Disaster Recovery with IBM Cloud Virtual Servers

A disaster recovery strategy is a comprehensive plan that outlines how an organization will respond to and recover from disruptive events, such as natural disasters, system failures, or cyberattacks, to ensure the continued availability of critical systems and data. It involves various elements, including Recovery Time Objective (RTO), Recovery Point Objective (RPO), and the priority of virtual machines. Here's a breakdown of these components and steps to set up regular backups:

1. Define Disaster Recovery Strategy:

Recovery Time Objective (RTO): RTO is the maximum acceptable downtime for your critical systems and applications. It specifies the time it should take to recover these systems after a disaster. For example, if your RTO for a critical system is four hours, you should be able to restore it within that time frame.

Recovery Point Objective (RPO): RPO defines the maximum amount of data loss your organization can tolerate. It determines how frequently backups should be taken. For instance, if your RPO is one hour, you can afford to lose only up to one hour of data in the event of a disaster.

Priority of Virtual Machines: Categorize your virtual machines (VMs) based on their criticality to your business operations. For example:

Priority 1: VMs critical for immediate business continuity.

Priority 2: VMs important for regular operations but not as critical as Priority 1.

Priority 3: VMs used for non-essential functions.

Test/Development VMs: VMs used for testing and development purposes.

2. Set Up Regular Backups:

Identify Backup Tools or Scripts: Choose backup solutions or scripts that are suitable for your virtualization platform (e.g., VMware, Hyper-V, or cloud-based solutions). Ensure they are compatible with your VMs and provide the necessary features for backup and recovery.

Configure Backup Schedules: Define backup schedules for your VMs based on their RPO and priority levels. Priority 1 VMs should have more frequent backups, while less critical VMs can have less frequent backups.

Data Backup and Storage: Back up the VMs and associated data to a secure location. Consider using a combination of on-site and off-site backups to mitigate data loss due to disasters.

Test Backups: Regularly test the backup process to ensure the integrity of your backup data. This can involve performing restoration tests to verify that you can recover VMs and data as needed.

Monitor and Automate: Implement monitoring and alerting systems to notify you of any backup failures or issues. Whenever possible, use automation to schedule and execute backup processes, reducing the risk of human error.

Versioning: Some backup solutions provide versioning, allowing you to restore to a specific point in time, which can be helpful in case of data corruption or other issues.

Encryption: Ensure that your backup data is encrypted to protect it from unauthorized access.

Disaster Recovery Plan Documentation: Document your disaster recovery plan, including backup procedures, roles and responsibilities, and contact information for team members.

Compliance and Regulations: If your organization has specific regulatory requirements, ensure that your backup and recovery processes are in compliance.

Disaster Recovery Testing: Periodically test your entire disaster recovery plan, including the backup and recovery procedures, to identify and address any potential issues.

It's crucial to tailor your disaster recovery strategy and backup practices to your organization's specific needs and risks. Regular reviews and updates are necessary to adapt to changes in technology, business requirements, and potential threats. By following these steps, you can establish a robust and effective disaster recovery strategy that ensures the continuity of your critical systems and data.

Real time building process:

Building a disaster recovery application involves creating a software solution that can automate the recovery process, including tasks like deploying virtual machines, restoring data, and ensuring minimal downtime. To build such an application, you can follow these general steps:

Define Disaster Recovery Requirements:

Determine the Recovery Time Objective (RTO) and Recovery Point Objective (RPO) for your virtual machines. This helps you set recovery goals and plan your application accordingly.

Identify the priority of virtual machines and data, categorizing them into critical, important, and non-essential categories.

Select a Development Platform:

Choose a development platform or technology stack suitable for your application. Common choices include Python, Java, .NET, or cloud-native technologies like AWS Lambda, Azure Functions, or Google Cloud Functions.

Design the Application:

Plan the architecture of your disaster recovery application. Consider the following elements:

User interface: Determine how users will interact with the application.

Backend: Design the core logic for backup, recovery, and monitoring.

Integrations: Identify third-party tools, services, or APIs for data recovery, virtual machine management, and alerting.

Data storage: Decide how and where you will store application data and configurations.

Develop the Application:

Write the code for your application based on the design. Implement features for backup, recovery, and management of virtual machines. You may need to use relevant SDKs or libraries to interact with virtualization platforms, cloud providers, and backup solutions.

Testing and Validation:

Test your application thoroughly to ensure it meets your RTO and RPO requirements. Conduct various test scenarios, including disaster recovery drills, to validate the application's effectiveness.

Implement Automation:

Automate recovery tasks to minimize manual intervention. This may involve scripting or using infrastructure as code (IaC) tools to automate the provisioning and configuration of virtual machines.

Monitoring and Alerts:

Implement monitoring and alerting within your application to provide real-time status updates. Monitor the health of the virtual machines and data, and set up alerts for any issues that may arise.

Documentation:

Create comprehensive documentation for your disaster recovery application, including user guides, recovery procedures, and maintenance instructions.

Security and Compliance:

Ensure that your application adheres to security best practices, including access controls, encryption, and compliance with relevant regulations (e.g., GDPR, HIPAA).

Deployment:

Deploy your application in a production environment. This may involve deploying to on-premises infrastructure or utilizing cloud resources, depending on your disaster recovery architecture.

User Training:

Train your team and relevant stakeholders on how to use the disaster recovery application effectively. This includes following recovery procedures and managing virtual machines.

Continual Improvement:

Disaster recovery is an ongoing process. Regularly review and update your application to account for changes in infrastructure, technology, and business needs. Conduct periodic disaster recovery tests to ensure the application's effectiveness.

Post-Disaster Analysis:

After a disaster event, conduct a post-mortem analysis to assess the performance of your disaster recovery application. Identify any areas for improvement and make necessary adjustments.

Compliance and Reporting:

If your organization requires compliance reporting, ensure that the application provides the necessary data and logs for compliance purposes.

Building a disaster recovery application is a complex task that requires careful planning, development, and ongoing maintenance. It is crucial to align the application with your organization's recovery objectives and maintain it to adapt to changing requirements and technologies.

Python script to set up regular backups of an on-premises virtual machine using the rsync command. You can schedule this script to run at specified intervals using a task scheduler or cron job on Linux

Sample script:

import subprocess

import datetime

import shutil

import os

# Source directory to be backed up

SOURCE\_DIR = "/path/to/source/directory"

# Destination directory for backups

BACKUP\_DIR = "/path/to/backup/directory"

# Backup retention policy (e.g., keep the last 7 backups)

RETENTION = 7

# Create a timestamp for the backup folder name

timestamp = datetime.datetime.now().strftime("%Y%m%d-%H%M")

backup\_dest = os.path.join(BACKUP\_DIR, timestamp)

# Create the backup directory

os.makedirs(backup\_dest)

# Perform the backup using rsync

backup\_command = f"rsync -avz {SOURCE\_DIR}/ {backup\_dest}/"

subprocess.run(backup\_command, shell=True)

# Clean up old backups based on the retention policy

all\_backups = sorted(os.listdir(BACKUP\_DIR))

if len(all\_backups) > RETENTION:

backups\_to\_remove = all\_backups[:len(all\_backups) - RETENTION]

for old\_backup in backups\_to\_remove:

shutil.rmtree(os.path.join(BACKUP\_DIR, old\_backup))