

1.Explain in your own words and examples, what is Shell Scripting for DevOps?

Shell scripting for DevOps involves using shell scripting languages (such as Bash) to automate and streamline various tasks in the DevOps workflow. DevOps (Development and Operations) is a software development methodology that emphasizes collaboration, communication, and integration between software developers and IT operations teams.

Here's how shell scripting is used in DevOps:

Automation: Shell scripts automate repetitive tasks such as deploying applications, provisioning infrastructure, configuring servers, running tests, and monitoring systems. For example, you can write a shell script to deploy a new version of an application to multiple servers simultaneously.

Configuration Management: Shell scripts are used to manage configurations across servers and environments. Configuration management tools like Ansible, Chef, or Puppet often use shell scripts to execute tasks on remote servers.

Continuous Integration/Continuous Deployment (CI/CD): Shell scripts play a crucial role in CI/CD pipelines by automating build, test, and deployment processes. For example, a shell script can be triggered by a Git commit webhook to build and deploy an application automatically.

Monitoring and Logging: Shell scripts can be used to monitor system health, collect metrics, and analyse logs. For instance, you can write a shell script to check server CPU usage periodically and send alerts if it exceeds a certain threshold.

Infrastructure as Code (IaC): Shell scripts are often used to provision and manage infrastructure resources programmatically. Tools like Terraform or CloudFormation use shell scripts to orchestrate the creation and configuration of cloud resources.

Customization and Extensions: Shell scripts can be customized and extended according to specific project requirements. Developers can write scripts tailored to their project's needs, integrating with existing tools and processes.

Overall, shell scripting in DevOps enables teams to improve efficiency, consistency, and reliability in software development and operations by automating tasks, managing configurations, implementing CI/CD pipelines, monitoring systems, managing infrastructure, and providing customization and extensions to streamline the development and deployment processes.

Q2. What is `#!/bin/bash`? can we write `#!/bin/sh` as well?

The `#!/bin/bash` or "shebang" line at the beginning of a script file indicates the path to the interpreter that should be used to execute the script. In this case, it specifies that the script should be executed using the Bash shell.

Similarly, you can use `#!/bin/sh` to specify that the script should be executed using the Bourne shell or a compatible shell.

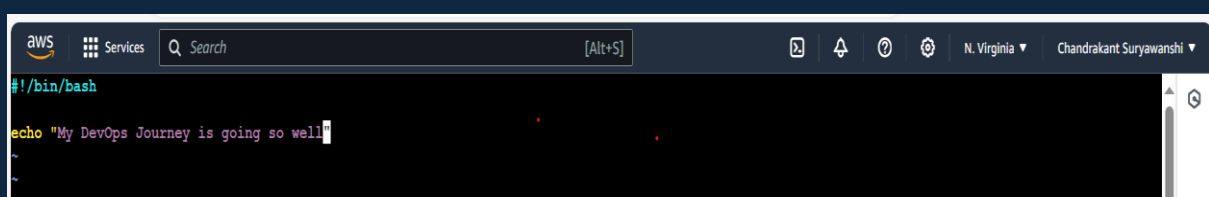
In general:

1. `#!/bin/bash` specifies the Bash shell, which is a widely used shell on Unix-like operating systems and is the default shell on most Linux distributions.
2. `#!/bin/sh` specifies the Bourne shell, which is a simpler shell compared to Bash and is available on most Unix-like systems.

The choice between `#!/bin/bash` and `#!/bin/sh` depends on your specific needs. If your script uses features specific to Bash, such as arrays or associative arrays, then you should use `#!/bin/bash`. However, if your script only uses features that are compatible with the Bourne shell, you can use `#!/bin/sh` to ensure portability across different Unix-like systems.

Q3. Write a Shell Script which prints I will complete **#My DevOps journey is going so well.**

Certainly! Here's a simple shell script that prints "I will complete # My DevOps journey is going so well."

