DISASTER RECOVERY WITH IBM CLOUD VIRTUAL SERVERS

PROJECT OVERVIEW :

The project titled 'Disaster Recovery with IBM Cloud Virtual Servers' aims to establish a robust and reliable disaster recovery solution utilizing IBM Cloud's virtual server infrastructure. This project's primary goal is to ensure the continuity of critical business operations in the face of unexpected disruptions, such as natural disasters or hardware failures. Key components include data replication, failover mechanisms, and automated recovery processes, all leveraging IBM Cloud's scalable and secure virtual server environment. By implementing this disaster recovery strategy, organizations can minimize downtime, protect valuable data, and maintain business resilience in the event of unforeseen crises. Project Objectives

DEFINITION :

Disaster recovery (DR) is a crucial aspect of any business continuity plan, and it involves ensuring the rapid recovery of IT systems and data in the event of a disaster or disruptive event. IBM Cloud provides a range of services.

STEPS FOR BULIDING DISASTER RECOVERY PLAN BY CONFIGURING REPLICATION AND TESTING PROCEDURES :

Creating a disaster recovery plan involves several steps, including configuring replication and testing recovery procedures.

The specific code for this process will depend on the technologies and infrastructure .

Step 1: Configuring Replication

Assuming you're using a database as an example, let's consider setting up replication between a primary and secondary server.

1) Set up Primary Server (e.g., MySQL)

CODE:

 # On Primary Server

CREATE USER 'replication_user'@'%' IDENTIFIED BY 'your_password';

GRANT REPLICATION SLAVE ON *.* TO 'replication_user'@'%'; **#Take a backup of the database** mysqldump -u root -p

your_database >

your_database_backup.sql;

 # Get the current binary log file and position

SHOW MASTER STATUS;

2) Set up Secondary Server (e.g., MySQL)

#On Secondary Server

STOP SLAVE;

CHANGE MASTER TO MASTER_HOST='primary_server_ip',

MASTER_USER='replication_user',

MASTER_PASSWORD='your_password',

MASTER_LOG_FILE='log_file_from_primary',

MASTER_LOG_POS=log_position_from_primary;

START SLAVE;

Step 2: Testing Recovery Procedures

1) Simulate Disaster (e.g., Failover)

 # On Primary Server (simulate failure)

service mysql stop;

#Promote the Secondary Server to become the new Primary Server.

 # Update application configurations to point to the new Primary Server.

2) Validate Data Consistency

 # On the new Primary Server (formerly Secondary)

SHOW SLAVE STATUS;

 # Ensure replication is running.

Step 3: Set Up Replication

Configure replication for critical systems and data to ensure real-time or periodic backups. The method of replication may depend on your infrastructure (e.g., database replication, file system replication, etc.).

Code:

Example for Database Replication (using SQL Server):

 # Enable replication on the database

USE master;

EXEC sp_replicationdboption 'YourDatabase', 'publish', true;

 # Create a publication

```
EXEC sp_addpublication @publication = 'YourPublication';
```

 # Add articles to the publication

```
EXEC sp_addarticle @publication = 'YourPublication', @article = 'Table1', @source_table = 'dbo.Table1', @type = 'logbased';
```

 # Create a subscription

```
EXEC sp_addsubscription @publication = 'YourPublication', @subscriber = 'YourSubscriber';
```

 # Start the snapshot agent to initialize the subscription

```
EXEC sp_startpublication_snapshot @publication = 'YourPublication';
```

Step 4: Automate Recovery Scripts

Automate recovery scripts to streamline the recovery process. This may include scripts to restore databases, deploy applications, or configure network settings.

Code:

Example for Database Restore (using SQL Server):

 # Restore the database from backup

RESTORE DATABASE YourDatabase FROM DISK = 'C:\Backup\YourDatabase.bak' WITH REPLACE;

Step 5: Conduct Regular Tests

Regularly test the disaster recovery plan to ensure its effectiveness. This involves simulating a disaster scenario and executing the recovery procedures.

Code:

Example Recovery Test Script (using PowerShell):

 # Example script to restore a database backup

 # Set variables

$backupPath = "C:\Backup\YourDatabase.bak"
$databaseName = "YourDatabase"

 # Stop application services or any connections to the database

 # Execute the restore command

Restore-SqlDatabase -ServerInstance "YourSQLServer" -Database $databaseName -BackupFile $backupPath –Replace

 # Start application services or restore connections

Step 6 :Monitor and Update

Regularly monitor the health of the replication and recovery procedures.

STEPS TO IMPLEMENT REPLICATION OF DATA AND VIRTUAL MACHINE IMAGES FROM ONPREMISES TO IBM CLOUD VIRTUAL SERVERS:

Step 1: Set Up IBM Cloud Virtual Servers

1. Create IBM Cloud Account: If you don't have an IBM Cloud account, sign up for one.
2. Provision Virtual Servers: Use the IBM Cloud Console to provision virtual servers in the desired region.

