Title: Disaster Recovery With IBM Cloud Virtual Server

PHASE – 4

DEVELOPMENT PART – II

Building a disaster recovery plan involves setting up data replication between primary and secondary sites, and thoroughly testing the recovery procedures. In this example, I'll demonstrate a basic setup using MySQL database replication, and then outline steps for testing the recovery procedures.

Disaster Recovery Plan Outline:

Database Replication:

a. Primary Site:

Set up MySQL server as the primary database.

Enable binary logging and configure server ID.

Create a replication user with appropriate privileges.

b. Secondary Site:

Set up MySQL server as the secondary database.

Configure it as a replica of the primary server.

Start the replication process.

c. Monitoring and Maintenance:

Implement monitoring for replication status.

Regularly monitor the replication lag.

Sample MySQL Replication Configuration:

Primary Server (my.cnf):

pythonCopy code

[mysqld] server-id=1 log-bin=mysql-bin binlog-do-db=my_database

Secondary Server (my.cnf):

[mysqld] server-id=2 relay-log=mysql-relay-bin replicate-do-db=my_database

Create Replication User:

CREATE USER 'replication_user'@'%' IDENTIFIED BY 'password'; GRANT REPLICATION SLAVE ON *.* TO 'replication_user'@'%'; FLUSH PRIVILEGES;

Testing Recovery Procedures:

a. Planned Failover:

Simulate a controlled failover from primary to secondary.

Stop the primary database.

Promote the secondary to the primary role.

Update application configurations to point to the new primary.

b. Unplanned Failover:

Simulate an unexpected failure in the primary site.

Manually trigger the failover process.

Monitor the failover process to ensure it completes successfully.

c. Data Validation:

Verify that data on the secondary site matches the primary.

Use tools like checksums or manual spot-checking.

d. Failback (if applicable):

After primary site is restored, set up replication back to the original primary.

Verify data consistency.

Testing Recovery Procedures:

Planned Failover:

Stop the MySQL service on the primary server.

Promote the secondary server to be the new primary.

systemctl stop mysql

On the secondary server, run:

STOP SLAVE; CHANGE MASTER TO MASTER_HOST='<primary-ip>', MASTER_PORT=3306, MASTER_USER='replication_user', MASTER_PASSWORD='password'; START SLAVE;

Update application configuration to point to the new primary server.

Unplanned Failover:

Simulate a network failure or shut down the primary server.

On the secondary server, initiate the failover as described in the previous step.

Data Validation:

Compare data between primary and secondary servers to ensure they are consistent.

SELECT COUNT(*) FROM my_database.my_table; -- Compare counts of important tables

Failback (if applicable):

After primary site is restored, configure replication back to the original primary.

CHANGE MASTER TO MASTER_HOST='<original-primary-ip>', MASTER_PORT=3306, MASTER_USER='replication_user', MASTER_PASSWORD='password'; START SLAVE;

Implement replication of data and virtual machine images from on-premises to IBM Cloud Virtual Servers.

1. Establish Network Connectivity:

Option 1: VPN:

Set up a Virtual Private Network (VPN) between your on-premises data center and IBM Cloud VPC.

Configure routing and firewall rules to allow traffic between the two networks.

Option 2: IBM Direct Link:

Provision an IBM Direct Link connection between your on-premises location and IBM Cloud data center for high-speed, private network access.

2. Replication of Data:

a. Using Storage Replication:

Option 1: Block-Level Replication:

Set up block-level replication using technologies like IBM Spectrum Virtualize, SAN-based replication, or similar solutions.

Configure synchronous or asynchronous replication based on your RPO (Recovery Point Objective) requirements.

Option 2: File-Level Replication:

Use tools like rsync or commercial solutions for file-level replication.

b. Using Database Replication:

If you're dealing with databases, consider setting up database-level replication (e.g., MySQL replication, PostgreSQL streaming replication, etc.) for continuous synchronization.

3. Replication of Virtual Machine Images:

Create a process to regularly back up virtual machine images on-premises.

Upload these images to IBM Cloud Object Storage or another suitable storage solution in IBM Cloud.

4. Testing and Validation:

Regularly test the replication process to ensure it works as expected.

Validate that data and images are consistent between on-premises and IBM Cloud.

5. Failover and Failback:

Define procedures for failing over from on-premises to IBM Cloud in case of a disaster.

Establish procedures for failing back from IBM Cloud to on-premises if needed.

6. Security Considerations:

Ensure data encryption in transit and at rest.

Implement access controls and encryption keys for data security.

7. Monitoring and Alerting:

Set up monitoring for replication status, network health, and virtual machine availability.

Configure alerts to notify relevant teams in case of any issues.

Conduct recovery tests to ensure that the disaster recovery plan works as intended. Simulate a disaster scenario and practice recovery procedures.

1. Preparation:

Before conducting the recovery test, make sure to:

Notify Stakeholders: Inform relevant stakeholders about the test, including the date, time, and expected duration.

Document the Test Plan: Clearly outline the objectives, steps, and expected outcomes of the recovery test.

Backup Data: Ensure that you have recent backups of critical data and configurations.

Isolate the Test Environment: If possible, conduct the test in an isolated environment to avoid impacting production systems.

2. Simulate the Disaster Scenario:

Choose a disaster scenario to simulate. For example, you could simulate:

Hardware Failure: Simulate a critical server failure.

Data Corruption: Introduce data corruption or deletion.

Network Outage: Simulate a network failure.

3. Initiate the Recovery Process:

Follow the steps outlined in your disaster recovery plan to initiate the recovery process based on the simulated disaster scenario. This may include:

Failover Procedures: If you're switching from a primary to a secondary site, follow the failover procedures.

Data Restoration: Restore data from backups if necessary.

Configuration Updates: Update configurations as per the recovery plan.

4. Monitor and Validate:

During the recovery process, monitor the progress and validate that each step is executed correctly. Ensure that:

Data is restored accurately.

Services and applications are brought back online.

Network connectivity is re-established.

5. Functional Testing:

Perform functional tests to verify that all critical systems and applications are functioning as expected. This may involve:

Accessing key applications.

Performing transactions or operations.

6. Performance Testing:

If applicable, conduct performance testing to ensure that the recovered systems meet performance requirements.

7. Data Verification:

Verify the integrity and consistency of the recovered data. This can include comparing checksums or performing spot checks on critical data.

8. Document the Test Results:

Document the results of the recovery test, including any issues encountered, the time taken for recovery, and any deviations from the expected outcomes.

9. Post-Test Review:

After the test, gather feedback from the team members involved. Discuss any challenges faced, areas for improvement, and lessons learned.

10. Update the Disaster Recovery Plan:

Based on the findings from the test, update the disaster recovery plan as necessary. This could involve refining procedures, addressing gaps, or improving documentation.

11. Communicate the Results:

Share the results of the recovery test with relevant stakeholders. Highlight any improvements made to the disaster recovery plan.

12. Regularly Repeat Tests:

Schedule regular recovery tests to ensure that your disaster recovery plan remains effective and up-to-date.

Sample Code :

```
import requests
PRIMARY_SERVER = "http://primary-server"
SECONDARY_SERVER = "http://secondary-server"
def check_primary_health():
    try:
        response = requests.get(PRIMARY_SERVER)
        return response.status_code == 200
    except Exception as e:
        return False
def initiate_failover():
    try:
        response = requests.post(SECONDARY_SERVER + "/failover")
        return response.status_code == 200
    except Exception as e:
        return False
if not check_primary_health():
    if initiate_failover():
        print("Failover initiated successfully")
    else:
        print("Failover failed")
else:
    print("Primary server is healthy")
```