**DevOps Intern Technical Test**

**Section 1: Scripting and Automation**

1. Create a folder named DevOps:

* Open your file manager.
* Navigate to the location where you want to create the folder (e.g., your home directory).
* Right-click and select "New Folder.

2. Create a Backup folder for gzip:

* Follow the same steps as above but create a folder named "Backup" instead of "DevOps".

3. Create backup\_script.sh on Desktop:

* Open a text editor.
* Write script.

Bash

---------------------------------------------------------------------------------------------------------------------

#!/bin/bash

# Function to print usage instructions

usage() {

echo "Usage: $0 <directory\_to\_backup> <backup\_directory>"

exit 1

}

# Check if the correct number of arguments is provided

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

echo "Error: Invalid number of arguments."

usage

fi

# Assigning arguments to variables

DIR\_TO\_BACKUP="$1"

BACKUP\_DIR="$2"

# Check if the directory to back up exists

if [ ! -d "$DIR\_TO\_BACKUP" ]; then

echo "Error: The directory to back up does not exist: $DIR\_TO\_BACKUP"

exit 1

fi

# Check if the backup directory exists

if [ ! -d "$BACKUP\_DIR" ]; then

echo "Error: The backup directory does not exist: $BACKUP\_DIR"

exit 1

fi

# Get the current date in YYYY-MM-DD format

CURRENT\_DATE=$(date +%Y-%m-%d)

# Create the backup file name with the current date

BACKUP\_FILE\_NAME="backup\_$(basename "$DIR\_TO\_BACKUP")\_$CURRENT\_DATE.tar.gz"

# Create the backup

tar –czf "BACKUP\_FILE\_NAME" -C "$(dirname "$DIR\_TO\_BACKUP")" "$(basename "$DIR\_TO\_BACKUP")"

# Check if the backup was successful

if [ $? -ne 0 ]; then

echo "Error: Failed to create the backup."

exit 1

fi

#Move the backup file to the backup directory

mv "$BACKUP FILE NAME "SBACKUP\_DIR”

if [ $? -ne 0 ]; then

echo "Error: Failed to move the backup file to the backup directory."

