions=$(aws lambda list-functions --query 'Functions[*].[FunctionName,LastModified]' --output text --profile $AWS_PROFILE)
6
7      echo "FunctionName    LastModified"
8      echo "$lambda_functions"
9
10     # This script does not check the usage of Lambda functions as it can be complex and requires more specific logic
11 }
12 '''

```

```
1  # Function to check IAM users
2  ```bash
3  check_iam_users() {
4      echo "Checking IAM users..."
5      iam_users=$(aws iam list-users --query 'Users[*].[UserName,CreateDate]' --output text --profile $AWS_PROFILE)
6
7      echo "UserName    CreateDate"
8      echo "$iam_users"
9
10     # This script does not check the activity of IAM users as it can be complex and requires more specific logic
11 }
```

```
1  # Main function to call other functions
2  ```bash
3  main() {
4      check_ec2_instances
5      check_ebs_volumes
6      check_s3_buckets
7      check_lambda_functions
8      check_iam_users
9  }
10  ```
11  # Run the main function
12  ```bash
13  main
14  ```
```

Generating and Reporting Resource Usage:

Answer

Automating AWS Resource Reporting with Shell Scripting

In this project, we will create a shell script that generates a daily report on the usage of AWS resources for your organization. This script will help track and manage the resources used by different teams or developers.

Why Automate Resource Reporting?

As a DevOps engineer, one of your responsibilities is to ensure cost-effectiveness in your organization's cloud infrastructure. When multiple teams or developers are using AWS resources, it's crucial to monitor their usage to avoid unnecessary costs.

Typically, organizations use reporting dashboards to track resource usage. However, for the purpose of this shell scripting project, we will generate the report and send it to your manager.

Creating the Shell Script

1. **Gather AWS Resource Information:** We will use the AWS CLI to collect information about various AWS resources, such as:
 - EC2 instances
 - S3 buckets
 - Lambda functions
 - IAM users
2. **Generate a Report File:** The shell script will create a file containing the collected resource usage information.
3. **Automate Report Generation:** To ensure that the report is generated and sent to your manager every day at a specific time (e.g., 6 PM), we will use a Cron job.

What is a Cron Job?

A Cron job is a scheduled task in Linux that allows you to run scripts or commands at

 Ask follow-up

 Pro 

For example, if you are creating a DevOps Zero to Hero course on YouTube, you can schedule your videos to be published at a specific time each day using a Cron job.

What is a Cron Job?

A Cron job is a scheduled task in Linux that allows you to run scripts or commands at specified intervals. This is useful for automating repetitive tasks without manual intervention.

For example, if you are creating a DevOps Zero to Hero course on YouTube, you can schedule your videos to be published at a specific time each day using a Cron job. YouTube will publish the video at the scheduled time, so you don't need to log in and manually publish it yourself.

Similarly, in our case, we will create a Cron job that will execute the shell script at 6 PM every day, generating the AWS resource usage report and sending it to your manager.

Benefits of Automating Resource Reporting

- **Consistent Reporting:** By scheduling the report generation using a Cron job, you ensure that the report is generated and sent to your manager at the same time every day, regardless of your availability.
- **Timely Notifications:** Your manager will receive the report at the scheduled time, allowing them to stay informed about resource usage and make informed decisions.
- **Reduced Manual Effort:** Automating the reporting process saves time and effort, as you don't need to manually generate and send the report every day.

Conclusion

In this project, we will combine our knowledge of shell scripting and the AWS CLI to create an automated AWS resource usage reporting system. By integrating the shell script with a Cron job, we will ensure that the report is generated and sent to your manager at a specific time every day.

This project is a practical application of shell scripting skills and can be included in your resume, as it demonstrates your ability to automate tasks and manage cloud resources effectively.

🔗 Share 🔄 Rewrite

📌 📝 ...

👤 Ask follow-up

👤 Pro 📌



Explanation

In this part of the script, Abhishek is explaining how to generate a report on AWS resource usage and automate the process using a shell script and Cron jobs. Here's a breakdown in simple terms:

Generating and Reporting Resource Usage

Daily Reporting:

- Every day, at a specific time, you need to provide a report to your manager about the usage of AWS resources like EC2 instances, S3 buckets, Lambda functions, and IAM users.

Automating the Report:

- While usually, this data is sent to a reporting dashboard automatically, for learning purposes, we assume you need to provide this information directly to your manager.

Creating a Shell Script:

- You will write a shell script that gathers all the resource usage information.
- The script will create a file containing details such as the number of active EC2 instances, S3 buckets, Lambda functions, and IAM users.

Scheduling the Script with Cron Jobs:

- Manually running the script every day can be unreliable if you're unavailable.
- Instead, use a Cron job to automate this task. A Cron job is a time-based scheduler in Unix-like operating systems that runs scripts or commands at specified times.

Understanding Cron Jobs



What is a Cron Job?



Message ChatGPT



Understanding Cron Jobs

What is a Cron Job?

- A Cron job schedules tasks to run automatically at specific times or intervals.
- For example, if you upload a video to YouTube and schedule it to publish at 7 pm, YouTube handles the publishing for you at the set time.

Using Cron Jobs:

- Similarly, you can schedule a Cron job to run your shell script every day at 6 pm.
- This ensures the script runs automatically, generating the report without you having to log in and run it manually.

Practical Application:

- In Linux, a Cron job can be set to execute the script at any given time.
- You can say, "I can make use of the Cron job in Linux to run this script daily at a specified time."

Combining Shell Script and AWS CLI

Using AWS CLI:

- You will use the AWS Command Line Interface (CLI) to gather information about your AWS resources.
- The script will combine your knowledge of shell scripting and AWS CLI to get the necessary data.

Summary of the Lesson

What You Will Learn:

- How to write a shell script that collects AWS resource usage information.
 - How to automate the execution of this script using Cron jobs.
 - This project is a common task in many organizations and can be added to your resume.
-

Summary of the Lesson

What You Will Learn:

- How to write a shell script that collects AWS resource usage information.
 - How to automate the execution of this script using Cron jobs.
 - This project is a common task in many organizations and can be added to your resume.
-

Practical Demonstration

Starting the Demonstration:

- Abhishek transitions to a practical demonstration by stopping the screen share and opening the terminal to start writing the script.
 - He checks if AWS CLI is installed as a prerequisite.
-

This explanation breaks down the key points and processes in a straightforward manner, making it easier to understand the purpose and steps involved in the project.



Daily Reporting to Management

Answer

Explanation of Key Concepts from the Text

Daily Reporting to Management

In many organizations, employees are required to generate and submit reports regularly, such as daily resource usage reports to their managers. These reports typically include information about cloud resources, such as:

- Active EC2 instances
- S3 buckets
- Lambda functions
- IAM users

Automation of Reporting

Instead of manually generating and sending these reports every day, which can be time-consuming and prone to errors, organizations often automate this process using scripts. Common scripting languages for this purpose include:

- **Shell Scripting:** A powerful way to automate tasks in Unix/Linux environments.
- **Python:** A versatile programming language that can also be used for automation tasks.

The generated report can be sent to a reporting dashboard or directly to the manager, depending on the organization's practices.

Using Shell Scripts

For the purpose of this project, we will focus on creating a shell script that collects the required information about AWS resources. The script will:

1. Gather data on active EC2 instances.
2. List all S3 buckets.
3. Check for Lambda functions.

Using Shell Scripts

For the purpose of this project, we will focus on creating a shell script that collects the required information about AWS resources. The script will:

1. Gather data on active EC2 instances.
2. List all S3 buckets.
3. Check for Lambda functions.
4. Compile this information into a report file.

Importance of Scheduled Reporting

To ensure that the report is generated and sent at a specific time (e.g., 6 PM), we can use a **Cron job**. A Cron job is a scheduled task in Linux that allows you to run scripts or commands automatically at specified intervals.

Example of Cron Job Functionality

- **Analogy:** Just as you can schedule a video to be published on YouTube at a specific time without needing to log in at that moment, a Cron job allows your Linux system to execute a script at a predetermined time without manual intervention.
- **Setting Up a Cron Job:**
 1. Open the crontab configuration:

