DISASTER RECOVERY WITH IBM CLOUD VIRTUAL SERVERS

Automated recovery scripts

Automated recovery scripts can be used to automate the process of restoring systems and data after a disaster. This can help to reduce the amount of time required to recover and get back to business.

Automated recovery scripts can be made more complex to meet the specific needs of an organization. For example, a script could be written to automatically switch to a secondary data center in the event of a primary data center outage.

Automated recovery scripts are triggered when a disaster is detected. The scripts restore systems and data from backups and bring them back online.

By automating the recovery process, organizations can significantly reduce the amount of time it takes to recover from a disaster.

Here is an example of how the system might be used:

- **A fire broke out in the primary data center, causing the servers to shut down.**
- **The proactive monitoring system detects the outage and triggers the automated recovery scripts.**
- **The automated recovery scripts back up the data from the primary data center to the secondary data center.**
- **The automated recovery scripts then fail over the applications from the primary data center to the secondary data center.**
- **Once the fire has been extinguished and the primary data center has been restored, the automated recovery scripts can restore the data from the secondary data center to the primary data center.**

By automating the recovery process, the organization was able to minimize the downtime caused by the fire and get their applications back up and running quickly.

The automated recovery scripts can perform a variety of tasks, such as:

- Backing up the data from the primary data center to the secondary data center
- Failing over the applications from the primary data center to the secondary data center
- Restoring the data from the secondary data center to the primary data center once the disaster has been resolved

Here are some additional benefits of using automated recovery scripts during disasters:

- **Reduced human error**: Automated recovery scripts eliminate the possibility of human error, which can be a major cause of problems during disaster recovery.
- **Increased speed**: Automated recovery scripts can perform tasks much faster than humans can, which can significantly reduce the amount of time it takes to recover from a disaster.
- **Improved accuracy**: Automated recovery scripts are always accurate and consistent, which can help to ensure that the recovery process is successful.
- **Reduced costs**: Automated recovery scripts can help to reduce the costs associated with disaster recovery by eliminating the need for manual intervention.

Here is an example of a simple automated recovery script in Python:

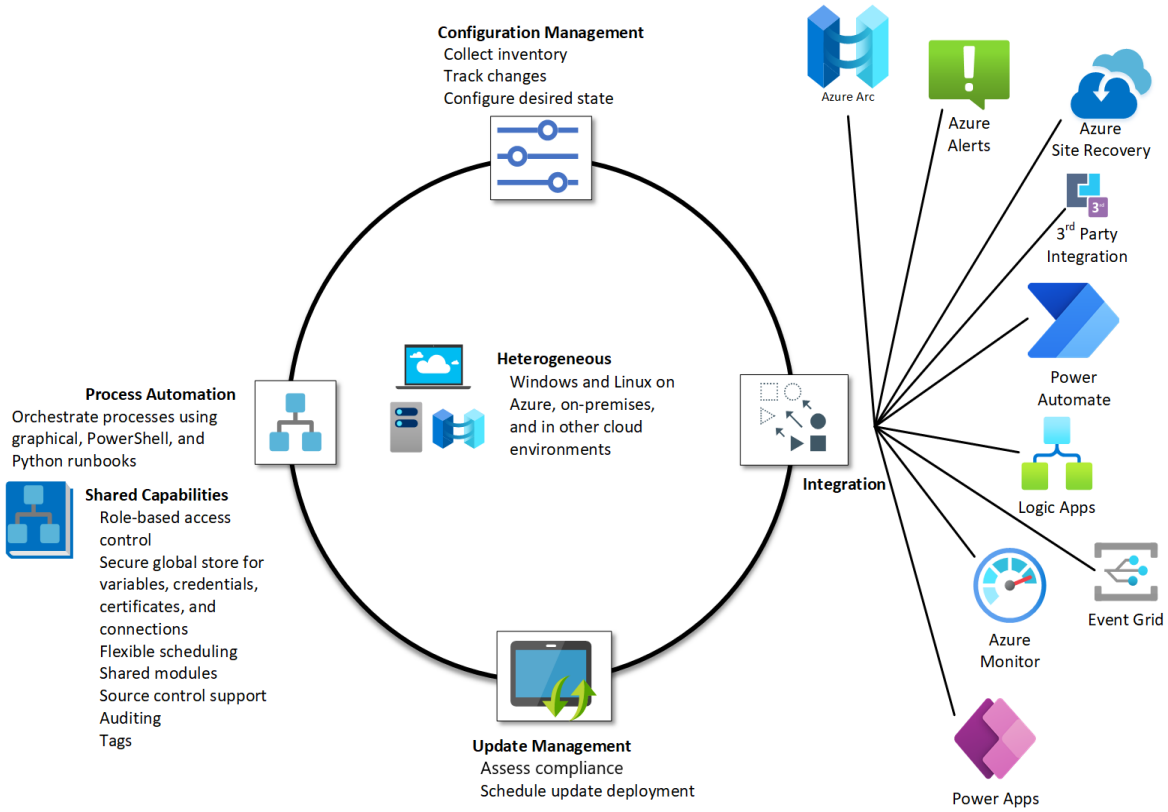
```
import sys

def get_system_state():
  # Get the current state of the system here.
  # This could involve checking the status of various services,
  # or reading the contents of log files.
  pass

def check_for_errors():
  # Check for any errors in the system.
  # This could involve looking for specific errors in the log files,
  # or checking the status of specific services.
  pass

def recover_from_errors():
  # Attempt to recover from any errors that are found.
  # This could involve restarting services or restoring data from
backups.
  pass
```

```
def notify_user():
  # Notify the user of any errors that could not be recovered from.
  # This could involve sending an email or SMS message,
  # or logging an error message to a file.
  pass

if __name__ == '__main__':
  # Get the system state.
  system_state = get_system_state()

  # Check for errors.
  errors = check_for_errors()

  # If errors are found, attempt to recover from them.
  if errors:
    try:
      recover_from_errors()
    except Exception as e:
      # Recovery failed. Notify the user.
      notify_user(e)
  else:
    # No errors found. Continue.
    pass
```

