Here's a Python script that takes three parameters: two strings and a directory name, and substitutes any occurrence of the first string with the second string for any file in the directory, recursively

import os

import sys

def replace\_string\_in\_file(filepath, old\_string, new\_string):

with open(filepath, 'r') as f:

content = f.read()

content = content.replace(old\_string, new\_string)

with open(filepath, 'w') as f:

f.write(content)

def replace\_string\_in\_directory(directory, old\_string, new\_string):

for root, dirs, files in os.walk(directory):

for file in files:

filepath = os.path.join(root, file)

replace\_string\_in\_file(filepath, old\_string, new\_string)

if \_\_name\_\_ == '\_\_main\_\_':

if len(sys.argv) != 4:

print('Usage: python script.py <old\_string> <new\_string> <directory>')

sys.exit(1)

old\_string = sys.argv[1]

new\_string = sys.argv[2]

directory = sys.argv[3]

replace\_string\_in\_directory(directory, old\_string, new\_string)

Explanation:

* The function **replace\_string\_in\_file(filepath, old\_string, new\_string)** takes the file path, a string to replace (old\_string), and a replacement string (new\_string) as parameters. It reads the contents of the file, replaces all occurrences of **old\_string** with **new\_string**, and writes the modified contents back to the same file.
* The function **replace\_string\_in\_directory(directory, old\_string, new\_string)** takes the directory name to traverse, the string to replace (old\_string), and the replacement string (new\_string) as parameters. It uses the **os.walk()** function to recursively traverse all files and directories in the directory and calls **replace\_string\_in\_file()** to replace the string in each file.
* Finally, the **if \_\_name\_\_ == '\_\_main\_\_':** section checks that the script is being run directly (rather than being imported into another script) and retrieves the three arguments passed on the command line: **old\_string**, **new\_string**, and **directory**. It then calls the **replace\_string\_in\_directory()** function with these arguments.

To run this script, you can type the following command in your terminal:

python script.py <old\_string> <new\_string> <directory>

Replace **<old\_string>**, **<new\_string>**, and **<directory>** with the appropriate values.

Here's a Bash script that takes three parameters: two strings and a directory name, and substitutes any occurrence of the first string with the second string for any file in the directory, recursively

#!/bin/bash

if [ $# -ne 3 ]; then

echo "Usage: $0 <old\_string> <new\_string> <directory>"

exit 1

fi

old\_string="$1"

new\_string="$2"

directory="$3"

find "$directory" -type f -print0 | while IFS= read -r -d '' file; do

sed -i "s/$old\_string/$new\_string/g" "$file"

done

Explanation:

* The first line **#!/bin/bash** indicates that the script should be executed with the bash shell.
* The **if [ $# -ne 3 ]; then** section checks that there are exactly three arguments passed on the command line. If not, the script displays an error message and exits with a non-zero exit code (1).
* The variables **old\_string**, **new\_string**, and **directory** are initialized with the values of the arguments passed on the command line.
* The command **find "$directory" -type f -print0** recursively traverses the **$directory** directory and displays all files (**-type f**), separated by the NULL character (**-print0**). This output is then processed by the **while** loop.
* The **while** loop uses **IFS= read -r -d '' file** to read the names of the found files one by one, separating the lines according to the NULL character. **IFS=** sets the IFS (Internal Field Separator) to empty so that the loop correctly handles filenames containing spaces or special characters. The **-r** option ensures that newline characters in filenames are handled correctly, and the **-d ''** option sets the end-of-line character to be NULL.
* For each file, the command **sed -i "s/$old\_string/$new\_string/g" "$file"** replaces all occurrences of **$old\_string** with **$new\_string** and writes the modifications directly to the file (**-i** for "in-place").

The script ends with an exit code of 0 (success).

To run this script, save it to a file (e.g., **replace\_strings.sh**), then type the following command in your terminal:

bash replace\_strings.sh <old\_string> <new\_string> <directory>

Replace **<old\_string>**, **<new\_string>**, and **<directory>** with the appropriate values.