```
bash
```

```
crontab -e
```



2. Add a new line to schedule the script:

```
bash
```

```
0 18 * * * /path/to/your/script.sh
```



2. Add a new line to schedule the script:

```
bash
0 18 * * * /path/to/your/script.sh
```

This line tells the system to run the script every day at 6 PM.

Benefits of Using Cron Jobs

- **Consistency:** Ensures that reports are generated and sent on time, regardless of whether you are available to run the script.
- **Efficiency:** Reduces the need for manual intervention, allowing you to focus on other important tasks.

AWS CLI Configuration

Before running the script, it is essential to have the AWS CLI installed and configured. This involves:

1. Running the command:

```
bash
aws configure
```

2. Providing your AWS access key, secret access key, default region, and output format (e.g., JSON).

Conclusion

In this project, we will combine our knowledge of shell scripting and the AWS CLI to automate the reporting of AWS resource usage. By integrating the shell script with a Cron job, we can ensure that the report is generated and sent to your manager every day at a

Conclusion

In this project, we will combine our knowledge of shell scripting and the AWS CLI to automate the reporting of AWS resource usage. By integrating the shell script with a Cron job, we can ensure that the report is generated and sent to your manager every day at a specific time. This approach not only saves time but also helps maintain cost-effectiveness in managing cloud resources.

 Share  Rewrite



Setting Up the Environment



Detailed Explanation

In this section, Abhishek explains how to set up the environment and write a shell script to track AWS resource usage. Here's a detailed breakdown:

Setting Up the Environment

1. AWS CLI Check:

- Abhishek ensures that the AWS Command Line Interface (CLI) is installed on his Mac. He also plans to connect to an EC2 instance to demonstrate the script on a Linux environment for consistency.

2. Choosing the Shell:

- He recommends using `bash` because it is widely used and most Linux distributions come with it by default. He mentions that some distributions might use `dash`, which has slight syntax differences from `bash`.

Configuring AWS CLI

1. AWS CLI Installation:

- Check if the AWS CLI is installed by running a command like `aws --version`.

2. AWS CLI Configuration:

- Configure the AWS CLI using the `aws configure` command. This process involves providing:
 - Access Key ID
 - Secret Access Key



2. AWS CLI Configuration:

- Configure the AWS CLI using the `aws configure` command. This process involves providing:
 - Access Key ID
 - Secret Access Key
 - Default Region Name
 - Default Output Format (e.g., JSON)

3. Authentication:

- Once configured, the CLI can authenticate and communicate with the AWS account.

Writing the Shell Script

1. Script Initialization:

- Open a text editor like `vim` to create the script file. For example, `vim AWS_resource_tracker.sh`.

2. Shebang:

- Start the script with the shebang line `#!/bin/bash` to specify that the script should be run with `bash`.

3. Script Metadata:

- Add comments at the beginning of the script to provide metadata:
 - **Author:** Your name (e.g., Abhishek).
 - **Date:** When you started writing the script (e.g., 11th of Jan).
 - **Version:** Initial version or draft (e.g., V1).
 - **Description:** Brief description of what the script does (e.g., "This script will report the AWS resource usage").



Writing the Script Body

1. Comments and Simplicity:


- Add comments to explain the purpose of each section of the script.
- For simplicity, avoid using shell functions and keep the script straightforward, considering the audience might be beginners.

2. Objects to Track:

- List the AWS resources you want to track, such as EC2 instances, S3 buckets, Lambda functions, and IAM users.

Example Script Outline

bash

 Copy code

```
#!/bin/bash

# Author: Abhishek
# Date: 11th of Jan
# Version: V1
# Description: This script will report the AWS resource usage.

# List of objects to track
# - EC2 Instances
# - S3 Buckets
# - Lambda Functions
# - IAM Users

# Check if AWS CLI is installed and configured
if ! command -v aws &> /dev/null
then
    echo "AWS CLI is not installed. Exiting..."
    exit 1
fi
```

```

# Check if AWS CLI is installed and configured
if ! command -v aws &> /dev/null
then
    echo "AWS CLI is not installed. Exiting..."
    exit 1
fi

# Function to check if AWS CLI is configured
aws configure list &> /dev/null
if [ $? -ne 0 ]; then
    echo "AWS CLI is not configured. Please run 'aws configure' to set it up."
    exit 1
fi

# Create a report file
REPORT_FILE="aws_resource_report.txt"
echo "AWS Resource Usage Report" > $REPORT_FILE
echo "Generated on: $(date)" >> $REPORT_FILE
echo "" >> $REPORT_FILE

# Fetch and report EC2 instances
echo "EC2 Instances:" >> $REPORT_FILE
aws ec2 describe-instances --query 'Reservations[*].Instances[*].[InstanceId,State.Name,In
echo "" >> $REPORT_FILE

# Fetch and report S3 buckets
echo "S3 Buckets:" >> $REPORT_FILE
aws s3 ls >> $REPORT_FILE
echo "" >> $REPORT_FILE

# Fetch and report Lambda functions
echo "Lambda Functions:" >> $REPORT_FILE
aws lambda list-functions --query 'Functions[*].[FunctionName,Runtime]' --output table >>
echo "" >> $REPORT_FILE

```

```
# Fetch and report Lambda functions
echo "Lambda Functions:" >> $REPORT_FILE
aws lambda list-functions --query 'Functions[*].[FunctionName,Runtime]' --output table >>
echo "" >> $REPORT_FILE

# Fetch and report IAM users
echo "IAM Users:" >> $REPORT_FILE
aws iam list-users --query 'Users[*].[UserName,CreateDate]' --output table >> $REPORT_FILE