Here are some tips for developing effective automated recovery scripts:

- Make sure that the scripts are well-documented and easy to understand.
- Use modular design principles to make the scripts easy to maintain and update.
- Test the scripts thoroughly to ensure that they are working as expected.
- Monitor the scripts to identify any potential problems.

Diagram of a simple automated recovery script:



- **Detect disaster**: The first step is to detect that a disaster has occurred. This can be done using various methods, such as monitoring system logs or environmental sensors.
- **Trigger recovery script**: Once a disaster has been detected, the recovery script is triggered. The script will typically perform the following steps:
- Shut down systems safely.
- Back up any critical data.
- Restore systems from backups.
- Start-up systems and bring them back online.
- **Monitor recovery:** The recovery process should be monitored to ensure that it is successful. If any problems occur, the recovery team can intervene manually.

Proactive monitoring

Proactive monitoring can be used to identify and address potential problems before they cause a disaster. This can help to prevent disasters from happening in the first place, and it can also help to reduce the impact of disasters that do occur.

Combining automated recovery scripts and proactive monitoring:

Automated recovery scripts and proactive monitoring can be combined to provide a more comprehensive disaster recovery solution. For example, proactive monitoring can be used to identify and address potential problems before they cause a disaster. If a disaster does occur, automated recovery scripts can be used to quickly restore systems and data.

1. **Proactive monitoring**: The proactive monitoring system collects metrics from systems and applications and analyzes them to identify potential problems. If a potential problem is identified, an alert is generated.
2. **Automated recovery scripts**: The automated recovery scripts are triggered when a disaster is detected. The scripts restore systems and data from backups and bring them back online.
3. **Human intervention**: The disaster recovery team monitors the recovery process and intervenes manually if necessary.

Here are some tips for implementing effective proactive monitoring:

- Identify the systems and applications that you want to monitor.
- Define the metrics, logs, and other data that you want to collect.
- Set thresholds for the metrics and logs that you are monitoring.
- Implement alerts that will notify you when the thresholds are exceeded.
- Review the alerts and address the potential problems.
- Regularly test the monitoring system to ensure that it is working as expected.

Here is a simple example of a proactive monitoring script in Python:

```
import requests
import time

def get_cpu_usage():
```

```
  """Returns the CPU usage as a percentage."""

  response = requests.get("http://localhost:9090/metrics")
  metrics = response.json()

  cpu_usage = metrics["process_cpu_seconds_total"] /
time.process_time() * 100

  return cpu_usage

def send_alert(cpu_usage):
  """Sends an alert if the CPU usage exceeds a threshold."""

  if cpu_usage > 90:
    print ("Alert! CPU usage is too high: {} %".format(cpu_usage))

if __name__ == "_main__":
  while True:
    cpu_usage = get_cpu_usage()
    send_alert(cpu_usage)

    time. Sleep (1)
```

Here is an example of how the script could be used in a production environment:

1.Deploy the script to a server that has access to the systems and applications that you want to monitor.
2.Configure the script to run continuously.
3.Configure the script to send alerts to the appropriate people or teams.

Diagram of a Proactive monitoring:

1.**Collect metrics**: The first step is to collect metrics from systems and applications. This data can be used to identify potential problems, such as high CPU usage or low disk space.

2.**Analyze metrics**: The collected metrics are analyzed to identify potential problems. This analysis can be performed manually or using automated tools.

3.**Alert and respond**: If a potential problem is identified, an alert is generated. The alert can be sent to a human administrator or to an automated system. The appropriate response to the alert will depend on the problem.