A screenshot of an AWS CloudShell terminal window. The top bar shows the AWS logo, 'Services', a search bar, and the user's location 'N. Virginia' and name 'Chandrakant Suryawanshi'. The terminal has a dark background with a light blue prompt `#!/bin/bash`. Below it, the command `echo "My DevOps Journey is going so well"` is entered, and the output `My DevOps Journey is going so well` is displayed on the next line.

```

ubuntu@ip-172-31-21-196:~$ chmod 755 print.sh
chmod: cannot access 'print.sh': No such file or directory
ubuntu@ip-172-31-21-196:~$ chmod 755 Print.sh
ubuntu@ip-172-31-21-196:~$ ls
10.txt  3.txt  5.txt  7.txt  9.txt  Print.sh  chandu.txt  devopsbatch-6  file10.txt  file3.txt  file5.txt  file7.txt  file9.txt
2.txt  4.txt  6.txt  8.txt  Git    aws.txt  devops_tool.txt  file         file2.txt  file4.txt  file6.txt  file8.txt  folder
ubuntu@ip-172-31-21-196:~$ ./Print.sh
My DevOps Journey is going so well

```

Save this script in a file, let's say **Print.sh**. Make sure to give execute permission to the script file using the command **chmod 755 Print.sh**. Then you can run the script using **./Print.sh**, and it will print the message.

Q4. Write a Shell Script to take user input, input from arguments and print the variables?

Below is a shell script that demonstrates taking user input and input from arguments, and then prints the variables:

```

#!/bin/bash # Taking user input

echo "Enter your name:"

read name

# Taking input from arguments

arg1=$1

arg2=$2

# Printing variables

echo "User input: $name"

echo "Argument 1: $arg1"

echo "Argument 2: $arg2"

```

Save this script in a file, let's say **input.sh**. Make sure to give execute permission to the script file using the command **chmod +x input.sh**. Then you can run the script using **./input.sh**, and it will prompt you to enter your name and then print the variables. You can also provide arguments while running the script like **./input.sh argument1 argument2**, and it will print those arguments as well.

```
aws Services Q Search [Alt+S] N. Virginia Chandrakant Suryawanshi
#!/bin/bash

# Taking user input
echo "Enter your name:"
read name

# Taking input from arguments
arg1=$1
arg2=$2

# Printing variables
echo "User input: $name"
echo "Argument 1: $arg1"
echo "Argument 2: $arg2"
```

```
ubuntu@ip-172-31-21-196:~$ touch input.sh
ubuntu@ip-172-31-21-196:~$ vim input.sh
ubuntu@ip-172-31-21-196:~$ chmod 755 input.sh
ubuntu@ip-172-31-21-196:~$ ./input.sh
Enter your name:
Chandrakant
User input: Chandrakant
Argument 1:
Argument 2:
ubuntu@ip-172-31-21-196:~$ vim input.sh
ubuntu@ip-172-31-21-196:~$ ./input.sh Sanjay Suryawanshi
Enter your name:
Chandrakant
User input: Chandrakant
Argument 1: Sanjay
Argument 2: Suryawanshi
ubuntu@ip-172-31-21-196:~$
```

Q5. Write an Example of If else in Shell Scripting by taking input from user to comparing 2 numbers?

- We prompt the user to enter the first number using **echo** and **read** and store the input in the variable **num1**.
- Similarly, we prompt the user to enter the second number and store it in the variable **num2**.
- We use an if-elif-else statement to compare the values of **num1** and **num2**.
- If the numbers are equal, we print a message indicating that.
- If **num1** is greater than **num2**, we print a message indicating that.
- If **num1** is less than **num2**, we print a message indicating that.

```
aws Services Search [Alt+S] N. Virginia Chandrakant Suryawanshi

#!/bin/bash

# Prompt for the user to enter the first number
echo "Enter num1"
read num1

# Prompt for the user to enter the second number
echo "Enter num2"
read num2

# Compare the numbers
if [ $num1 -gt $num2 ]; then
    echo "$num1 is greater than $num2"
elif [ $num1 -lt $num2 ]; then
    echo "$num1 is less than $num2"
else
    echo "$num1 is equal to $num2"
fi
```

```
ubuntu@ip-172-31-21-196:~$ vim compare_num.sh
ubuntu@ip-172-31-21-196:~$ ./compare_num.sh
Enter num1
25
Enter num2
50
25 is less than 50
ubuntu@ip-172-31-21-196:~$ ./compare_num.sh
Enter num1
25
Enter num2
25
25 is equal to 25
ubuntu@ip-172-31-21-196:~$ ./compare_num.sh
Enter num1
50
Enter num2
25
50 is greater than 25
ubuntu@ip-172-31-21-196:~$
```