# Notify completion
echo "Report generation complete. Check the file: $REPORT_FILE"
```


Running the Script

1. Make the Script Executable:
 - Run `chmod +x AWS_resource_tracker.sh` to make the script executable.
2. Execute the Script:
 - Run the script using `./AWS_resource_tracker.sh`.

Automating with Cron Jobs

1. Cron Job Setup:
 - Schedule the script to run daily at 6 PM using a Cron job. Edit the cron table with `crontab -e` and add the following line:

bash

 Copy code

```
0 18 * * * /path/to/AWS_resource_tracker.sh
```

- This line schedules the script to run at 6 PM every day.



Conclusion

Automating with Cron Jobs

1. Cron Job Setup:

- Schedule the script to run daily at 6 PM using a Cron job. Edit the cron table with `crontab -e` and add the following line:

```
bash
0 18 * * * /path/to/AWS_resource_tracker.sh
```

Copy code

- This line schedules the script to run at 6 PM every day.

Conclusion

This explanation covers the setup, writing, and automation of a shell script to track AWS resource usage, providing a detailed and deep understanding of each step.

What is Cron job and what is the purpose and why we need?



Claude 3 Haiku

Okay, let me explain Cron jobs in a simple way.

Imagine you have a lot of tasks that you need to do on your computer, like:

- Backing up your files every night
- Sending out a newsletter every month
- Checking for software updates every week

Instead of having to remember to do these tasks yourself, you can set up something called a "Cron job" to do them automatically for you.

A Cron job is a special program in your computer that runs tasks at specific times or intervals that you set. It's like an alarm clock for your computer – you tell it when to do certain things, and it will do them without you having to remember.

The purpose of Cron jobs is to help automate and streamline the tasks you need to do regularly. This saves you time and effort, and also helps make sure these important tasks get done on time, every time.

Here are some of the main reasons why Cron jobs are useful:

1. Automation: Cron jobs let you automate repetitive tasks so you don't have to do them

Here are some of the main reasons why Cron jobs are useful:

1. **Automation:** Cron jobs let you automate repetitive tasks so you don't have to do them manually.
2. **Efficiency:** By automating tasks, Cron jobs help you work more efficiently and reduce the chance of forgetting to do something important.
3. **Consistency:** Cron jobs ensure that tasks are completed on a regular schedule, even if you forget to do them yourself.
4. **Resource Management:** Cron jobs can run tasks when your computer has more resources available, instead of running them constantly.

So in summary, Cron jobs are a powerful tool that let you schedule and automate important tasks on your computer. They save you time and effort, and help make sure important things get done on a consistent schedule.

✦ Get smarter answer from GPT-4o | ▾

📄 ↻ 🗨️ 🔊



GPT-4o mini

A **Cron job** is a scheduled task in Unix-based operating systems (like Linux and macOS) that automatically runs scripts or commands at specified times or intervals.

What is it?

1. **Scheduled Task:** Think of a Cron job as an alarm clock for your computer. Just as you set an alarm to wake up at a specific time, you can set a Cron job to perform tasks at specific times.
2. **Command Line Tool:** Cron is a built-in service that runs in the background of your operating system. It reads a special file (called a "crontab") where you list the tasks you want to run and their schedule.

Purpose and Why We Need It

1. **Automation:** Cron jobs help automate repetitive tasks. For instance, you might want to back up your files every night or send out reminder emails every morning. Instead of doing these things manually, you can set up Cron jobs to do them automatically.
2. **Efficiency:** By automating routine tasks, you save time and reduce the chance of human error. This means your tasks get done consistently, allowing you to focus on more

-
2. **Efficiency:** By automating routine tasks, you save time and reduce the chance of human error. This means your tasks get done consistently, allowing you to focus on more important work.
 3. **Consistency:** Some tasks need to be done regularly (e.g., daily reports, database backups). Cron jobs ensure these tasks are completed on time without having to remember to do them.
 4. **Resource Management:** Instead of running processes constantly, you can run them when you need them, which can save system resources, like CPU and memory.

Example Scenarios:

- **Daily Backup:** Set a Cron job to back up your website files every night at 2 AM.
- **Regular Updates:** Schedule a task to check for software updates every week.
- **Sending Emails:** Automatically send a newsletter every month on the first Monday.

How It Works:

You define a Cron job using a simple format that specifies:

- **Minute** (0–59)

How It Works:

You define a Cron job using a simple format that specifies:

- **Minute** (0–59)
- **Hour** (0–23)
- **Day of the Month** (1–31)
- **Month** (1–12)
- **Day of the Week** (0–7 where both 0 and 7 represent Sunday)

For example, the following entry means “run the script every day at 5 PM”:

```
0 17 * * * /path/to/your/script.sh
```

In summary, Cron jobs are helpful for automating tasks you want to occur regularly, making it easier to manage time and resources effectively.



What is a Cron Job?

A Cron job is a scheduled task on Unix-like operating systems that automatically runs scripts or commands at specified intervals. The "Cron" system uses a special syntax to define when tasks should run.

Purpose of a Cron Job

The primary purpose of a Cron job is to automate repetitive tasks, saving time and reducing the chance of human error. Here are some common uses:

1. **Regular Backups:** Automatically back up important files or databases at regular intervals (e.g., daily, weekly).
2. **System Maintenance:** Perform routine maintenance tasks like cleaning up temporary files or logs.
3. **Report Generation:** Automatically generate and send reports at specific times.
4. **Monitoring and Alerts:** Run health checks on systems and send alerts if something goes wrong.

Why Do We Need Cron Jobs?

Cron jobs are essential because they help ensure tasks are performed consistently and on time without manual intervention. Here's why they are beneficial:

1. **Automation:** Eliminates the need to manually run tasks, saving time and effort.
2. **Consistency:** Ensures tasks are executed at precise intervals, reducing the chance of forgetting or making mistakes.
3. **Efficiency:** Frees up human resources to focus on more critical or complex tasks.
4. **Reliability:** Automates repetitive tasks, making sure they run even when you're not available (e.g., during weekends or holidays).

How Cron Jobs Work



Cron jobs are managed using the `crontab` file, which contains a list of commands to be executed at

How Cron Jobs Work

Cron jobs are managed using the `crontab` file, which contains a list of commands to be executed at specified times. The syntax for scheduling tasks in the `crontab` file is:

```
SCSS Copy code
* * * * * command_to_run
| | | | |
| | | | | Day of the week (0-7) (0 and 7 represent Sunday)
| | | | | Month (1-12)
| | | | | Day of the month (1-31)
| | | | | Hour (0-23)
| | | | | Minute (0-59)
```

Example

If you want to run a script every day at 6 PM, you would add the following line to your `crontab` file:

```
javascript Copy code
0 18 * * * /path/to/your_script.sh
```

- `0` = Minute (0th minute)
- `18` = Hour (6 PM in 24-hour format)
- `*` = Every day of the month
- `*` = Every month
- `*` = Every day of the week

This tells the Cron system to run the script located at `/path/to/your_script.sh` every day at 6 PM.

Summary



Summary

Cron jobs are powerful tools for automating repetitive tasks, ensuring they are performed consistently and on time. They are widely used in system administration, DevOps, and development environments to maintain system health, perform routine maintenance, generate reports, and more.

Overview or summary of above concepts:



Detailed Explanation of the Shell Script Project

This content breaks down and explains a shell script project aimed at automating the tracking of cloud resources, specifically within AWS, and generating reports that could be used for cost management and optimization. Let's walk through the key concepts and ideas presented, and I'll explain each in detail:

1. Introduction to Cloud Infrastructure and DevOps Needs

- **Cloud Infrastructure:** Companies move to cloud platforms like AWS or Azure to reduce the overhead associated with managing physical servers. Cloud platforms handle maintenance, security patches, and upgrades, so you don't need a dedicated team to manage these tasks. Additionally, cloud platforms operate on a **pay-as-you-go** model, meaning you only pay for the resources you use, unlike physical infrastructure where you have to invest in hardware whether or not it's being used.
 - **Cost Management:** One of the critical roles of a DevOps engineer or cloud administrator is to ensure that the cloud resources are being used efficiently. Without careful monitoring, unused resources can accumulate and incur unnecessary costs. For instance, an EC2 instance (a virtual server) might be running even though it's not being used, or an EBS volume (a virtual hard drive) might exist without being attached to an instance.
-


2. Project Overview

- **Objective:** The project focuses on creating a shell script that tracks the usage of certain AWS resources (e.g., EC2 instances, S3 buckets, Lambda functions, IAM users) and generates a report
-

2. Project Overview

- **Objective:** The project focuses on creating a shell script that tracks the usage of certain AWS resources (e.g., EC2 instances, S3 buckets, Lambda functions, IAM users) and generates a report summarizing the usage.
 - **Daily Reporting:** The script is designed to run daily and generate a report that can be reviewed by a manager or sent to a reporting dashboard. This report will help in identifying unused resources, which can then be decommissioned to save costs.
 - **Automation via Cron Job:** To automate the execution of the script at a specific time each day (e.g., 6 PM), the script will be scheduled using a **Cron job**. Cron jobs are scheduled tasks in Linux that can run commands or scripts at specified times and intervals. This ensures that the report is generated and sent without manual intervention.
-

3. Writing the Shell Script

- **Environment Setup:**
 - **Bash Shell:** The script is written in the Bash shell (`#!/bin/bash`), which is widely used in Linux environments. It's important to ensure that the script runs in a compatible environment to avoid issues with syntax differences that might arise if a different shell, like `dash`, is used.
 - **AWS CLI Configuration:** The AWS Command Line Interface (CLI) needs to be installed and configured with your AWS credentials. This allows the script to interact with AWS services using commands like `aws ec2 describe-instances` to list EC2 instances.
- **Script Metadata:**
 - The script includes comments at the  beginning to describe its purpose, the author's name, the creation date, and a brief description of what the script does. This information is useful

- **Script Metadata:**
 - The script includes comments at the beginning to describe its purpose, the author's name, the creation date, and a brief description of what the script does. This information is useful for others who might review or modify the script in the future.
-

4. Script Logic Breakdown

- **Resource Tracking:**
 - The script will track multiple AWS resources, such as EC2 instances, S3 buckets, Lambda functions, and IAM users.
- **Example Commands:**

- To list EC2 instances:

```
bash
```

[Copy code](#)

```
aws ec2 describe-instances
```

- To list S3 buckets:

```
bash
```

[Copy code](#)

```
aws s3 ls
```

- **Output Handling:**
 - The script will output the collected data into a file that can be reviewed or sent to a reporting system.
- **Cron Job Setup:**



- **Cron Job Setup:**

- The script will be integrated with a Cron job to automate its execution at a specific time every day. The Cron job ensures that the script runs consistently without manual intervention.

- **Example Cron Job Entry:**

- To run the script daily at 6 PM:

```
bash
```

[Copy code](#)