Step 2: Configure On-Premises Replication Software

Choose a replication tool that fits your requirements. IBM offers solutions like IBM

Spectrum Protect for data backup and replication.

1. **Install and Configure Replication Software:** Install the replication software on your onpremises servers and configure it to replicate data and VM images to IBM Cloud Virtual Servers.

Step 3: Testing Code for Replication Verification

Create a script to test the replication process and verify that data is being replicated accurately. Below is a basic example using Python and the IBM Cloud API for Virtual Servers.

PYTHON CODE:

```
# Install the required Python libraries

# pip install requests

import requests

import json import

time
```

```
api_key = "your_ibm_cloud_api_key"
api_endpoint = "https://api.cloud.ibm.com"
# Virtual Server details server_id =
"your_virtual_server_id" # Function to
check the replication status def
check_replication_status():
url=f"{api_endpoint}/v1/instances/{server_id}/status"
    headers = {"Authorization": f"Bearer {api_key}"}
response = requests.get(url, headers=headers)
    if response.status_code == 200:
        return response.json()
else:
        return None
# Function to wait for replication completion
def wait_for_replication_completion():
```

```
    max_retries = 30
interval_seconds = 60     for _
in range(max_retries):
        status = check_replication_status()        if
status and status["status"] == "replicated":
print("Replication completed successfully.")
            return
        time.sleep(interval_seconds)
    print("Replication did not complete within the
expected time.")
```

 # Trigger replication and wait for completion

```
def initiate_replication():
url=f"{api_endpoint}/v1/instances/{server_id}/rep
licate"
headers = {"Authorization": f"Bearer {api_key}",
"Content-Type": "application/json"}
payload = {
```

```
        "source_instance_id":
"your_source_instance_id",
        "source_region": "your_source_region",
    }
    response = requests.post(url, headers=headers,
data=json.dumps(payload))     if
response.status_code == 202:
        print("Replication initiated successfully.")
wait_for_replication_completion()
    else:
        print(f"Failed to initiate replication. Status
code: {response.status_code}")

# Main execution if
__name__ == "__main__":
    initiate_replication()
```

This is the testing code for replication verification.

STEPS TO CONDUCT RECOVERY TESTS TO ENSURE THAT THE DISASTER RECOVERY PLAN WORKS AS INTENDED. SIMULATE A DISASTER SCENARIO AND PRACTICE RECOVERY PROCEDURES:

Step 1: Disaster Simulation

Simulate a disaster scenario by intentionally causing a failure in a controlled environment. This could involve:

- Simulating hardware failure.
- Temporarily disconnecting network connections.
- Deleting critical files or data.

Step 2: Practice Recovery Procedures

Execute the recovery procedures outlined in your disaster recovery plan. This may include:

- Restoring data from backups.

- Activating standby systems.
- Rebuilding virtual machines.
- Redirecting traffic to alternative servers.

Step 3: Testing Code for Recovery Verification

Create a script to automate the verification of the recovery process. This script should check the status of recovered systems, data integrity, and the overall health of the IT infrastructure. Below is a basic example using Python:

Code:

 # Install the required Python libraries

 # pip install requests

import requests

import json **import**

time

```
# IBM Cloud API key and endpoint api_key
= "your_ibm_cloud_api_key" api_endpoint
= "https://api.cloud.ibm.com"
# Virtual Server details server_id =
"your_virtual_server_id"
# Function to check the recovery status def
check_recovery_status():
    url =
f"{api_endpoint}/v1/instances/{server_id}/status"
headers = {"Authorization": f"Bearer {api_key}"}
response = requests.get(url, headers=headers)    if
response.status_code == 200:
        return response.json()
else:
        return None
```

```
# Function to wait for recovery completion def
wait_for_recovery_completion():
    max_retries = 30
interval_seconds = 60     for _
in range(max_retries):
        status = check_recovery_status()         if
status and status["status"] == "active":
print("Recovery completed successfully.")
            return
        time.sleep(interval_seconds)

    print("Recovery did not complete within the
expected time.")
# Trigger recovery and wait for completion def
initiate_recovery():
    url =
```

```
f"{api_endpoint}/v1/instances/{server_id}/recover"
    headers = {"Authorization": f"Bearer {api_key}", "Content-Type": "application/json"}
payload = {
        "source_instance_id": "your_source_instance_id",
        "source_region": "your_source_region",
    }
    response = requests.post(url, headers=headers, data=json.dumps(payload))     if response.status_code == 202:
        print("Recovery initiated successfully.")
wait_for_recovery_completion()
    else:
        print(f"Failed to initiate recovery. Status code: {response.status_code}")
```

```
# Main execution if

__name__ == "__main__":

    initiate_recovery().
```

This is the testing code for recovery.

CONCLUSION:

This script initiates the recovery process and checks the recovery status until it is completed. After running this script, validate that the recovered systems are functioned and that data integrity is maintained.