exit 1

ft

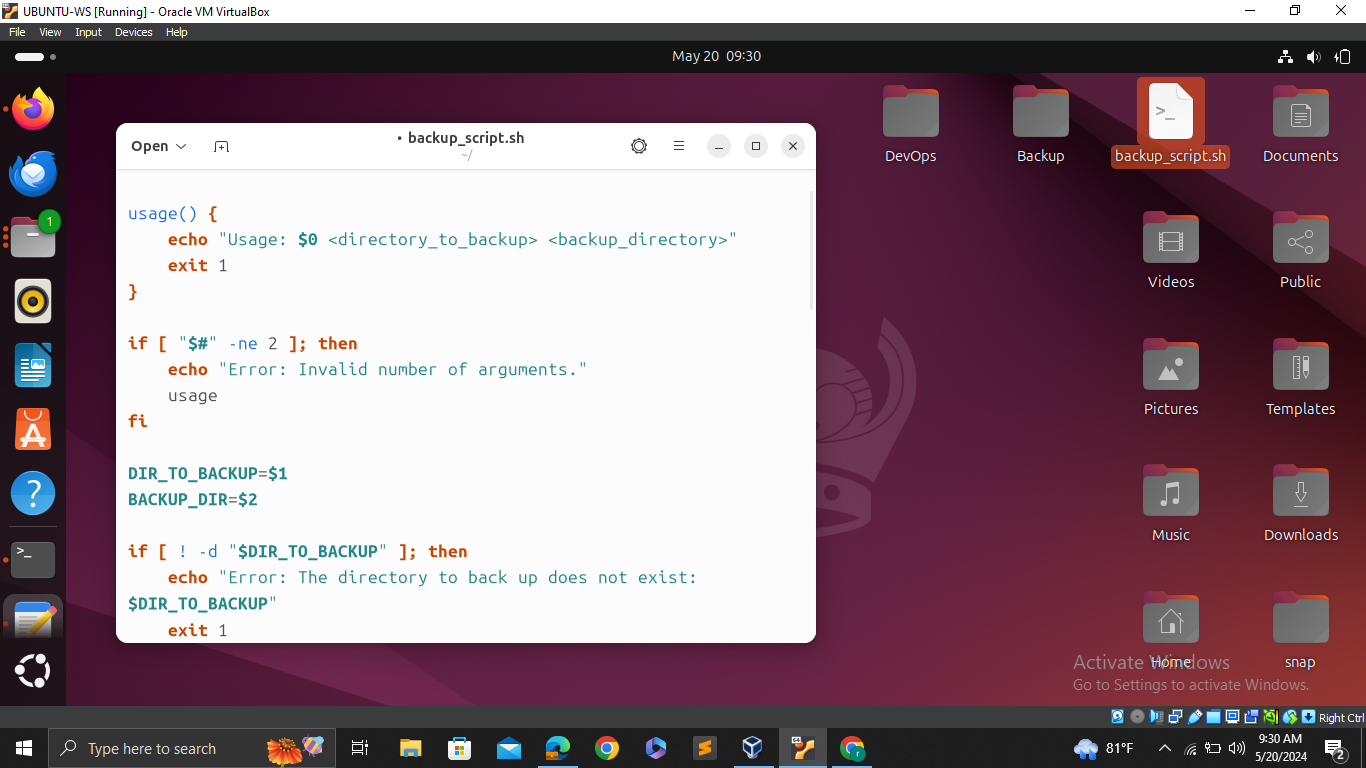
# Print success message

echo "Backup created and moved to $BACKUP\_DIR/$BACKUP\_FILE\_NAME successfully."

exit 0

**-------------------------------------------------------------------------------------------------------------------------------**

