BASH

THE BOURNE-AGAIN SHELL

SHELL SCRIPTING

Author: - Ratnadeep Sali

1. Automating Server Provisioning (AWS EC2 Launch)

```
#!/bin/bash


# Variables

INSTANCE_TYPE="t2.micro"

AMI_ID="ami-0abcdef1234567890"  # Replace with the correct AMI ID


KEY_NAME="my-key-pair"  # Replace with your key pair name

SECURITY_GROUP="sg-0abc1234def567890"  # Replace with your security group ID

SUBNET_ID="subnet-0abc1234def567890"  # Replace with your subnet ID

REGION="us-west-2"  # Replace with your AWS region 3


# Launch EC2 instance

aws ec2 run-instances --image-id $AMI_ID --count 1 --instance-type $INSTANCE_TYPE \

--key-name $KEY_NAME --security-group-ids $SECURITY_GROUP --subnet-id
$SUBNET_ID --region $REGION


echo "EC2 instance launched successfully!"
```

---

2. System Monitoring (CPU Usage Alert)

```
#!/bin/bash


# Threshold for CPU usage
```

```
CPU_THRESHOLD=80


# Get the current CPU usage
CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk '{print 100 - $1}')


# Check if CPU usage exceeds threshold if (( $(echo "$CPU_USAGE >
$CPU_THRESHOLD" | bc -l) )); then
  echo "Alert: CPU usage is above $CPU_THRESHOLD%. Current usage is
$CPU_USAGE%" | mail -s "CPU Usage Alert" user@example.com
fi
```

---

3. Backup Automation (MySQL Backup)

```
#!/bin/bash
# Variables
DB_USER="root"
DB_PASSWORD="password"
DB_NAME="my_database"
BACKUP_DIR="/backup"
DATE=$(date +%F)


# Create backup directory if it doesn't exist mkdir -p
$BACKUP_DIR
```

# Backup command

```
mysqldump -u $DB_USER -p$DB_PASSWORD $DB_NAME > $BACKUP_DIR/backup_$DATE.sql
```

# Optional: Compress the backup gzip

```
$BACKUP_DIR/backup_$DATE.sql
```

```
echo "Backup completed successfully!"
```

---

4. Log Rotation and Cleanup

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

# Variables

```
LOG_DIR="/var/log/myapp"

ARCHIVE_DIR="/var/log/myapp/archive"

DAYS_TO_KEEP=30
```

# Create archive directory if it doesn't exist

```
mkdir -p $ARCHIVE_DIR
```

# Find and compress logs older than 7 days

```
find $LOG_DIR -type f -name "*.log" -mtime +7 -exec gzip {} \; -exec mv {} $ARCHIVE_DIR \;
```

# Delete logs older than 30 days

```
find $ARCHIVE_DIR -type f -name "*.log.gz" -mtime +$DAYS_TO_KEEP -exec rm {} \;

echo "Log rotation and cleanup completed!"
```

---

5. CI/CD Pipeline Automation (Trigger Jenkins Job)

```
#!/bin/bash

# Jenkins details
JENKINS_URL="http://jenkins.example.com"
JOB_NAME="my-pipeline-job"
USER="your-username"
API_TOKEN="your-api-token"

# Trigger Jenkins job
curl -X POST "$JENKINS_URL/job/$JOB_NAME/build" --user "$USER:$API_TOKEN"

echo "Jenkins job triggered successfully!"
```

---

6. Deployment Automation (Kubernetes Deployment)

```
#!/bin/bash

# Variables
```

```
NAMESPACE="default"

DEPLOYMENT_NAME="my-app"

IMAGE="my-app:v1.0"


# Deploy to Kubernetes

kubectl set image deployment/$DEPLOYMENT_NAME
$DEPLOYMENT_NAME=$IMAGE --namespace=$NAMESPACE


echo "Deployment updated to version $IMAGE!"
```

---

7. Infrastructure as Code (Terraform Apply)

```
#!/bin/bash


# Variables

TF_DIR="/path/to/terraform/config"


# Navigate to Terraform directory cd $TF_DIR


# Run terraform apply terraform apply -

auto-approve


echo "Terraform apply completed successfully!"
```

8. Database Management (PostgreSQL Schema Migration) bash

```
#!/bin/bash


# Variables

DB_USER="postgres"

DB_PASSWORD="password"

DB_NAME="my_database"

MIGRATION_FILE="/path/to/migration.sql"


# Run schema migration

PGPASSWORD=$DB_PASSWORD psql -U $DB_USER -d $DB_NAME -f
$MIGRATION_FILE


echo "Database schema migration completed!"
```

9. User Management (Add User to Group)

```
#!/bin/bash


# Variables

USER_NAME="newuser"
```

GROUP_NAME="devops"

# Add user to group

usermod -aG $GROUP_NAME $USER_NAME

echo "User $USER_NAME added to group $GROUP_NAME!"