Exercice 2 :

here's a Bash script that counts the number of script files in a directory, subdividing it by the shebang interpreter:

#!/bin/bash

# Ask for the directory to examine

read -p "Enter the absolute path of the directory to examine: " directory

# check that the directory exists

if [ ! -d "$directory" ]; then

echo "Le répertoire n'existe pas."

exit 1

fi

# display a confirmation message

echo "Le répertoire à examiner est : $directory"

# count the number of script files for each shebang interpreter

declare -A script\_counts=()

#Iterates over all files in the directory

for filepath in "$directory"/\*; do

if [ -x "$filepath" ] && [ -f "$filepath" ]; then

first\_line=$(head -n 1 "$filepath" | tr -d '\r')

if [[ $first\_line == "#!"\* ]]; then

script\_counts[$shebang]=$(( ${script\_counts[$shebang]} + 1 ))

fi

fi

done

# and display the results

echo "Distribution of script files in $directory :"

for shebang in "${!script\_counts[@]}"; do

echo "- $shebang : ${script\_counts[$shebang]} files"

done

Explication :

The script begins by asking the user for the absolute path of the directory to examine using the read command. Then, it checks if the directory exists using the [ -d "$directory" ] operator. If the directory does not exist, the script displays an error message and exits with an error code of 1.

Next, the script initializes an empty associative array script\_counts to store the counts. The associative array uses the shebang interpreter as the key and the number of script files using that shebang interpreter as the value.

Next, the script traverses through all the files in the directory using a for loop and the file pattern "$directory"/\*. For each file, the script uses the [ -x "$filepath" ] operator to check if the file is executable and the [ -f "$filepath" ] operator to check that it is a regular file. If these two conditions are met, the script uses the command head -n 1 "$filepath" | tr -d '\r' to read the first line and remove any possible line break characters. If the first line starts with "#!", the script extracts the shebang interpreter using the string substitution ${first\_line#"#!"}. Finally,

here's a Bash script that counts the number of script files in a directory, subdividing it by the shebang interpreter:

#!/usr/bin/env python3

import os

# Ask for the directory to examine

directory = input("Entrez le chemin absolu du répertoire à examiner : ")

# check that the directory exists

if not os.path.isdir(directory):

print("Le répertoire n'existe pas.")

exit(1)

# display a confirmation message

print(f"Le répertoire à examiner est : {directory}")

# Initializes a dictionary to store the counts

script\_counts = {}

# count the number of script files for each shebang interprete

for filename in os.listdir(directory):

filepath = os.path.join(directory, filename)

if os.access(filepath, os.X\_OK) and os.path.isfile(filepath):

#verifies that the file is executable and is a regular file (and not a directory, symbolic link,etc.)

with open(filepath, "r") as f:

#opens the file in read-only mode

first\_line = f.readline().strip()

#reads the first line (which should be the shebang interpreter)

if first\_line.startswith("#!"):

#verifies that the first line starts with "#!"

shebang = first\_line[2:]

#extracts the shebang interpreter

script\_counts[shebang] = script\_counts.get(shebang, 0) + 1

# increments the counter for this shebang interpreter.

# and display the results

print("Répartition des fichiers de script dans", directory, ":")

for shebang, count in script\_counts.items():

print(f"- {shebang} : {count} fichiers")

Explication :

The script starts by asking the user for the absolute path of the directory to be examined. Then, it checks if the directory exists using the os.path.isdir() function. If the directory does not exist, the script displays an error message and exits with an error code of 1.

Next, the script initializes an empty script\_counts dictionary to store the counts. The dictionary uses the shebang interpreter as the key and the number of script files using that shebang interpreter as the value.

Next, the script iterates over all the files in the directory using the os.listdir() function. For each file, the script uses the os.access() function to check if the file is executable and the os.path.isfile() function to check that it is a regular file. If these two conditions are met, the script opens the file in read-only mode using the with open(filepath, "r") as f: statement and reads the first line using the readline() method. If the first line starts with "#!", the script extracts the shebang interpreter using a string slice shebang = first\_line[2:]. Finally, the script uses the dict.get() method to increment the counter for that shebang interpreter.

Finally, the script displays the results by iterating over the script\_counts dictionary using a for loop. For each key-value pair, the script uses the print() statement to display the number of script files using that shebang interpreter.