```
0 18 * * * /path/to/script.sh
```

5. Practical Application

- **Real-World Usage:**

- The script is a practical tool that can be used in real-world DevOps scenarios, especially for startups or smaller organizations that need to manage their cloud costs effectively without investing in expensive monitoring tools.

- **Customization and Expansion:**

- While the example focuses on a few AWS services, the script can be expanded to include more services or integrate with more sophisticated reporting and alerting systems.

- **Portability and Simplicity:**

- The project demonstrates how shell scripting combined with AWS CLI can be a powerful and simple solution for managing cloud infrastructure, even without advanced programming knowledge.



5. Practical Application

- **Real-World Usage:**
 - The script is a practical tool that can be used in real-world DevOps scenarios, especially for startups or smaller organizations that need to manage their cloud costs effectively without investing in expensive monitoring tools.
 - **Customization and Expansion:**
 - While the example focuses on a few AWS services, the script can be expanded to include more services or integrate with more sophisticated reporting and alerting systems.
 - **Portability and Simplicity:**
 - The project demonstrates how shell scripting combined with AWS CLI can be a powerful and simple solution for managing cloud infrastructure, even without advanced programming knowledge.
-

Conclusion

This project provides a clear and straightforward approach to tracking AWS resources using a shell script. The integration of Cron jobs ensures automation, and the use of AWS CLI commands makes the script versatile and powerful. Although this example is relatively simple, it serves as a strong foundation that can be built upon for more complex cloud management tasks.

OR



This transcript dives into the practical use of a shell script project that many DevOps engineers use to monitor cloud infrastructure, particularly on AWS. I'll break down and explain each concept in detail to make it easier to understand:

1. Why Move to Cloud Infrastructure?

- **Manageability:** Managing physical servers on your own can be complex and resource-intensive. You need a dedicated team to handle data centers, server maintenance, security patches, and upgrades.
- **Cost Efficiency:** Cloud providers like AWS or Azure offer a pay-as-you-go model, meaning you only pay for the resources you actually use. This is more cost-effective compared to owning physical servers, where costs are incurred whether resources are fully utilized or not.

2. Challenges in Cloud Cost Management

- **Even though cloud providers charge based on usage, managing costs can become tricky.** For example, a developer might create multiple AWS resources (like EC2 instances, S3 buckets, or EBS volumes) and forget to terminate or delete them when no longer needed. These unused resources can still incur costs.
- **DevOps Responsibility:** One of the key responsibilities of a DevOps engineer or AWS admin is to monitor and optimize cloud resource usage to keep costs under control.

3. Resource Usage Monitoring

- **Goal:** The primary goal is to track and report on the usage of cloud resources daily. This ensures that any unnecessary or idle resources are identified and can be terminated or optimized to save costs.

4. Shell Script for Resource Tracking

- **Shell Scripting:** The speaker emphasizes using shell scripts combined with AWS CLI (Command

4. Shell Script for Resource Tracking

- **Shell Scripting:** The speaker emphasizes using shell scripts combined with AWS CLI (Command Line Interface) to track and report resource usage. Although Python or AWS SDK could be used, the focus here is on shell scripting because it's straightforward and accessible for those already familiar with it.
- **Resources to Track:** Common AWS resources include EC2 (Elastic Compute Cloud) instances, S3 (Simple Storage Service) buckets, Lambda functions, and IAM (Identity and Access Management) users.

5. Automation with Cron Jobs

- **Cron Job:** To automate the script execution, the speaker suggests using a Cron job. A Cron job in Linux is a time-based task scheduler that can run scripts or commands at specified intervals, such as daily at 6 PM. This ensures the resource usage report is generated automatically without manual intervention.
- **Example:** If you have a shell script that generates a resource usage report, you can set a Cron job to run this script every day at a specific time. This automation ensures consistency and reliability in reporting.

6. AWS CLI Setup

- **AWS CLI:** AWS CLI is a command-line tool that allows you to interact with AWS services using commands. Before running any AWS commands, you need to configure the CLI with your AWS credentials (Access Key, Secret Key, Default Region, Output Format).
- **Configuration:** You can configure the AWS CLI using the `aws configure` command, which prompts you to enter the necessary credentials. Once configured, you can use the CLI to query information about your AWS resources.



7. Shell Script Structure

7. Shell Script Structure

- **Shebang** (`#!/bin/bash`): The script starts with a shebang, which tells the system which interpreter to use for the script. In this case, it's Bash.
- **Documentation**: The script includes comments at the beginning to document the author, date, version, and purpose of the script. This practice helps in maintaining and understanding the script later.
- **Simplicity**: The speaker chooses to write the script without using shell functions, aiming for simplicity, as the audience may not be familiar with more advanced shell scripting concepts yet.

8. Script Workflow

- **Plan**: The speaker plans to write a script that tracks specific AWS resources (like EC2, S3, Lambda, IAM) and generates a report. This report could be either stored in a file or sent to a reporting dashboard, depending on the requirements.
- **Execution**: The script would be executed daily via a Cron job to ensure consistent monitoring and reporting.

9. Practical Example

- The speaker proceeds to demonstrate writing the actual shell script, starting with configuring AWS CLI and ensuring the necessary tools are available on the Linux environment.
- **Script Execution**: After setting up the environment, the speaker writes a simple shell script to gather information about AWS resources and prepares to automate its execution using a Cron job.

10. Learning Outcome

- By the end of the session, the audience should understand how to write a basic shell script to track AWS resources, configure it to run automatically using Cron jobs, and integrate it with AWS
-

10. Learning Outcome

- By the end of the session, the audience should understand how to write a basic shell script to track AWS resources, configure it to run automatically using Cron jobs, and integrate it with AWS CLI for cloud resource management. This kind of script is a practical tool that can be added to a resume as a relevant DevOps project experience.

Conclusion

This explanation provides a comprehensive understanding of the script's purpose, how it fits into a DevOps engineer's role, and the practical steps involved in creating and automating it. The goal is to use this knowledge to monitor and optimize cloud infrastructure, ensuring cost efficiency and effective resource management.



Write about script:



Writing the AWS Resource Tracker Shell Script

We'll break down the process step-by-step for creating an AWS Resource Tracker shell script that can be scheduled to run periodically using Cron jobs. This script will track various AWS resources and generate a report.

1. File Creation and Setup

Open a terminal and create a new script file named `aws_resource_tracker.sh` using a text editor like `vim`.

```
bash Copy code  
  
vim aws_resource_tracker.sh
```

Once inside `vim`, press `i` to enter insert mode. Start writing your script by specifying the interpreter and adding metadata.

2. Shebang and Metadata

Begin with the shebang to specify the script interpreter, followed by author information and script details.

```
bash Copy code  
  
#!/bin/bash  
  
# Author: Abhishek  
# Date: 11th Jan  
# Version: V1  
# Description: This script will report the AWS resource usage.
```

3. Defining the Purpose and Tracking Object

Add comments to define what resources the script will track. This helps in maintaining and

3. Defining the Purpose and Tracking Objects

Add comments to define what resources the script will track. This helps in maintaining and understanding the script.

```
bash Copy code

# This script tracks the following AWS resources:
# 1. EC2 Instances
# 2. S3 Buckets
# 3. Lambda Functions
# 4. IAM Users
```

4. AWS CLI Configuration Check

Ensure the AWS CLI is configured properly. Add a check for AWS CLI availability and configuration.

```
bash Copy code

# Check if AWS CLI is installed
if ! command -v aws &> /dev/null
then
    echo "AWS CLI not installed. Please install it to proceed."
    exit 1
fi

# Check if AWS CLI is configured
aws sts get-caller-identity &> /dev/null
if [ $? -ne 0 ]; then
    echo "AWS CLI is not configured. Please run 'aws configure' to set up."
    exit 1
fi
```



5. Defining the Output File

5. Defining the Output File

Specify the output file where the report will be saved.

```
bash Copy code  
  
# Define the output file  
output_file="aws_resource_report_$(date +%Y-%m-%d').txt"
```

6. Tracking EC2 Instances

Use AWS CLI commands to get the list of EC2 instances and their statuses.

```
bash Copy code  
  
# Track EC2 Instances  
echo "EC2 Instances:" > $output_file  
aws ec2 describe-instances --query "Reservations[*].Instances[*].[InstanceId, State.Name]"  
echo "" >> $output_file
```

7. Tracking S3 Buckets

Similarly, get the list of S3 buckets.

```
bash Copy code  
  
# Track S3 Buckets  
echo "S3 Buckets:" >> $output_file  
aws s3api list-buckets --query "Buckets[*].Name" --output text >> $output_file  
echo "" >> $output_file
```

8. Tracking Lambda Functions

Get the list of Lambda functions.



8. Tracking Lambda Functions

Get the list of Lambda functions.

```
bash Copy code

# Track Lambda Functions
echo "Lambda Functions:" >> $output_file
aws lambda list-functions --query "Functions[*].FunctionName" --output text >> $output_file
echo "" >> $output_file
```

9. Tracking IAM Users

Get the list of IAM users.

```
bash Copy code

# Track IAM Users
echo "IAM Users:" >> $output_file
aws iam list-users --query "Users[*].UserName" --output text >> $output_file
echo "" >> $output_file
```

10. Finalizing the Script

End the script by printing a success message.