---

10. Security Audits (Check for Open Ports)

#!/bin/bash

# Check for open ports

OPEN_PORTS=$(netstat -tuln)

# Check if any ports are open (excluding localhost) if [[$OPEN_PORTS =~ "0.0.0.0" || $OPEN_PORTS =~ "127.0.0.1"]]; then echo "Security Alert: Open ports detected!" echo "$OPEN_PORTS" | mail -s "Open Ports Security Alert" user@example.com

else

 echo "No open ports detected."

Fi

---

11. Performance Tuning

This script clears memory caches and restarts services to free up system resources.

```
#!/bin/bash

# Clear memory caches to free up resources sync; echo

3 > /proc/sys/vm/drop_caches # Restart services to free

up resources systemctl restart nginx systemctl restart

apache2
```

---

12. Automated Testing

This script runs automated tests using a testing framework like pytest for Python or JUnit for Java.

```
#!/bin/bash
# Run unit tests using pytest (Python example)

pytest tests/


# Or, run JUnit tests (Java example) mvn test
```

---

13. Scaling Infrastructure

This script automatically scales EC2 instances in an Auto Scaling group based on CPU usage.

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

# Check CPU usage and scale EC2 instances

```
CPU_USAGE=$(aws cloudwatch get-metric-statistics --namespace AWS/EC2

--metric-name CPUUtilization --dimensions

Name=InstanceId,Value=i-1234567890abcdef0 --statistics Average --period 300

--start-time $(date -d '5 minutes ago' --utc +%FT%TZ) --end-time $(date --utc +%FT%TZ) --query
'Datapoints[0].Average' --output text)


if (( $(echo "$CPU_USAGE > 80" | bc -l) )); then

  aws autoscaling update-auto-scaling-group --auto-scaling-group-name my-auto-scaling-group --desired-
capacity 3

fi
```

---

14. Environment Setup

This script sets environment variables for different environments (development, staging, production).

```
#!/bin/bash

# Set environment variables for different stages if [ "$1" ==

"production" ]; then   export DB_HOST="prod-

db.example.com"   export API_KEY="prod-api-key" elif [

"$1" == "staging" ]; then   export DB_HOST="staging-

db.example.com"   export API_KEY="staging-api-key"

else

  export DB_HOST="dev-db.example.com"   export

API_KEY="dev-api-key"

fi
```

15. Error Handling and Alerts

This script checks logs for errors and sends a Slack notification if an error is found. #!/bin/bash

# Check logs for error messages and send Slack notification if grep -i "error"

/var/log/myapp.log; then

 curl -X POST -H 'Content-type: application/json' --data '{"text":"Error found in logs!"}' https://hooks.slack.com/services/your/webhook/url

fi

16. Automated Software Installation and Updates

This script installs Docker if it's not already installed on the system.

#!/bin/bash

# **Install Docker** if ! command -v docker &> /dev/null; then

curl -fsSL https://get.docker.com -o get-docker.sh sudo sh get-

docker.sh

fi

17. Configuration Management

This script updates configuration files (like nginx.conf) across multiple servers. #!/bin/bash

```
# Update nginx configuration across all servers scp nginx.conf
user@server:/etc/nginx/nginx.conf ssh user@server "systemctl
restart nginx"
```

---

18. Health Check Automation

This script checks the health of multiple web servers by making HTTP requests.

```
#!/bin/bash

# Check if web servers are running for server in "server1" "server2" "server3"; do   curl -s --head
http://$server | head -n 1 | grep "HTTP/1.1 200 OK" > /dev/null   if [ $? -ne 0 ]; then     echo
"$server is down"   else     echo "$server is up"

  fi

done
```

---

19. Automated Cleanup of Temporary Files

This script removes files older than 30 days from the /tmp directory to free up disk space.

```
#!/bin/bash


# Remove files older than 30 days in /tmp find /tmp -type f
-mtime +30 -exec rm -f {} \;
```

20. Environment Variable Management

This script sets environment variables from a .env file.

