Scenario 1: Automated Health Check and Reporting of Multiple Web Services

This script will read a list of web service URLs from a CSV file, perform a health check (by sending an HTTP GET request), and then log the status and response time to another CSV file.

```
import requests
import csv
from datetime import datetime
import time
import os

def check_web_service(url):
    """Performs a health check on a given URL."""
    try:
        start_time = time.time()
        response = requests.get(url, timeout=5)
        end_time = time.time()
        response_time = end_time - start_time
        status_code = response.status_code
        if status_code == 200:
            return "UP", status_code, f"{response_time:.2f}"
        else:
            return "DOWN", status_code, f"{response_time:.2f}"
    except requests.exceptions.RequestException as e:
        return "ERROR", str(e), ""

def main():
    """Reads URLs from a CSV, checks their status, and writes results to another CSV."""
    input_csv_file = "web_services.csv"
    output_csv_file = "health_check_report.csv"

    if not os.path.exists(input_csv_file):
        print(f"Error: Input CSV file '{input_csv_file}' not found.")
        return

    web_services = []
    try:
        with open(input_csv_file, 'r', newline='') as csvfile:
            reader = csv.reader(csvfile)
            next(reader, None)  # Skip header row if it exists
            for row in reader:
                if row:
                    web_services.append(row[0].strip())
    except Exception as e:
        print(f"Error reading CSV file: {e}")
        return

    report_data = [["Timestamp", "URL", "Status", "Status Code", "Response Time (s)"]]
    for url in web_services:
        status, code, response_time = check_web_service(url)
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        report_data.append([timestamp, url, status, code, response_time])
        print(f"Checked {url}: Status - {status}, Code - {code}, Response Time - {response_time}s")
        time.sleep(1) # Be polite to the servers

    try:
        with open(output_csv_file, 'w', newline='') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerows(report_data)
        print(f"\nHealth check report written to '{output_csv_file}'")
    except Exception as e:
        print(f"Error writing to CSV file: {e}")

if __name__ == "__main__":
    main()
```

To use this:

Create a CSV file named `web_services.csv` in the same directory as the script. This file should contain a list of URLs in the first column (you can have a header row which will be skipped). For example:

Code snippet

```
URL
https://www.google.com
https://httpbin.org/delay/2
https://nonexistent-website.xyz
```

Run the Python script. It will read the URLs, check their status, print the results to the console, and save a detailed report in `health_check_report.csv`.

Scenario 2: Automated Deployment Configuration from YAML and API Update

This script reads deployment configurations from a YAML file, extracts specific parameters, and then uses an API (simulated here with `requests.post`) to update the deployment settings.

```
import yaml

import requests

import json

import os

from datetime import datetime


def load_config(config_file):

    """Loads configuration from a YAML file."""

    try:

        with open(config_file, 'r') as f:

            return yaml.safe_load(f)

    except FileNotFoundError:

        print(f"Error: Configuration file '{config_file}' not found.")

        return None

    except yaml.YAMLError as e:

        print(f"Error parsing YAML file: {e}")

        return None


def update_deployment_api(api_url, deployment_data):

    """Simulates updating a deployment API."""

    try:
        headers = {'Content-Type': 'application/json'}
        response = requests.post(api_url, headers=headers, data=json.dumps(deployment_data), timeout=10)
        response.raise_for_status()  # Raise an exception for bad status codes
        return response.json()
    except requests.exceptions.RequestException as e:
        print(f"Error communicating with API: {e}")
        return None


def main():
    """Loads deployment config from YAML and updates a (simulated) API."""
    config_file = "deployment_config.yaml"
    api_endpoint = "https://your-deployment-api.com/update"

    config = load_config(config_file)
    if not config:
        return

    deployment_name = config.get('deployment_name')
    version = config.get('version')
    replicas = config.get('replicas')
    environment = config.get('environment')
    deploy_timestamp = datetime.now().isoformat()

    if deployment_name and version and environment is not None:
        deployment_info = {

            "name": deployment_name,

            "version": version,

            "replicas": replicas,

            "environment": environment,

            "deployed_at": deploy_timestamp

        }

        print("Deployment information extracted:", deployment_info)



        api_response = update_deployment_api(api_endpoint, deployment_info)

        if api_response:

            print("API update successful. Response:", json.dumps(api_response, indent=4))

        else:

            print("API update failed.")

    else:

        print("Error: Missing required fields in the configuration file.")



if __name__ == "__main__":

    main()
```

To use this:

Create a YAML file named `deployment_config.yaml` in the same directory as the script. This file should contain deployment parameters. For example:

YAML

```
deployment_name: my-app

version: 1.2.3
```

```
replicas: 3

environment: production

database_url: "internal:db"
```

Run the Python script. It will load the configuration, extract relevant details, and simulate sending this data to a deployment API. You would replace `"https://your-deployment-api.com/update"` with the actual API endpoint.