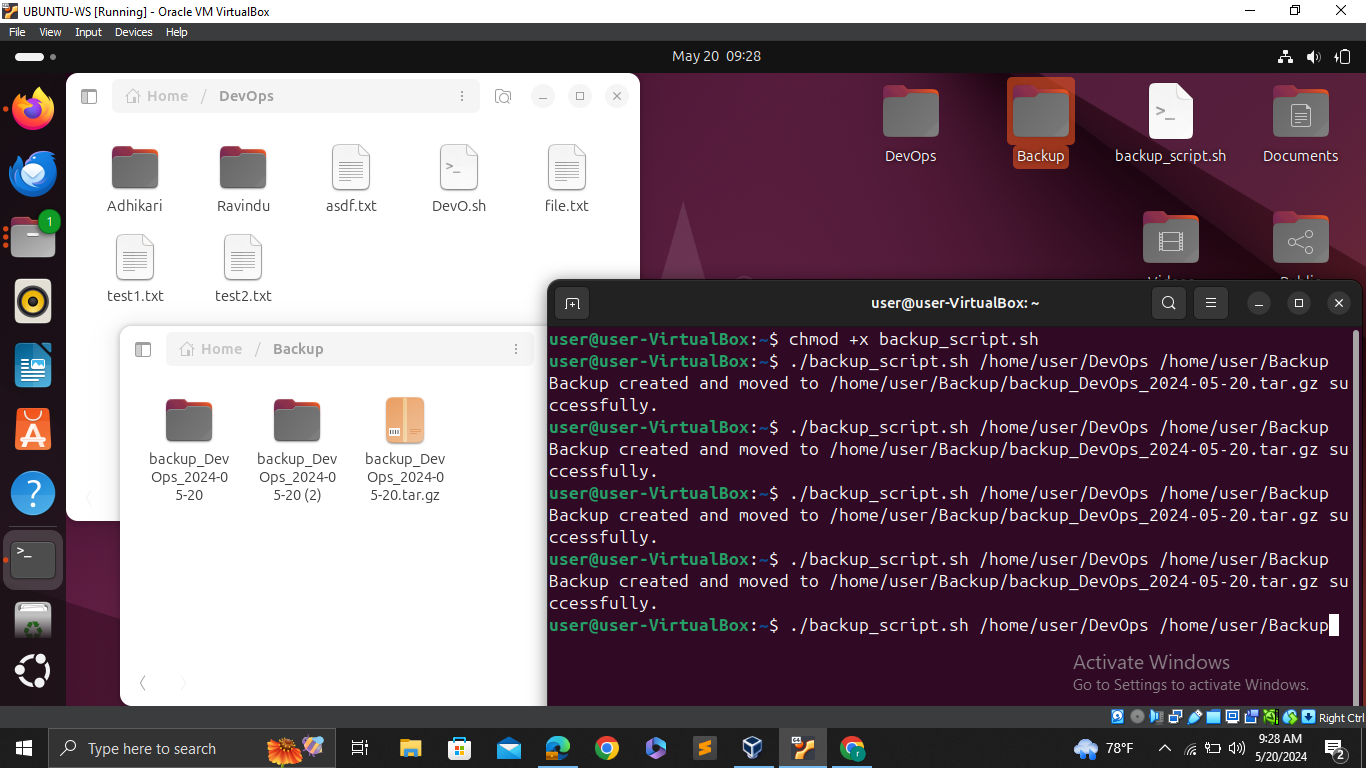
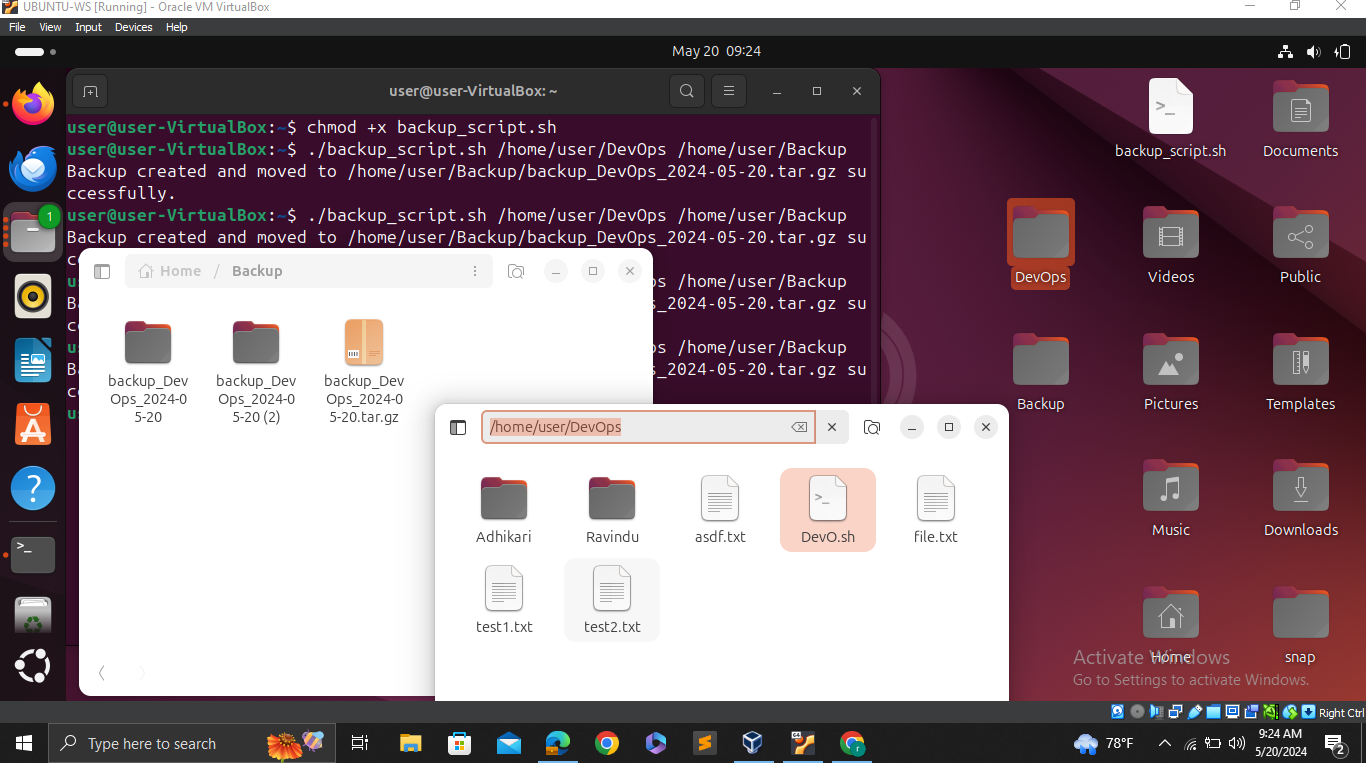
* Save the script with the name "backup\_script.sh" on Desktop.