#!/bin/bash

# **Set environment variables from a .env file** export $(grep -v '^#'

.env | xargs)

21. Server Reboot Automation

This script automatically reboots the server during off-hours (between 2 AM and 4 AM).

#!/bin/bash

# **Reboot server during off-hours** if [$(date +%H) -ge 2] && [

$(date +%H) -lt 4]; then   sudo reboot

fi

22. SSL Certificate Renewal

This script renews SSL certificates using certbot and reloads the web server.

#!/bin/bash

# Renew SSL certificates using certbot certbot

renew systemctl reload nginx

---

23. Automatic Scaling of Containers

This script checks the CPU usage of a Docker container and scales it based on usage.

#!/bin/bash

# Check CPU usage of a Docker container and scale if necessary

CPU_USAGE=$(docker stats --no-stream --format "{{.CPUPerc}}" my-container | sed 's/%//') if (($(echo "$CPU_USAGE > 80" | bc -l))); then   docker-compose scale my-container=3

fi

---

24. Backup Verification

This script verifies the integrity of backup files and reports any corrupted ones. #!/bin/bash

# Verify backup files integrity for backup in

/backups/*.tar.gz; do   if ! tar -tzf $backup > /dev/null

2>&1; then   echo "Backup $backup is corrupted"

  else   echo "Backup $backup is valid"

  fi

done

25. Automated Server Cleanup

This script removes unused Docker images, containers, and volumes to save disk space.

```
#!/bin/bash


# Remove unused Docker images, containers, and volumes docker system prune -af
```

26. Version Control Operations

This script pulls the latest changes from a Git repository and creates a release tag. #!/bin/bash

```
# Pull latest changes from Git repository and create a release tag git pull origin

main git tag -a v$(date +%Y%m%d%H%M%S) -m "Release $(date)"

git push origin --tags
```

27. Application Deployment Rollback

This script reverts to the previous Docker container image if a deployment fails. #!/bin/bash

```
# Rollback to the previous Docker container image if deployment fails

if [ $? -ne 0 ]; then

  docker-compose down   docker-compose pull my-

app:previous   docker-compose up -d
```

fi

---

28. Automated Log Collection

This script collects logs from multiple servers and uploads them to an S3 bucket. #!/bin/bash

# Collect logs and upload them to an S3 bucket tar -czf

/tmp/logs.tar.gz /var/log/*

aws s3 cp /tmp/logs.tar.gz s3://my-log-bucket/logs/$(date

+%Y%m%d%H%M%S).tar.gz

---

29. Security Patch Management

This script checks for available security patches and applies them automatically.

#!/bin/bash

# Check and apply security patches sudo apt-get

update sudo apt-get upgrade -y --only-upgrade

---

30. Custom Monitoring Scripts

This script checks if a database service is running and restarts it if necessary.

#!/bin/bash

# Check if a database service is running and restart it if necessary if ! systemctl is-active --quiet mysql; then   systemctl restart mysql   echo "MySQL service was down and has been restarted" else   echo "MySQL service is running"

fi

---

31. DNS Configuration Automation (Route 53)

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

# Variables

```
ZONE_ID="your-hosted-zone-id"

DOMAIN_NAME="your-domain.com"

NEW_IP="your-new-ip-address"
```

# Update Route 53 DNS record

```
aws route53 change-resource-record-sets --hosted-zone-id $ZONE_ID --change-batch '{
  "Changes": [
    {
      "Action": "UPSERT",
      "ResourceRecordSet": {
        "Name": "'$DOMAIN_NAME'",
        "Type": "A",
        "TTL": 60,
        "ResourceRecords": [
          {
            "Value": "'$NEW_IP'"
          }
        ]
      }
    }
  ]
}'
```

32. Automated Code Linting and Formatting (ESLint and Prettier)

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

# Run ESLint

```
npx eslint . --fix
```

# Run Prettier npx prettier --write

"**/*.js"