```
bash Copy code

echo "AWS resource usage report generated: $output_file"
```


11. Making the Script Executable

Save and exit the editor by pressing `ESC`, typing `:wq`, and pressing `Enter`. Then, make the script executable.

11. Making the Script Executable

Save and exit the editor by pressing `ESC`, typing `:wq`, and pressing `Enter`. Then, make the script executable.

bash


 Copy code

```
chmod +x aws_resource_tracker.sh
```

12. Automating with Cron Job

To automate the script execution daily at 6 PM, add a Cron job. Edit the Cron table using `crontab -e`.

bash

 Copy code


```
0 18 * * * /path/to/aws_resource_tracker.sh
```

This Cron job will run the script every day at 6 PM.

Complete Script

Here is the complete script for reference:

bash

 Copy code

```
#!/bin/bash
```

```
Author: Abhishek
```

```
Date: 11th Jan
```

```
Version: V1
```

```
Description: This script will report the AWS resource usage.
```


```
Check if AWS CLI is installed
```



Complete Script

Here is the complete script for reference:

bash

 Copy code

```
#!/bin/bash

# Author: Abhishek
# Date: 11th Jan
# Version: V1
# Description: This script will report the AWS resource usage.

# Check if AWS CLI is installed
if ! command -v aws &> /dev/null
then
    echo "AWS CLI not installed. Please install it to proceed."
    exit 1
fi

# Check if AWS CLI is configured
aws sts get-caller-identity &> /dev/null
if [ $? -ne 0 ]; then
    echo "AWS CLI is not configured. Please run 'aws configure' to set up."
    exit 1
fi

# Define the output file
output_file="aws_resource_report_$(date +%Y-%m-%d').txt"

# Track EC2 Instances
echo "EC2 Instances:" > $output_file
aws ec2 describe-instances --query "Reservations[*].Instances[*].[InstanceId, State.Name]"
echo "" >> $output_file
```

```

11

# Define the output file
output_file="aws_resource_report_$(date +%Y-%m-%d').txt"

# Track EC2 Instances
echo "EC2 Instances:" > $output_file
aws ec2 describe-instances --query "Reservations[*].Instances[*].[InstanceId, State.Name]"
echo "" >> $output_file

# Track S3 Buckets
echo "S3 Buckets:" >> $output_file
aws s3api list-buckets --query "Buckets[*].Name" --output text >> $output_file
echo "" >> $output_file

# Track Lambda Functions
echo "Lambda Functions:" >> $output_file
aws lambda list-functions --query "Functions[*].FunctionName" --output text >> $output_file
echo "" >> $output_file

# Track IAM Users
echo "IAM Users:" >> $output_file
aws iam list-users --query "Users[*].UserName" --output text >> $output_file
echo "" >> $output_file

echo "AWS resource usage report generated: $output_file"

```

This detailed explanation should help you understand the steps involved in creating and automating an AWS resource tracker shell script.

< 2/2 >     

To get whole code of ‘Complete Script’ or either you can on [github ‘devops-zero-to-hero/Day-7 Live AWS Project using SHELL SCRIPTING for DevOps AWSDevOps project/](#)Writing the Script.md’:



```
1  ### Complete Script
2
3  Here is the complete script for reference:
4
5  ```bash
6  #!/bin/bash
7
8  # Author: Abhishek
9  # Date: 11th Jan
10 # Version: V1
11 # Description: This script will report the AWS resource usage.
12
13 # Check if AWS CLI is installed
14 if ! command -v aws &> /dev/null
15 then
16     echo "AWS CLI not installed. Please install it to proceed."
17     exit 1
18 fi
19
```



```
1  # Check if AWS CLI is configured
2  aws sts get-caller-identity &> /dev/null
3  if [ $? -ne 0 ]; then
4      echo "AWS CLI is not configured. Please run 'aws configure' to set up."
5      exit 1
6  fi
7
8  # Define the output file
9  output_file="aws_resource_report_$(date +%Y-%m-%d').txt"
10
11 # Track EC2 Instances
12 echo "EC2 Instances:" > $output_file
13 aws ec2 describe-instances --query "Reservations[*].Instances[*].[InstanceId, State.Name]" --output text >> $output_file
14 echo "" >> $output_file
15
16 # Track S3 Buckets
17 echo "S3 Buckets:" >> $output_file
18 aws s3api list-buckets --query "Buckets[*].Name" --output text >> $output_file
19 echo "" >> $output_file
20
```



```
1 # Track Lambda Functions
2 echo "Lambda Functions:" >> $output_file
3 aws lambda list-functions --query "Functions[*].FunctionName" --output text >> $output_file
4 echo "" >> $output_file
```



```
1 # Track IAM Users
2 echo "IAM Users:" >> $output_file
3 aws iam list-users --query "Users[*].UserName" --output text >> $output_file
4 echo "" >> $output_file
5
6 echo "AWS resource usage report generated: $output_file"
7 ````
```



```
1 This detailed explanation should help you understand the steps involved in creating and automating an AWS resource tracker shell script.
```

16:00 - 20:00



Step-by-Step Breakdown of Learning and Using AWS CLI Commands

In this section, you'll learn how to discover and use specific AWS CLI commands, starting with listing S3 buckets and then moving on to other AWS services like EC2, Lambda, and IAM. The process involves exploring the AWS CLI documentation to find relevant commands and then implementing them in a script.

1. Discovering AWS CLI Commands for S3

To start with, you want to list all the S3 buckets in your AWS account. Here's how you can find and use the command:

- **Exploring S3 Commands:**
 - First, open the AWS CLI documentation or use the `aws help` command.
 - Navigate to the S3 section and look for a command that can list the buckets. In this case, the `ls` command is what you need.
- **Command to List S3 Buckets:**
 - The command `aws s3 ls` is used to list all the S3 buckets.
- **Implementing in the Script:**
 - Add the command to your script with comments for clarity.

```
bash
```

[Copy code](#)

```
# List S3 buckets  
aws s3 ls
```

2. Learning the Command for EC2 Instances

Next, you want to list all EC2 instances. If you're unfamiliar with the command, here's how to find it:

- Exploring EC2 Commands:



2. Learning the Command for EC2 Instances

Next, you want to list all EC2 instances. If you're unfamiliar with the command, here's how to find it:

- Exploring EC2 Commands:
 - Go back to the AWS CLI documentation and search for EC2 commands.
 - The command `describe-instances` will give you the information about all EC2 instances.
- Command to List EC2 Instances:
 - The command `aws ec2 describe-instances` retrieves details of all EC2 instances.
- Implementing in the Script:
 - Add this command to your script with appropriate comments.

```
bash Copy code  
  
# List EC2 instances  
aws ec2 describe-instances
```

3. Finding the Command for Lambda Functions

To list AWS Lambda functions, follow these steps:

- Exploring Lambda Commands:
 - Look for Lambda-related commands in the AWS CLI documentation.
 - The command `list-functions` will list all your Lambda functions.
- Command to List Lambda Functions:
 - The command `aws lambda list-functions` is used to list Lambda functions.
- Implementing in the Script:
 - Add this command to your script.



- Implementing in the Script:
 - Add this command to your script.

```
bash Copy code  
  
# List AWS Lambda functions  
aws lambda list-functions
```

4. Listing IAM Users

Finally, to list IAM users, here's how you can discover and use the relevant command:

- Exploring IAM Commands:
 - Search for IAM commands in the AWS CLI documentation.
 - The command `list-users` will provide a list of IAM users.
- Command to List IAM Users:
 - The command `aws iam list-users` is used to list IAM users.
- Implementing in the Script:
 - Add this command to your script.