1. Script executable

* Set the script's permissions to make it executable using the command in the terminal.

chmod +x backup\_script.sh



1. Run the Script

./backup\_script.sh /home/user/DevOps /home/user/Backup

**Section 2: Version Control (Git)**

1. Clone a repository from GitHub

* The first step is to clone a repository from GitHub to local machine.
* . command used to clone a repository.

“git clone https://github.com/[Miyuranga305](https://github.com/Miyuranga305)/Cafe-Management-System1.git”

* + “git clone” is the command used to clone a repository.
  + Provide the URL of the GitHub repository want to clone after the “git clone” command.
* This command will download all the files and history of the repository to local machine.

2. Create a new branch called “feature-branch”

Branches are used to work on new features or make changes to a project without affecting the main codebase.

* Change directory to the cloned repository

“cd Cafe-Management-System1”

* Command to create a new branch named `feature-branch` and switch to it.

“git checkout -b feature-branch”

* The “-b” flag indicates that you want to create a new branch.
* This command allows to isolate changes from the main branch until they are ready

3. Make a change to a file and commit the change to the new branch

* That on the “feature-branch”, can make changes to the project.
* First, make a change to a file. This could be adding, modifying, or deleting content in a file.

“echo "Some changes" >> file.txt”

* stages the changes made to the file. Staging means marking the changes as ready to be committed.

“git add file.txt”

* commits the changes to the repository with a descriptive message.

“git commit -m "Add some changes to file.txt"”

* Committing saves changes permanently to the branch's history.

4. Merge the `feature-branch` back into the `main` branch

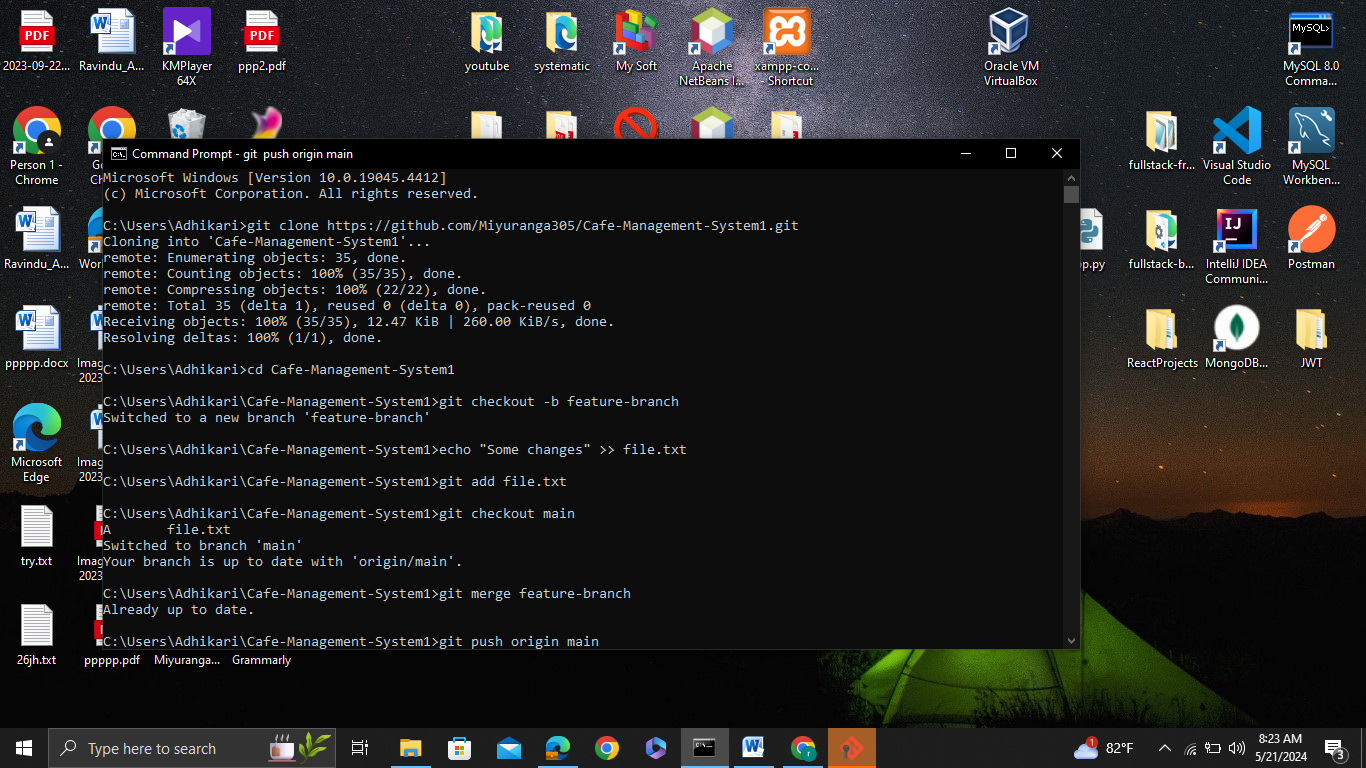
* Completed work on the “feature-branch” and satisfied with the changes, merge it back into the “main” branch.
* Switches to the “main” branch.

