 **GRT INSTITUTE OF**

**ENGINEERING AND**

**TECHNOLOGY, TIRUTTANI - 631209**

**Approved by AICTE, New Delhi Affiliated to Anna University, Chennai**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**PROJECT TITLE**

***Disaster Recovery with IBM Cloud Virtual Servers***

**COLLEGE CODE:1103**

**Phase-3**

**Priyadharshini.S**

3rd year, 5th sem

Reg no.:110321104037

[**priya2004madhi@gmail.com**](mailto:priya2004madhi@gmail.com)

**DISASTER RECOVERY WITH IBM CLOUD VIRTUAL SERVERS**

**BUILDING THE DISASTER RECOVERY PLAN :**

Creating a comprehensive source code for a disaster recovery (DR) solution involving primary site, production site, and DR site with IBM Cloud virtual servers is a complex task. In such a solution, you'd typically use various services, APIs, and infrastructure orchestration tools to manage and switch between sites.

I can provide a simplified Python script that simulates the concept of switching between these sites and generates text-based output.

**SOURCE CODE:**

**class DisasterRecoveryManager:**

**def \_\_init\_\_(self):**

**self.primary\_site = "Primary Site"**

**self.production\_site = "Production Site"**

**self.dr\_site = "DR Site"**

**self.current\_site = self.primary\_site**

**def switch\_to\_site(self, site):**

**print(f"Switching to {site}...")**

**self.current\_site = site**

**def run\_production(self):**

**if self.current\_site == self.production\_site:**

**print("Already in the Production Site.")**

**else:**

**self.switch\_to\_site(self.production\_site)**

**print("Running in the Production Site.")**

**def run\_dr(self):**

**if self.current\_site == self.dr\_site:**

**print("Already in the DR Site.")**

**else:**

**self.switch\_to\_site(self.dr\_site)**

**print("Running in the DR Site.")**

**def run\_primary(self):**

**if self.current\_site == self.primary\_site:**

**print("Already in the Primary Site.")**

**else:**

**self.switch\_to\_site(self.primary\_site)**

**print("Running in the Primary Site.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**drm = DisasterRecoveryManager()**

**while True:**

**print("\nChoose an option:")**

**print("1. Switch to Production Site")**

**print("2. Switch to DR Site")**

**print("3. Switch to Primary Site")**

**print("4. Exit")**

**choice = input()**

**if choice == "1":**

**drm.run\_production()**

**elif choice == "2":**

**drm.run\_dr()**

**elif choice == "3":**

**drm.run\_primary()**

**elif choice == "4":**

**print("Exiting the disaster recovery manager.")**

**break**

**else:**

**print("Invalid choice. Please choose a valid option.")**

In this simplified Python script, you can simulate switching between primary, production, and DR sites through text-based input.

**DISASTER RECOVERY STRATERGY:**

A disaster recovery (DR) strategy is a comprehensive plan that an organization develops to ensure the continuity of its operations and data in the event of a disaster or major disruption.

Every situation is unique and there is no single correct way to develop a disaster recovery plan. However, there are three principal goals of disaster recovery that form the core of most DRPs.

1. prevention, including proper backups, generators, and surge protectors
2. detection of new potential threats, a natural byproduct of routine inspections
3. correction, which might include holding a “lessons learned” brainstorming session and securing proper insurance policies

Key components of a DR strategy include:

* Recovery Time Objective (RTO)
* Recovery Point Objective (RPO)
* The prioritization of virtual machines

**RECOVERY TIME OBJECTIVE:**

The Recovery Time Objective (RTO) is a crucial metric in disaster recovery planning. It specifies the maximum allowable downtime for a system, application, or business process following a disruption, such as a natural disaster, hardware failure, or cyberattack. RTO is defined as the target time within which a system or process should be restored and made operational after an incident to avoid significant negative impacts on the organization's operations.

**Examples of RTOs:**

RTOs can vary widely based on the nature of the business and its reliance on different systems.

* An e-commerce website might have an RTO of a few minutes to maintain real-time availability.
* A financial institution may have an RTO of a few hours to ensure that core banking systems are operational.
* A non-essential internal application might have a longer RTO, possibly measured in days.

**RELATION TO RPO:**

RTO is closely related to the Recovery Point Objective (RPO). While RTO focuses on the time it takes to restore a system or process, RPO specifies the acceptable data loss in the event of a disruption. Both RTO and RPO are key parameters in designing a DR strategy.

**RECOVERY POINT OBJECTIVE (RPO):**

The Recovery Point Objective (RPO) is a critical parameter in disaster recovery planning that defines the maximum allowable data loss, measured in time, following a disruption. RPO specifies the point in time to which data must be recoverable after a disaster, ensuring that data loss remains within acceptable limits.