33. Automated API Testing (Using curl)

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

```
# API URL

API_URL="https://your-api-endpoint.com/endpoint"


# Make GET request and check for 200 OK response

RESPONSE=$(curl --write-out "%{http_code}" --silent --output /dev/null
$API_URL)


if [ $RESPONSE -eq 200 ]; then   echo "API is up and running" else

echo "API is down. Response code: $RESPONSE"

fi
```

---

34. Container Image Scanning (Using Trivy)

```
#!/bin/bash

# Image to scan

IMAGE_NAME="your-docker-image:latest"


# Run Trivy scan trivy image --exit-code 1 --severity HIGH,CRITICAL $IMAGE_NAME

if [ $? -eq 1 ]; then

  echo "Vulnerabilities found in image: $IMAGE_NAME"

  exit 1 else   echo "No vulnerabilities found in image: $IMAGE_NAME"

fi
```

---

35. **Disk Usage Monitoring and Alerts (Email Notification)**

```
#!/bin/bash

# Disk usage threshold

THRESHOLD=80


# Get current disk usage percentage

DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')


# Check if disk usage exceeds threshold if [ $DISK_USAGE

-gt $THRESHOLD ]; then

  echo "Disk usage is above threshold: $DISK_USAGE%" | mail -s "Disk Usage Alert" your-
email@example.com

fi
```

---

36. **Automated Load Testing (Using Apache Benchmark)**

```
#!/bin/bash

# Target URL

URL="https://your-application-url.com"


# Run Apache Benchmark with 1000 requests and 10 concurrent requests ab -n 1000 -c 10 $URL
```

---

37. **Automated Email Reports (Server Health Report)**

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

# Server Health Report

REPORT=$(top -n 1 | head -n 10)

# Send report via email echo "$REPORT" | mail -s "Server Health Report" your-

email@example.com

---

38. DNS Configuration Automation (Route 53)

Introduction: This script automates the process of updating DNS records in AWS Route 53 when the IP address of a server changes. It ensures that DNS records are updated dynamically when new servers are provisioned.

#!/bin/bash

# Variables

ZONE_ID="your-hosted-zone-id"

DOMAIN_NAME="your-domain.com"

NEW_IP="your-new-ip-address"

# Update Route 53 DNS record

```
aws route53 change-resource-record-sets --hosted-zone-id $ZONE_ID --change-batch '{
  "Changes": [
    {
      "Action": "UPSERT",
      "ResourceRecordSet": {
        "Name": "'$DOMAIN_NAME'",
        "Type": "A",
        "TTL": 60,
        "ResourceRecords": [
         {
           "Value": "'$NEW_IP'"
         }
        ]
      }
    }
  ]
}'
```

39. Automated Code Linting and Formatting (ESLint and Prettier)

Introduction: This script runs ESLint and Prettier to check and automatically format JavaScript code before deployment. It ensures code quality and consistency.

```
#!/bin/bash

# Run ESLint
npx eslint . --fix
```

# **Run Prettier** npx prettier --write

"**/*.js"

---

40. Automated API Testing (Using curl)

Introduction: This script automates the process of testing an API by sending HTTP requests and verifying the response status. It helps ensure that the API is functioning correctly.

```
#!/bin/bash


# API URL

API_URL="https://your-api-endpoint.com/endpoint"


# Make GET request and check for 200 OK response

RESPONSE=$(curl --write-out "%{http_code}" --silent --output /dev/null

$API_URL)


if [ $RESPONSE -eq 200 ]; then   echo "API is up and running" else

echo "API is down. Response code: $RESPONSE"

fi
```

---

41. Container Image Scanning (Using Trivy)

Introduction: This script scans Docker images for known vulnerabilities using Trivy. It ensures that only secure images are deployed in production. #!/bin/bash

```
# Image to scan

IMAGE_NAME="your-docker-image:latest"


# Run Trivy scan trivy image --exit-code 1 --severity HIGH,CRITICAL $IMAGE_NAME

if [ $? -eq 1 ]; then

  echo "Vulnerabilities found in image: $IMAGE_NAME"

  exit 1 else   echo "No vulnerabilities found in image: $IMAGE_NAME"

fi
```