“git checkout main”

* Merges the changes from “feature-branch” into the “main” branch.

“git merge feature-branch”

* Merging integrates the changes from the feature branch into the main branch's history.



5. Push the changes to the remote repository

* After merging the changes into the “main” branch locally, you need to push those changes to the remote repository on GitHub.
* Pushes the changes from your local “main” branch to the remote repository's “main” branch on GitHub.

“git push origin main”

* “origin” is the default name Git gives to the remote repository from which you cloned.
* Pushing updates the remote repository with local changes, making them accessible to others and ensuring work is backed up.

**Section 3: Continuous Integration / Continuous Deployment (CI/CD)**

Step 1: Create the Workflow Directory and File

1. Create a directory for workflows:

* In your repository, create a new directory named “.github/workflows”. This is where GitHub Actions looks for workflow files.

bash - “mkdir -p .github/workflows”

2. Create the workflow file:

* Inside the “.github/workflows” directory, create a file named “main.yml”.

bash - “touch .github/workflows/main.yml”

Step 2: Define the Workflow Trigger

* Specify the trigger for the workflow:

In “main.yml”, define that the workflow should be triggered on every push to the “main” branch.

yaml

name: CI/CD Pipeline

on:

push:

branches:

- main

* “name” : Sets a name for the workflow for easy identification.
* “on”: Specifies the event that triggers the workflow. Here, it is triggered by a `push` event to the `main` branch.

Step 3: Define the Job and Runner

* Set up the job and specify the runner:

Define the job that will run the steps and specify the runner environment (“ubuntu-latest”).

jobs:

build:

runs-on: ubuntu-latest

* “jobs” : Contains a list of jobs to be run. Here, we have one job named “build”.
* “runs-on” : Specifies the virtual environment for the job. “ubuntu-latest” is a commonly used Linux environment.

Step 4: Define the Steps in the Job

1. Check out the code:

* Add a step to check out the repository's code using “actions/checkout”.

steps:

- name: Check out code

uses: actions/checkout@v3

* “steps” : Lists all the steps for the job.
* “name” : Provides a name for the step.
* “uses” : Specifies an action to use. “actions/checkout@v3” checks out the repository's code.

2. Set up the Python environment:

* Add a step to set up a Python environment using “actions/setup-python”. yaml -

- name: Set up Python

uses: actions/setup-python@v4

with:

python-version: '3.x'

* “uses” : Specifies the `actions/setup-python` action to set up Python.
* “with” : Provides parameters to the action. `python-version: '3.x' specifies the version of Python to set up

3. Install dependencies:

* Add a step to install dependencies listed in “requirements.txt”. yaml -

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

* “run” : Executes shell commands. Here, it upgrades “pip” and installs the dependencies from “requirements.txt”.

4. Run tests using pytest:

* Add a step to run tests with “pytest”. Yaml -

- name: Run tests

run: |

pytest

* “run” : Executes the `pytest` command to run tests.

Complete `main.yml` File

name: CI/CD Pipeline

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Check out code

uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v4

with:

python-version: '3.x' # Specify the Python version you need

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Run tests

run: |

pytest

**Section 6: Troubleshooting**

**1) List the possible causes of the application downtime**

1. Server Issues

* Server Crash
* Overloaded Server
* Hardware Failure

2. Network Issues

* Network Outage
* DNS Problems

3. Software Issues

* Application Bugs
* Database Problems
* Dependency Failures

4. Security Issues

* DDoS Attack
* Hacking or Malware

5. Configuration Issues

* Configuration Changes
* Expired Certificates

6. Resource Exhaustion

* Memory Leaks
* Disk Space

7. Service Outages

* Cloud Service Issues
* Service Dependencies

8. Maintenance or Updates

* Scheduled Maintenance
* Updates and Patches

**2) Explain the diagnostic steps you would take to identify the root cause.**

1. Verify the Issue

* Check Monitoring Systems - Look at the monitoring dashboards (e.g., New Relic, Datadog) to confirm the downtime and gather initial clues.
* Access Logs - Review the server logs, application logs, and access logs for any errors or unusual activity around the time of the downtime.
* User Reports - Verify if the downtime is being experienced by all users or a subset (e.g., specific geographies, user groups).