RPO is a time-based metric that quantifies the acceptable age of the data to be recovered after a disruptive event. It represents the maximum amount of data that an organization is willing to lose in the event of a disaster.

**Examples of RPOs:**

RPOs can vary widely based on the nature of the business and its dependence on different data-dependent processes. Examples include:

* An e-commerce platform may have an RPO of a few seconds to ensure minimal data loss in transactions.
* A financial institution may target an RPO of a few minutes to maintain accurate account data.
* An internal email system might have a longer RPO, such as a few hours, where a limited amount of data loss is acceptable.

**RELATION TO RTO:**

While RPO defines the acceptable data loss, the Recovery Time Objective (RTO) specifies the maximum tolerable downtime for a system. Both RPO and RTO are critical parameters in disaster recovery planning, and they are interconnected. The choice of data replication, backup frequency, and recovery technology affects both RPO and RTO.

**THE PRIORITIZATION OF VIRTUAL MACHINES DETAILS:**

The prioritization of virtual machines (VMs) in a disaster recovery (DR) strategy is an essential aspect of ensuring business continuity. VM prioritization determines the order in which VMs are recovered or brought back into operation following a disaster. The priority assigned to each VM is based on its criticality to business operations.

VM prioritization, often referred to as "VM prioritization order" or "failover order," specifies which VMs should be recovered first and which should follow in the event of a disaster. It ranks VMs based on their importance to the organization's core functions.

**Examples of VM Prioritization:**

* Critical VMs: Database servers, primary web servers, and customer relationship management (CRM) systems.
* High-Priority VMs: Secondary application servers, email servers, and document management systems.
* Medium-Priority VMs: Testing and development servers, internal communication tools, and non-essential applications.
* Low-Priority VMs: Archive servers, old data repositories, and non-essential backup servers.

VM prioritization is a critical component of a successful disaster recovery strategy. It ensures that in the event of a disaster, limited resources are allocated to the most critical systems first, helping to minimize downtime and maintain business operations as smoothly as possible.

**SETTING UP OF BACKUPS:**

Creating a complete disaster recovery program with output for IBM Cloud Virtual Servers is a complex and customized task, and providing a fully functional program with complete output is beyond the scope of a simple response. However, I can provide you with a simplified example that demonstrates the basic concepts of a disaster recovery plan using Python.

**SOURCE CODE:**

**import random**

**import time**

**# Simulate the primary and secondary servers**

**primary\_server\_status = "running"**

**secondary\_server\_status = "stopped"**

**def create\_backup\_server():**

**global secondary\_server\_status**

**print("Creating a backup virtual server...")**

**# Simulate the creation of a backup server**

**time.sleep(5)**

**secondary\_server\_status = "running"**

**print("Backup virtual server created and running.")**

**def simulate\_failover():**

**global primary\_server\_status, secondary\_server\_status**

**print("Simulating a failover...")**

**# Simulate primary server failure**

**primary\_server\_status = "stopped"**

**# Simulate secondary server taking over**

**secondary\_server\_status = "running"**

**print("Failover complete.")**

**def main():**

**print("Disaster Recovery Simulation")**

**print("Primary Virtual Server Status:", primary\_server\_status)**

**print("Secondary Virtual Server Status:", secondary\_server\_status)**

**while True:**

**print("\nOptions:")**

**print("1. Simulate Primary Server Failure")**

**print("2. Create Backup Server")**

**print("3. Exit")**

**choice = input("Enter your choice: ")**

**if choice == "1":**

**if primary\_server\_status == "running":**

**simulate\_failover()**

**else:**

**print("Primary server is already stopped.")**

**elif choice == "2":**

**if secondary\_server\_status == "stopped":**

**create\_backup\_server()**

**else:**

**print("Backup server is already running.")**

**elif choice == "3":**

**print("Exiting the program.")**

**break**

**else:**

**print("Invalid choice. Please try again.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

Here's a basic Python script:

This script simulates a disaster recovery scenario with two virtual servers: a primary and a secondary (backup) server. The primary server can fail, and the secondary server can take over.

Here's what you can expect when running this script:

1.You will be presented with options to simulate a primary server failure, create a backup server, or exit the program.

2.When you choose to simulate a primary server failure, it will change the status of the primary server to "stopped" and the status of the secondary server to "running," simulating a failover.  
 3.When you choose to create a backup server, it will change the status of the secondary server to "running," simulating the creation of a backup server.

4.You can repeat these actions to simulate different disaster recovery scenarios.

In a real-world scenario, would need to use IBM Cloud APIs and SDKs to automate the creation and management of virtual servers, networking, and data replication to achieve a true disaster recovery solution.