---

42. Disk Usage Monitoring and Alerts (Email Notification)

Introduction: This script monitors disk usage and sends an alert via email if the disk usage exceeds a specified threshold. It helps in proactive monitoring of disk space.

```
#!/bin/bash


# Disk usage threshold

THRESHOLD=80


# Get current disk usage percentage

DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')


# Check if disk usage exceeds threshold if [ $DISK_USAGE

-gt $THRESHOLD ]; then

  echo "Disk usage is above threshold: $DISK_USAGE%" | mail -s "Disk Usage Alert" your-
email@example.com
```

```
fi
```

43. Automated Load Testing (Using Apache Benchmark)

Introduction: This script runs load tests using Apache Benchmark (ab) to simulate traffic on an application. It helps measure the performance and scalability of the application.

bash

```
#!/bin/bash

# Target URL
URL="https://your-application-url.com"

# Run Apache Benchmark with 1000 requests and 10 concurrent requests ab -n 1000 -c 10 $URL
```

44. Automated Email Reports (Server Health Report)

Introduction: This script generates a server health report using system commands like top and sends it via email. It helps keep track of server performance and health.

```
#!/bin/bash

# Server Health Report
REPORT=$(top -n 1 | head -n 10)
```

# **Send report via email** echo "$REPORT" | mail -s "Server Health Report" your-

email@example.com

---

45. Automating Documentation Generation (Using pdoc for Python)

Introduction: This script generates HTML documentation from Python code using pdoc. It helps automate the process of creating up-to-date documentation from the source code.

#!/bin/bash

# **Generate documentation using pdoc** pdoc --html your-

python-module --output-dir docs/

# **Optionally, you can zip the generated docs** zip -r docs.zip

docs/

# List all cron jobs

crontab -l

# Edit cron jobs   crontab -e

# Remove all cron jobs   crontab -r

```
# Use a specific editor (e.g., nano)

EDITOR=nano crontab -e


# Cron Job Syntax

# * * * * * command_to_execute

# ┬ ┬ ┬ ┬ ┬

# | | | | |

# | | | | └── Day of the week (0-6, Sunday=0)

# | | | └──── Month (1-12 or JAN-DEC)

# | | └────── Day of the month (1-31)

# | └──────── Hour (0-23)

# └────────── Minute (0-59)


# Run a script every minute

* * * * * /path/to/script.sh


# Run a script every 5 minutes

*/5 * * * * /path/to/script.sh


# Run a script every 10 minutes

*/10 * * * * /path/to/script.sh
```

#

0

    Run a script at midnight

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

# Run a script every hour

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

# Run a script every 2 hours

0 */2 * * * /path/to/script.sh

# Run a script every Sunday at 3 AM

0 3 * * 0 /path/to/script.sh

# Run a script at 9 AM on the 1st of every month

0 9 1 * * /path/to/script.sh

# Run a script every Monday to Friday at 6 PM

0 18 * * 1-5 /path/to/script.sh

# Run a script on the first Monday of every month

0 9 * * 1 ["$(date +\%d)" -le 7] && /path/to/script.sh   Run a script on specific dates (e.g., 1st and 15th of the month)

    12 1,15 * * /path/to/script.sh

```
#

0

# Run a script between 9 AM and 5 PM, every hour

0 9-17 * * * /path/to/script.sh


# Run a script every reboot

@reboot /path/to/script.sh


# Run a script daily at midnight

@daily /path/to/script.sh


# Run a script weekly at midnight on Sunday

@weekly /path/to/script.sh


# Run a script monthly at midnight on the 1st

@monthly /path/to/script.sh


# Run a script yearly at midnight on January 1st

@yearly /path/to/script.sh
   Redirect cron job output to a log file

   0 * * * /path/to/script.sh >> /var/log/script.log 2>&1


# Run a job only if the previous instance is not running
```

```
#

0

0 * * * * flock -n /tmp/job.lock /path/to/script.sh


# Run a script with a random delay (0-59 minutes)

RANDOM_DELAY=$((RANDOM % 60)) && sleep $RANDOM_DELAY &&
/path/to/script.sh


# Run a script with environment variables

SHELL=/bin/bash

PATH=/usr/local/bin:/usr/bin:/bin

0 5 * * * /path/to/script.sh


# Check cron logs (Ubuntu/Debian)   grep CRON

/var/log/syslog


# Check cron logs (Red Hat/CentOS)   grep CRON

/var/log/cron


# Restart cron service (Linux)
```

sudo systemctl restart cron

# Check if cron service is running   sudo

systemctl status cron