2. Check Server Health

* Ping the Server - Use tools like `ping` or `traceroute` to check if the server is reachable.
* Server Resources - Check CPU, memory, disk space, and I/O statistics using tools like `top`, `htop`, `vmstat`, `df`, etc.
* Service Status - Verify that all necessary services (e.g., web server, database server) are running properly using commands like `systemctl status` or `service status`.

3. Network Diagnostics

* DNS Resolution - Ensure the domain is resolving correctly using `nslookup` or `dig`.
* Network Connectivity - Use `netstat` to check for open ports and `telnet` or `nc` to test connectivity to the server.

4. Application Layer Checks

* Check Application Logs - Look for exceptions, errors, or stack traces in the application logs.
* Recent Changes - Identify any recent code changes, deployments, or configuration changes that might have caused the issue.
* Dependency Checks - Verify the status of external dependencies and third-party services/APIs.

5. Database Checks

* Database Connectivity - Ensure that the application can connect to the database.
* Query Performance - Check for slow or failing queries in the database logs.
* Database Health - Look for database-specific issues such as table locks, high resource usage, or storage issues.

6. Security Assessments

* DDoS Attack - Check for unusual traffic patterns or spikes that might indicate a DDoS attack.
* Intrusion Detection - Review security logs and alerts for signs of hacking or malware.
* Certificate Validity - Ensure that SSL/TLS certificates are valid and not expired.

7. Configuration and Updates

* Configuration Files - Review recent changes to configuration files for the application, web server, and database.
* Rollback Updates - If a recent update or patch is suspected, consider rolling back to a previous stable version.

8. Consult with Cloud Provider

* Cloud Services Status - Check the status page of your cloud provider (e.g., AWS, Azure, Google Cloud) for any reported outages or issues.
* Support Tickets - Open a support ticket with the cloud provider if an outage or issue is suspected on their end.

9. Communication and Documentation

* Incident Communication - Keep stakeholders informed about the progress of the troubleshooting efforts.
* Document Findings - Document all findings, actions taken, and their outcomes for future reference and post-mortem analysis.

**3) Describe the actions you would take to resolve the issue and restore service.**

1. Server Issues

Server Crash or Overloaded Server

* Restart Server - If the server has crashed or is unresponsive, restart it.
* Scale Resources - If the server is overloaded, consider scaling up (increasing resources like CPU and RAM) or scaling out (adding more servers to distribute the load).
* Auto-scaling Configuration - Ensure that auto-scaling policies are correctly configured to handle traffic spikes.

Hardware Failure

* Replace Hardware - If a hardware component has failed, replace the faulty hardware.
* Migrate to New Server - If hardware replacement is not immediately possible, migrate the application to a backup server or a new server.

2. Network Issues

Network Outage

* Contact ISP - If the issue is with the internet service provider, contact them for resolution.
* Check Internal Network - Ensure that all internal network components like routers and switches are functioning properly.

DNS Problems

* Fix DNS Configuration - Update the DNS records if there are misconfigurations.
* Use Alternative DNS - Temporarily use an alternative DNS provider or service.

3. Software Issues

Application Bugs

* Roll Back Changes - Roll back to a previous stable version if a recent deployment introduced bugs.
* Fix and Redeploy - Identify and fix the bugs, then redeploy the application.

Database Problems

* Restore Database - Restore from the most recent backup if the database is corrupted.
* Optimize Queries - If the database is slow, optimize the problematic queries and indexes.
* Increase Resources - Scale up database resources if they are insufficient.

Dependency Failures

* Switch to Backup - If a third-party service is down, switch to a backup or alternative service if available.
* Retry Logic - Implement retry logic in the application to handle temporary failures of dependencies.

