PCET's
Pimpri
Chinchwad
University, Pune

Learn | Grow | Achieve

Checked By	Dr. Avinash Golande
Sign	
Date	
Grade	

EXPERIMENT NO . 07

- **Title:-** Implement a Python-based failover mechanism simulation where service requests are switched between primary and backup instances.

- **Objective :-**

  - To understand the concept of **failover** and **high availability** in cloud environments.
  - To simulate a **Python-based failover system** where requests automatically switch between primary and backup services.
  - To understand **health-check logic**, **failback**, and **system resilience** mechanisms used in distributed systems.

Resources used :- PC / Laptop with Python 3.x installedPython libraries: random, timeCode Editor (VS Code / PyCharm / IDLE)Operating System: Windows / Linux (Ubuntu recommended)

- **Theory :-**

In **distributed and cloud-based systems**, ensuring continuous service availability is one of the most important design goals. Businesses and cloud service providers cannot afford downtime because it directly impacts user experience and system reliability. Therefore, cloud infrastructure is built with the concept of **redundancy**, where multiple instances of a service exist — one active (primary) and another standby (backup) — to maintain smooth operations even during failures.

A **failover mechanism** is a fault-tolerance strategy used in distributed systems, where workloads are automatically transferred from a failed primary instance to a standby backup instance. When the **primary instance** stops responding (due to crash, maintenance, or network issues), the **failover process** ensures that the **backup instance** immediately takes over, minimizing downtime and preventing service interruption.

Key Concepts Involved:

- **Primary Instance:** The main active server or service responsible for handling client requests during normal operation.
- **Backup Instance:** A standby or secondary service that remains idle but ready to take over when the primary fails.
- **Failover:** The process of automatic switching from the primary instance to the backup instance when a failure is detected.
- **Failback:** Once the primary instance recovers, traffic is switched back from the backup to the primary system.
- **Health Check:** A monitoring mechanism that continuously checks the health status of the primary instance (via heartbeat signals, ping, or status checks).

Failover mechanisms are essential to achieve **High Availability (HA)** and **Fault Tolerance**, two critical aspects of **Cloud Reliability Engineering**. In large-scale cloud platforms such as **Amazon Web Services (AWS)**, **Microsoft Azure**, and **Google Cloud Platform (GCP)**, automatic failover systems are part of managed services like **AWS EC2 Auto Recovery**, **Azure Load Balancer**, and **Google Cloud Failover**. These ensure that if one virtual machine or node goes down, another takes its place almost instantly.

In this experiment, we simulated a **Python-based failover system** that mimics how real-world cloud environments handle service interruptions. The system performs continuous health checks on the primary service. When the primary instance fails, it automatically redirects all incoming requests to the backup instance. Once the primary instance becomes available again, the system performs a **failback**, switching requests back to the primary.

This experiment helps us understand how **resilience**, **monitoring**, and **automatic recovery** work in cloud systems. By simulating this process using simple Python logic, students gain practical exposure to key cloud concepts like **redundancy**, **load balancing**, **monitoring**, and **disaster recovery**. The experiment also highlights the importance of automation in modern DevOps and cloud infrastructure, ensuring that services remain uninterrupted even in failure conditions.