Exercice 3 :

Here is the cron string for performing weekly backup of the /home/user directory and sending it to a remote server via ssh:

0 0 \* \* 0 rsync -avz --delete -e "ssh -i /path/to/private/key" /home/user [user@192.168.1.100:/path/to/backup/folder](mailto:user@192.168.1.100:/path/to/backup/folder)

Explanation:

* 0 0 \* \* 0 specifies that the command will be executed every Sunday at midnight (00:00).
* rsync is used to copy files efficiently by only transferring the differences between the source and destination files.
* -avz instructs rsync to copy files recursively, preserving file attributes (such as permissions and timestamps), and compressing data during transfer to reduce the bandwidth used.
* --delete removes destination files that are not present in the source (to ensure that the backup is an exact copy of the source directory).
* -e "ssh -i /path/to/private/key" specifies the use of the private key to connect to the remote server via ssh.
* /home/user is the source directory we want to backup.
* user@192.168.1.100:/path/to/backup/folder is the destination path on the remote server where the backup will be stored.

Note that you will need to replace /path/to/private/key with the absolute path to your ssh private key, and user and 192.168.1.100 with the username and IP address of your remote server. You will also need to manually create the destination folder on the remote server before running this command.

Exercice 4 :

This is a proposal for an architecture to set up a secure, fast, and fault-tolerant WordPress site:

WordPress Architecture

The architecture includes the following components:

* + An Amazon Web Services (AWS) EC2 instance to host the WordPress application.
  + An Elastic Load Balancing (ELB) to distribute traffic between multiple EC2 instances.
  + An Amazon Relational Database Service (RDS) to store WordPress data.
  + An EC2 security group to restrict access to the EC2 instance and RDS database.
  + An Amazon Route 53 DNS zone to associate a domain name with the WordPress site.

The instructions to deploy this architecture are as follows:

1. Create an SSH key to connect to the EC2 instance and note its location.
2. Choose an AWS region to deploy your architecture and create an EC2 security group that allows incoming traffic on HTTP (80), HTTPS (443), and SSH (22) ports only from your IP address. Note the security group ID.
3. Create an RDS MySQL database in the same region as the EC2 instance and note the connection details (host, username, password, and database name).
4. Deploy the EC2 instance using an Amazon Linux 2 image, configuring the SSH key, the EC2 security group created in step 2, and opening HTTP (80) and HTTPS (443) ports in the EC2 firewall. When launching the instance, run a launch script to install Apache, PHP, MySQL, and WordPress (including necessary plugins and themes).
5. Create an ELB that distributes HTTP and HTTPS traffic between EC2 instances. Note the ELB URL.
6. Create a Route 53 DNS zone for your domain name and configure an A record entry that points to the ELB's IP address.
7. Configure the WordPress site to use the RDS database created in step 3.
8. Enable HTTPS by installing a free SSL certificate from Let's Encrypt.

To enable HTTPS and install a free SSL certificate from Let's Encrypt using Terraform, you can use the open-source module "terraform-aws-acm-issuer" to generate a Let's Encrypt SSL certificate and store it in AWS Certificate Manager (ACM).

Here are the steps to enable HTTPS with Terraform and Let's Encrypt:

* Add the "terraform-aws-acm-issuer" module to your Terraform configuration by specifying the necessary information to generate the Let's Encrypt SSL certificate.
* Use the "acm\_issuer" module to generate the Let's Encrypt SSL certificate by running **terraform apply**.
* Create an Elastic Load Balancing (ELB) load balancer for your EC2 instance by specifying the ACM SSL certificate created in the previous step.
* Add security rules to allow incoming HTTPS traffic on the ELB load balancer by opening port 443 in your EC2 security group.

1. Set up regular backups of the database and WordPress files.

* To set up regular backups of the WordPress database and files with Terraform, you can use the following resources:
* AWS Backup: AWS Backup is a fully managed backup service that enables backing up AWS services such as Amazon RDS, Amazon EBS, Amazon DynamoDB, and Amazon EFS. You can create an AWS Backup backup plan that defines the frequency and retention period of backups. To create a backup plan with Terraform, you can use the aws\_backup\_plan resource.
* AWS Lambda: AWS Lambda is a serverless compute service that allows you to run code in response to events, such as the creation of a backup in AWS Backup. You can use AWS Lambda to copy the backups created by AWS Backup to an S3 storage. To create a Lambda function with Terraform, you can use the aws\_lambda\_function resource.
* AWS S3: Amazon S3 is an object storage service that offers scalability, data availability, security, and top-notch performance. You can use AWS S3 to store the backups created by AWS Backup and copied by AWS Lambda. To create an S3 bucket with Terraform, you can use the aws\_s3\_bucket resource.