```
bash Copy code  
  
# List IAM users  
aws iam list-users
```

Final Script Implementation

Here's what your script would look like with all these commands included:

```
bash Copy code

#!/bin/bash

# Author: Abhishek
# Date: 11th Jan
# Version: V1
# Description: This script will report the AWS resource usage.

# List S3 buckets
aws s3 ls

# List EC2 instances
aws ec2 describe-instances

# List AWS Lambda functions
aws lambda list-functions

# List IAM users
aws iam list-users
```

OR



Let's break down and explain the concepts and steps involved in this process in detail.

1. Exploring AWS CLI Commands

When learning how to use AWS CLI (Command Line Interface) commands, it's common to start by looking up specific commands relevant to the tasks you want to perform. In this case, the task is to list AWS S3 buckets.

- **Using AWS CLI Documentation:** To find the appropriate command, the speaker navigates through AWS CLI documentation. For S3, they identify the `aws s3 ls` command, which is used to list all the S3 buckets in your AWS account.

2. Writing Shell Scripts with Comments

The speaker emphasizes the importance of adding comments before each command in a script. Comments explain what each part of the script does, which is crucial for readability and maintainability, especially when others or even the original author revisit the script later.

- **Example:** In the script, before listing S3 buckets, the speaker adds a comment:

```
bash
```

[Copy code](#)

```
# List S3 buckets
```

```
aws s3 ls
```

This helps anyone reading the script understand that this command is used to list all S3 buckets.

3. Exploring Other AWS CLI Commands

The speaker continues by finding commands for other AWS services such as EC2, Lambda, and IAM.

- **EC2 Instances:** To list EC2 instances, they use the command `aws ec2 describe-instances`. This command provides details about the instances running in your AWS environment.

```
bash
```

[Copy code](#)

3. Exploring Other AWS CLI Commands

The speaker continues by finding commands for other AWS services such as EC2, Lambda, and IAM.

- **EC2 Instances:** To list EC2 instances, they use the command `aws ec2 describe-instances`. This command provides details about the instances running in your AWS environment.

```
bash
```

[Copy code](#)

```
# List EC2 instances
aws ec2 describe-instances
```

- **Lambda Functions:** For listing Lambda functions, the command is `aws lambda list-functions`.

```
bash
```

[Copy code](#)

```
# List AWS Lambda functions
aws lambda list-functions
```

- **IAM Users:** To list IAM users, they look up the appropriate command, which is `aws iam list-users`.

```
bash
```

[Copy code](#)

```
# List IAM users
aws iam list-users
```

The process of finding the right command involves checking the AWS CLI documentation and searching for commands related to the specific AWS service (like EC2, Lambda, IAM).

4. Running and Testing the Script

Before finalizing the script, it's important to test it. The speaker explains the steps to make the script executable and run it.

- **Changing File Permissions:** Before running the script, permissions need to be set. Using `chmod 777` is mentioned, which gives all permissions (read, write, execute) to all users. This is generally not recommended for production scripts but is okay for testing.

```
bash Copy code  
  
chmod 777 aws_resource_tracker.sh
```

Note: `chmod 755` is a safer alternative, allowing the owner to write and others to read/execute.

- **Executing the Script:** After setting the permissions, the script is executed using:

```
bash Copy code  
  
./aws_resource_tracker.sh
```

5. Redirecting Output to a File

For better readability and to capture the output, the script's results can be redirected to a file. This is done using the redirection operator (`>`).

- **Example: Redirecting the output to a file named `output.txt`:**

```
bash Copy code  
  
./aws_resource_tracker.sh > output.txt
```

- **Using `more` for Paginated Output:** If the output is too long, you can pipe it through the `more` command, which allows you to scroll through the output page by page.

```
bash Copy code
```

- **Using `more` for Paginated Output:** If the output is too long, you can pipe it through the `more` command, which allows you to scroll through the output page by page.

```
bash Copy code  
  
./aws_resource_tracker.sh | more
```

6. Improvising the Script

The speaker hints at further improving the script by introducing more advanced techniques such as:

- **Set Arguments:** These allow for more flexibility by enabling the script to accept input parameters.
- **Debug Mode:** A debug mode helps trace and debug issues within the script.
- **Avoiding Pipe Issues:** Pipes can cause issues if not handled correctly, and the speaker plans to address these.
- **Modularizing with Functions:** Although not covered here, breaking the script into functions would make it more modular and easier to maintain.

Summary

This segment teaches the basics of writing a shell script to interact with AWS resources using AWS CLI. Key takeaways include:

- Using AWS CLI documentation to find and understand commands.
- Adding comments to scripts for clarity.
- Testing scripts by changing permissions and executing them.
- Redirecting output to files for better readability.
- Planning for script improvements to handle complex scenarios and edge cases.

20:00 - 24:00 Print Statement



Let's break down the concepts and steps in this part of the script, explaining them in detail and making them easy to understand.

1. Adding Print Statements for Clarity

When you run a script that interacts with AWS services, it can be hard to distinguish between different outputs, especially when there's a lot of data. To make the script more user-friendly, you can add print statements using the `echo` command in Bash.

- **Print Statements:** By adding `echo` commands before each section of your script, you provide context to the output, making it clear what each section is about.

```
bash Copy code  
  
echo "List of S3 buckets"  
aws s3 ls  
  
echo "List of EC2 instances"  
aws ec2 describe-instances  
  
echo "List of Lambda functions"  
aws lambda list-functions  
  
echo "List of IAM users"  
aws iam list-users
```

Explanation:

- `echo "message"`: This prints a message to the terminal, helping users understand the context of the following command's output.
- Each `aws` command now has a corresponding print statement that explains what the output will show.



2. Using Debug Mode with `set -x`

In shell scripting, the `set` command can be used to control the behavior of the shell. Two common options are `set -x` and `set +x`, which toggle debug mode.

- **Debug Mode (`set -x`):** When you enable this mode, the shell prints each command to the terminal as it is executed, which can be very helpful for debugging.

```
bash Copy code  
  
set -x
```

Explanation:

- `set -x`: Enables debugging by printing each command before it is executed.
- `set +x`: Disables debugging, returning to normal execution.

Use Case:

- This mode is useful when you want to see the exact commands being run, which helps in understanding the flow of the script and diagnosing where things might be going wrong.

3. Filtering Output with `jq`

AWS CLI often returns data in JSON format, which can be overwhelming if you're only interested in specific pieces of information, such as instance IDs. The `jq` tool in Linux is used to parse and filter JSON data.

- **Filtering with `jq`:** To extract specific information like instance IDs from a large JSON response, you can use `jq`.