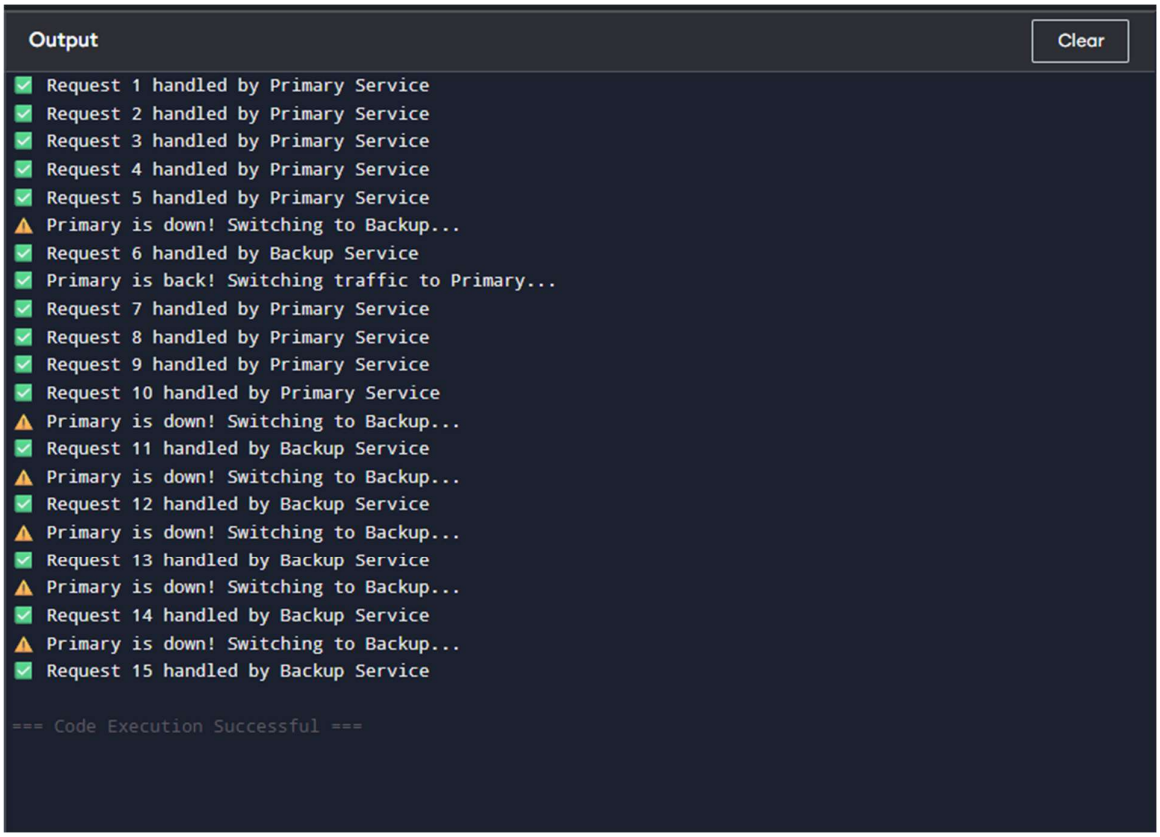
- ## Code

```
import random

import time
```

```
# ---------------- Service Instance ----------------
class ServiceInstance:
    def __init__(self, name):
        self.name = name
        self.is_active = True  # Initially active

    def process_request(self, request_id):
        if self.is_active:
            print(f"✅ Request {request_id} handled by {self.name}")
        else:
            raise Exception(f"❌ {self.name} is down!")

# ---------------- Failover System ----------------
class FailoverSystem:
    def __init__(self, primary, backup):
        self.primary = primary
        self.backup = backup
        self.active_instance = primary

    def health_check(self):
        """Check if active instance is alive, switch if not"""
        if not self.primary.is_active:
            print("⚠️ Primary is down! Switching to Backup...")
            self.active_instance = self.backup
        else:
            if self.active_instance != self.primary:
                print("✅ Primary is back! Switching traffic to Primary...")
                self.active_instance = self.primary

    def handle_request(self, request_id):
```

```
        self.health_check()
        try:
            self.active_instance.process_request(request_id)
        except Exception as e:
            print(e)
            print("➡️ Redirecting request to backup...")
            self.backup.process_request(request_id)

# ---------------- Simulation ----------------
if __name__ == "__main__":
    primary_service = ServiceInstance("Primary Service")
    backup_service = ServiceInstance("Backup Service")

    system = FailoverSystem(primary_service, backup_service)

    # Simulate 15 requests with random failures
    for req in range(1, 16):
        # Randomly simulate primary going down or recovering
        if random.random() < 0.2:   # 20% chance primary goes down
            primary_service.is_active = False
        elif random.random() < 0.2: # 20% chance primary recovers
            primary_service.is_active = True

        system.handle_request(req)
        time.sleep(0.5)
```

- **Output**



- **Conclusion:**

In this practical, we successfully developed and executed a **Python-based failover mechanism simulation**. The system automatically switched requests between a **primary** and **backup** instance based on their health status. Through this, we learned how **failover**, **failback**, and **health-check mechanisms** maintain **high availability** and **resilience** in cloud infrastructures.

This experiment provided hands-on experience in implementing reliability concepts that are fundamental to real-world **cloud computing environments** such as AWS, Azure, and Google Cloud.