4. Security Issues

DDoS Attack

* Activate DDoS Protection - Use DDoS protection services like Cloudflare or AWS Shield.
* Rate Limiting - Implement rate limiting to mitigate the impact of the attack.

Hacking or Malware

* Isolate the Server - Isolate the affected server to prevent further damage.
* Scan and Clean - Use security tools to scan and remove any malware.
* Patch Vulnerabilities - Apply security patches and updates to fix vulnerabilities.

5. Configuration Issues

Configuration Changes

* Revert Changes - Revert to previous configurations if recent changes caused the issue.
* Validate Configuration - Ensure all configurations are correct and there are no conflicts.

Expired Certificates

* Renew Certificates - Renew and install the SSL/TLS certificates.
* Automate Renewal - Set up automated certificate renewal to prevent future expirations.

6. Resource Exhaustion

Memory Leaks

* Restart Services -Restart the affected services to free up memory.
* Fix Memory Leaks - Identify and fix memory leaks in the application code.

Disk Space

* Free Up Space - Delete unnecessary files and logs to free up disk space.
* Increase Disk Size - Expand the disk size if space is consistently an issue.

7. Service Outages

Cloud Service Issues

* Switch Regions - If the cloud provider has a regional outage, switch to another region.
* Multi-Cloud Strategy - Consider a multi-cloud strategy to avoid single points of failure.

8. Maintenance or Updates

Scheduled Maintenance

* Notify Users - Inform users about the maintenance window and expected downtime.
* Ensure Completion - Make sure that maintenance tasks are completed as planned and all services are restarted properly.

Updates and Patches

* Rollback Updates - If updates caused issues, roll back to the previous stable version.
* Thorough Testing - Before applying future updates, ensure thorough testing in a staging environment.

Communication and Monitoring

* Inform Stakeholders - Keep all stakeholders informed about the issue status and expected resolution time.
* Post-Mortem Analysis - After restoring service, conduct a post-mortem analysis to understand the cause and prevent future occurrences.
* Enhanced Monitoring - Improve monitoring and alerting systems to catch similar issues early.

**Section 5: Monitoring and Logging**

1. **Explain which tools you would use for monitoring and logging**

Monitoring Tools

1.Prometheus

* An open-source monitoring and alerting toolbox with a focus on scalability and dependability is called Prometheus. It gathers metrics from targets that have been specified at certain intervals, assesses rule expressions, shows the outcomes, and can sound an alarm when specific parameters are satisfied.It provides strong querying capabilities with PromQL, is very compatible with Kubernetes, and connects well with other tools.

2. Grafana

* Prometheus is one of the many data sources that Grafana, an open-source analytics and monitoring platform, uses.It makes it simpler to analyse metrics and keep an eye on the application's health by offering comprehensive visualisations and dashboards.

3. Kube-state-metrics

* Kube-state-metrics is a simple service that gathers metrics on the object's status by listening to the Kubernetes API server.provide precise analytics on the condition of nodes, pods, and deployments all of which are Kubernetes objects.

4. Alertmanager

* Prometheus alerts are handled by Alertmanager, which enables routing to different notification channels, aggregation, and deduplication.makes sure the appropriate individuals are informed and that notifications are handled efficiently to avoid alert fatigue.

Logging Tools

1.Fluentd

* You may gather logs from numerous sources and route them to different destinations using Fluentd, an open-source data collector for unified logging layers.It is extensively utilised in the Kubernetes ecosystem, is quite flexible, and supports a variety of input and output plugins.

2. Elasticsearch

* Elasticsearch is a distributed, RESTful analytics and search engine that can store and explore massive volumes of log data in almost real-time.offers strong search features and works well with Kibana to visualise logs.

3. Kibana

* For log and time-series analytics, Kibana is an open-source application for data visualisation and analysis.makes it simple to view and examine logs kept in Elasticsearch.

4. Fluent Bit

* Fluent Bit is a lightweight log processor and forwarder designed for Kubernetes environments.More lightweight compared to Fluentd, making it suitable for environments where resource usage is a concern.