```
bash Copy code  
  
aws ec2 describe-instances | jq '.Reservations[].Instances[].InstanceId'
```

Explanation:

- `jq`: A powerful command-line tool for parsing JSON. It allows you to filter and transform JSON data in various ways.
- **Filtering the JSON:** The command `jq '.Reservations[].Instances[].InstanceId'` navigates through the JSON structure returned by `describe-instances` and extracts only the instance IDs.

Benefit:

- This makes the output much more manageable and easier to understand, especially when reporting to others who may only need to see key information like instance IDs.

4. Summary of Output Organization

The modifications improve the script's output in several ways:

- **Clarity:** By using `echo` statements, you provide clear headers for each section of the output, making it easier for users to understand what they're looking at.
- **Debugging:** Using `set -x` helps you or others understand how the script executes, which is invaluable during development or troubleshooting.
- **Simplicity:** Filtering large outputs with `jq` reduces the noise and focuses on the most important data, such as instance IDs, making the report more concise and easier to digest.

Example of the Final Script with All Improvements

```
bash Copy code

#!/bin/bash

echo "List of S3 buckets"

aws s3 ls
```

Example of the Final Script with All Improvements

```
bash Copy code

#!/bin/bash

echo "List of S3 buckets"
aws s3 ls

echo "List of EC2 instances"
aws ec2 describe-instances | jq '.Reservations[].Instances[].InstanceId'

echo "List of Lambda functions"
aws lambda list-functions

echo "List of IAM users"
aws iam list-users

set -x # Enable debug mode

# Example commands with debug mode enabled to see their execution
aws ec2 describe-instances | jq '.Reservations[].Instances[].InstanceId'

set +x # Disable debug mode
```

This script is now much more user-friendly, easier to debug, and provides clear, focused output that's easy to understand and share.



24: 28 mints video:



Let's break down the script provided and understand each concept in detail, with examples and expected outputs.

Overview of the Script

The script lists various AWS resources (S3 buckets, EC2 instances, Lambda functions, IAM users) and formats the output for clarity. It uses the `jq` tool to filter JSON output from AWS CLI commands.

Concepts Explained

1. Using `jq` for JSON Parsing

`jq` is a lightweight and flexible command-line JSON processor. It allows you to slice, filter, map, and transform JSON data.

Basic Usage Example:

- Command:

```
bash Copy code  
  
aws ec2 describe-instances | jq '.Reservations[].Instances[].InstanceId'
```

- Explanation:

- `aws ec2 describe-instances`: This AWS CLI command retrieves detailed information about EC2 instances.
- `|`: This pipe sends the output of the command on its left as input to the command on its right.
- `jq '.Reservations[].Instances[].InstanceId'`: This jq filter navigates through the JSON structure and extracts the instance IDs.
 - `.Reservations[]`: Iterates over each element in the `Reservations` array.

- `aws ec2 describe-instances`: This AWS CLI command retrieves detailed information about EC2 instances.
- `|`: This pipe sends the output of the command on its left as input to the command on its right.
- `jq '.Reservations[].Instances[].InstanceId'`: This jq filter navigates through the JSON structure and extracts the instance IDs.
 - `.Reservations[]`: Iterates over each element in the `Reservations` array.
 - `.Instances[]`: Iterates over each element in the `Instances` array.
 - `.InstanceId`: Extracts the `InstanceId` field from each instance.

Output Example:

```

json                                                                    Copy code
"i-0123456789abcdef0"
"i-0abcdef1234567890"

```

2. Adding Print Statements with `echo`


Command:

```

bash                                                                    Copy code
echo "List of S3 buckets"
aws s3 ls

```

Explanation:

- `echo "List of S3 buckets"`: Prints the text "List of S3 buckets" to the terminal. This helps in identifying the output section. 
-

3. Redirecting Output to a File

Command:

```
bash Copy code  
  
aws s3 ls > resource_tracker.txt
```

Explanation:

- `>`: Redirects the output of the command on its left to the file specified on its right (`resource_tracker.txt`). If the file exists, it is overwritten.

4. Combining Commands with Improved Output

Script:

```
bash Copy code  
  
#!/bin/bash  
  
echo "List of S3 buckets" > resource_tracker.txt  
aws s3 ls >> resource_tracker.txt  
  
echo "List of EC2 instances" >> resource_tracker.txt  
aws ec2 describe-instances | jq '.Reservations[].Instances[].InstanceId' >> resource_t  
  
echo "List of Lambda functions" >> resource_tracker.txt  
aws lambda list-functions >> resource_tracker.txt  
  
echo "List of IAM users" >> resource_tracker.txt  
aws iam list-users >> resource_tracker.txt
```

Explanation:

- ``>``: Creates or overwrites the file `resource_tracker.txt` with the specified output.
- ``>>``: Appends the output to `resource_tracker.txt` without overwriting existing content.

Final Improved Script

This script collects and formats AWS resource information and saves it to a file for easier reading and reporting.

```
bash Copy code

#!/bin/bash

# Print list of S3 buckets and save to file
echo "List of S3 buckets" > resource_tracker.txt
aws s3 ls >> resource_tracker.txt

# Print list of EC2 instances (only instance IDs) and save to file
echo "List of EC2 instances" >> resource_tracker.txt
aws ec2 describe-instances | jq '.Reservations[].Instances[].InstanceId' >> resource_tracker.txt

# Print list of Lambda functions and save to file
echo "List of Lambda functions" >> resource_tracker.txt
aws lambda list-functions >> resource_tracker.txt

# Print list of IAM users and save to file
echo "List of IAM users" >> resource_tracker.txt
aws iam list-users >> resource_tracker.txt
```

Running the Script and Viewing the Output

1. Make the Script Executable:

```
bash
```

[Copy code](#)

```
chmod +x aws_resource_tracker.sh
```

2. Run the Script:

```
bash
```

[Copy code](#)

```
./aws_resource_tracker.sh
```

3. View the Output:


```
bash
```

[Copy code](#)

```
cat resource_tracker.txt
```

Example Output

mathematica

 Copy code

List of S3 buckets

2024-01-01 12:34:56 bucket-name-1

2024-01-02 12:34:56 bucket-name-2

List of EC2 instances

"i-0123456789abcdef0"

"i-0abcdef1234567890"

List of Lambda functions

```
{
  "Functions": [
    {
      "FunctionName": "example-function-1",
      "Runtime": "nodejs12.x",
      ...
    },
    ...
  ]
}
```

List of IAM users

```
{
  "Users": [
    {
      "UserName": "user1",
      "CreateDate": "2023-01-01T00:00:00Z",
      ...
    },
    ...
  ]
}
```



Integrating with `cron`

You can schedule this script to run at regular intervals using `cron`.

1. Open the `cron` Editor:

```
bash Copy code  
  
crontab -e
```

2. Add a New Cron Job:

```
bash Copy code  
  
0 0 * * * /path/to/aws_resource_tracker.sh
```

Explanation:

- `0 0 * * *`: Runs the script daily at midnight.
- `/path/to/aws_resource_tracker.sh`: The full path to your script.

This setup will ensure that your AWS resource information is regularly updated and saved to a file, providing an easy reference for you or your team.