Terraform script :

# Configuration de la région et des informations d'identification AWS

provider "aws" {

region = "votre-region-ici"

}

# Création du groupe de sécurité EC2

resource "aws\_security\_group" "web" {

name\_prefix = "web-"

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 443

to\_port = 443

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["votre-adresse-ip-ici/32"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

#DNS

resource "aws\_route53\_zone" "main" {

name = "prince-domaine.com"

}

output "zone\_id" {

value = aws\_route53\_zone.main.zone\_id

}

# Création de l'instance EC2

resource "aws\_instance" "wordpress" {

ami = "votre-ami-ici"

instance\_type = "votre-type-instance-ici"

key\_name = "votre-cle-ssh-ici"

vpc\_security\_group\_ids = [aws\_security\_group.web.id]

connection {

type = "ssh"

user = "votre-nom-utilisateur-ssh-ici"

private\_key = file("votre-fichier-cle-ssh-ici")

host = self.public\_ip

}

provisioner "remote-exec" {

inline = [

"sudo yum update -y",

"sudo yum install -y httpd php php-mysql mariadb mariadb-server",

"sudo systemctl start mariadb",

"sudo systemctl enable mariadb",

"sudo mysql\_secure\_installation",

"sudo systemctl start httpd",

"sudo systemctl enable httpd",

"sudo setsebool -P httpd\_can\_network\_connect 1"

]

}

}

# Création de la base de données RDS

resource "aws\_db\_instance" "wordpress" {

allocated\_storage = 10

engine = "mysql"

engine\_version = "votre-version-mysql-ici"

instance\_class = "votre-classe-instance-rds-ici"

name = "votre-nom-base-donnees-ici"

username = "votre-nom-utilisateur-rds-ici"

password = "votre-mot-de-passe-rds-ici"

parameter\_group\_name = "default.mysql8.0"

skip\_final\_snapshot = true

publicly\_accessible = false

}

resource "aws\_elb" "example" {

name = "example-elb"

security\_groups = [aws\_security\_group.example.id]

subnets = aws\_subnet.example.\*.id

listener {

lb\_port = 80

lb\_protocol = "http"

instance\_port = 80

instance\_protocol = "http"

}

listener {

lb\_port = 443

lb\_protocol = "https"

instance\_port = 443

instance\_protocol = "https"

}

health\_check {

healthy\_threshold = 2

unhealthy\_threshold = 2

timeout = 3

interval = 30

target = "HTTPS:443/"

}

health\_check {

healthy\_threshold = 2

unhealthy\_threshold = 2

timeout = 3

interval = 30

target = "HTTP:80/"

}

}

# Créer un plan de sauvegarde AWS Backup pour la base de données RDS

resource "aws\_backup\_plan" "wordpress\_db\_backup\_plan" {

name = "wordpress-db-backup-plan"

rule {

rule\_name = "wordpress-db-backup-rule"

target\_vault\_name = "wordpress-db-backup-vault"

schedule\_expression = "cron(0 2 \* \* ? \*)"

start\_window\_minutes = 120

completion\_window\_minutes = 360

lifecycle {

delete\_after = 90

}

copy\_action {

destination\_vault\_arn = "arn:aws:backup:us-east-1:123456789012:backup-vault:wordpress-db-backup-vault"

}

enable\_continuous\_backup = true

}

}

# Créer une fonction AWS Lambda pour copier les sauvegardes AWS Backup vers un compartiment S3

resource "aws\_lambda\_function" "copy\_backups\_to\_s3" {

filename = "lambda\_function\_payload.zip"

function\_name = "copy\_backups\_to\_s3"

role = aws\_iam\_role.lambda\_exec.arn

handler = "index.handler"

runtime = "nodejs12.x"

environment {

variables = {

bucket\_name = aws\_s3\_bucket.backup\_bucket.id

}

}

}

# Créer un compartiment S3 pour stocker les sauvegardes AWS Backup

resource "aws\_s3\_bucket" "backup\_bucket" {

bucket = "wordpress-backups"

acl = "private"

versioning {

enabled = true

}

}