1. **Provide a high-level overview of how you would configure these tools.**

Configuring monitoring and logging tools for a web application running on a Kubernetes cluster involves several steps. Here's a high-level overview of how to configure Prometheus, Grafana, kube-state-metrics, Alertmanager, Fluentd, Elasticsearch, Kibana, and Fluent Bit:

Monitoring Configuration

1. Prometheus

Install Prometheus -Use Helm to install Prometheus:

bash

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo update

helm install prometheus prometheus-community/prometheus

Ensure the Prometheus server is up and running and can scrape metrics from your Kubernetes cluster.

Configure Scraping - Define scraping targets in the `prometheus.yml` configuration file to collect metrics from Kubernetes nodes, pods, and services.

2. Grafana

Install Grafana -Use Helm to install Grafana:

bash

helm repo add grafana https://grafana.github.io/helm-charts

helm repo update

helm install grafana grafana/grafana

Access the Grafana dashboard and set up Prometheus as a data source.

Create Dashboards - Import pre-built dashboards or create custom dashboards to visualize the metrics collected by Prometheus.

3. Kube-state-metrics

Install Kube-state-metrics - Use Helm to install kube-state-metrics:

bash

helm install kube-state-metrics prometheus-community/kube-state-metrics

Configure Prometheus to scrape metrics from kube-state-metrics by adding it to the scraping targets.

4. Alertmanager

Install Alertmanager - Use Helm to install Alertmanager:

bash

helm install alertmanager prometheus-community/alertmanager

Configure Alertmanager rules and notification channels in the `alertmanager.yml` file.

Logging Configuration

1. Fluentd

Install Fluentd - Use Helm to install Fluentd as a DaemonSet:

bash

helm repo add fluent https://fluent.github.io/helm-charts

helm repo update

helm install fluentd fluent/fluentd

Configure Fluentd to collect logs from Kubernetes nodes and containers and forward them to Elasticsearch.

2. Elasticsearch

Install Elasticsearch - Use Helm to install Elasticsearch:

bash

helm repo add elastic https://helm.elastic.co

helm repo update

helm install elasticsearch elastic/elasticsearch

Ensure Fluentd is configured to send logs to Elasticsearch using the appropriate Fluentd output plugin.

3. Kibana

Install Kibana - Use Helm to install Kibana:

bash

helm install kibana elastic/kibana

Connect Kibana to the Elasticsearch instance by configuring the Elasticsearch endpoint in Kibana settings.

Create Dashboards

Set up and customize Kibana dashboards to visualize the logs collected by Elasticsearch.

4. Fluent Bit

Install Fluent Bit - Use Helm to install Fluent Bit as a DaemonSet:

bash

helm install fluent-bit fluent/fluent-bit

Configure Fluent Bit to collect logs and forward them to Fluentd or directly to Elasticsearch.

**Section 2: Version Control (Git)**

1. Clone a Repository from GitHub

Command

bash

git clone https://github.com/ravindu/repository.git

Explanation:

* git clone`: This command creates a copy of an existing repository from a specified URL to your local machine.
* ‘https://github.com/ravindu/repository.git`: This is the URL of the repository you want to clone from GitHub.

2. Create a New Branch Called `feature-branch`

Command

bash

cd repository

git checkout -b feature-branch

Explanation:

* `cd repository`: Changes the current directory to the cloned repository.
* `git checkout -b feature-branch`: Creates a new branch named `feature-branch` and switches to it. The `-b` flag tells Git to create the branch if it does not already exist.

3. Make a Change to a File and Commit the Change to the New Branch

Commands:

bash

echo "Some changes" >> file.txt

git add file.txt

git commit -m "Add some changes to file.txt"

Explanation:

* `echo "Some changes" >> file.txt`: Adds the text "Some changes" to the file `file.txt`. This simulates making a change to the file.
* `git add file.txt`: Stages the changed file `file.txt` for the next commit.
* `git commit -m "Add some changes to file.txt"`: Commits the staged changes with a message describing the changes. The `-m` flag allows you to add a commit message directly from the command line.

4. Merge the `feature-branch` Back into the `main` Branch

Commands:

bash

git checkout main

git merge feature-branch

Explanation

* `git checkout main`: Switches to the `main` branch.
* `git merge feature-branch`: Merges the changes from `feature-branch` into the `main` branch.

5. Push the Changes to the Remote Repository

Command:

bash

git push origin main

Explanation:

* `git push origin main`: Pushes the commits from the local `main` branch to the `main` branch on the remote repository named `origin`. `origin` is the default name given to the remote repository when it's cloned.