Scenario 3: Log File Analysis and Alerting based on Time Windows

This script monitors a log file for specific error patterns within a defined time window. If the error count exceeds a threshold, it prints an alert.

```
import os

import time

from datetime import datetime, timedelta

import re


def analyze_logs(log_file, pattern, time_window_seconds, error_threshold):

    """Analyzes a log file for a pattern within a time window."""

    if not os.path.exists(log_file):

        print(f"Error: Log file '{log_file}' not found.")

        return
    current_time = datetime.now()
    start_time = current_time - timedelta(seconds=time_window_seconds)
    error_count = 0

    try:
        with open(log_file, 'r') as f:
            for line in f:
                timestamp_match = re.search(r'(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})', line) # Adjust regex for your timestamp format
                if timestamp_match:
                    log_timestamp_str = timestamp_match.group(1)
                    try:
                        log_timestamp = datetime.strptime(log_timestamp_str, '%Y-%m-%d %H:%M:%S') # Adjust format if needed
                        if start_time <= log_timestamp <= current_time:
                            if re.search(pattern, line):
                                error_count += 1
                    except ValueError:
                        print(f"Warning: Could not parse timestamp in line: {line.strip()}")
                elif re.search(pattern, line):
                    # If no timestamp, consider it within the window (simplification)
                    error_count += 1

        if error_count > error_threshold:
            print(f"Alert: Found {error_count} occurrences of '{pattern}' in '{log_file}' within the last
{time_window_seconds} seconds. Threshold exceeded ({error_threshold}).")
        else:
            print(f"No excessive occurrences of '{pattern}' found in '{log_file}' within the last
{time_window_seconds} seconds.")

    except Exception as e:
        print(f"Error reading log file: {e}")

def main():
    """Sets parameters for log analysis."""
    log_file_to_monitor = "application.log"
    error_regex_pattern = r"ERROR|Exception"
    monitoring_window = 60  # seconds
    max_errors = 5

    # Create a dummy log file for testing
    with open(log_file_to_monitor, 'w') as f:
        f.write(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')} - INFO - Application
started\n")
        for i in range(7):
            timestamp = (datetime.now() - timedelta(seconds=10 + i * 5)).strftime('%Y-%m-%d
%H:%M:%S')
            f.write(f"{timestamp} - ERROR - Something went wrong\n")
        f.write(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')} - INFO - Application
finished\n")
```

```
    analyze_logs(log_file_to_monitor, error_regex_pattern, monitoring_window, max_errors)


if __name__ == "__main__":
    main()
```

To use this:

1. ***Create a dummy log file named `application.log` (or use an existing one). The script includes a part to create a sample log file for testing.***
2. ***Run the Python script. It will analyze the log file for lines containing "ERROR" or "Exception" within the last 60 seconds. If the count exceeds 5, it will print an alert. You can adjust the `log_file_to_monitor`, `error_regex_pattern`, `monitoring_window`, and `max_errors` variables.***

Scenario 4: Automating Backups with Timestamps

This script takes a source directory, creates a timestamped backup of it in a destination directory, and can optionally clean up old backups based on a retention policy.

```
import os
import shutil
from datetime import datetime
import time

def create_backup(source_dir, backup_dir):
    """Creates a timestamped backup of the source directory."""
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    backup_path = os.path.join(backup_dir, f"backup_{timestamp}")

    try:
        shutil.copytree(source_dir, backup_path)
        print(f"Backup created successfully at: {backup_path}")
        return backup_path
    except FileNotFoundError:
        print(f"Error: Source directory '{source_dir}' not found.")
        return None
    except OSError as e:
        print(f"Error creating backup: {e}")
        return None

def cleanup_old_backups(backup_dir, retention_days):
    """Deletes backups older than the specified retention period."""
    if not os.path.exists(backup_dir):
        print(f"Warning: Backup directory '{backup_dir}' not found for cleanup.")
        return

    cutoff_time = datetime.now() - timedelta(days=retention_days)
    deleted_count = 0

    for item_name in os.listdir(backup_dir):
        item_path = os.path.join(backup_dir, item_name)
        try:
            if os.path.isdir(item_path) and item_name.startswith("backup_"):
                timestamp_str = item_name.split("_")[1]
                backup_time = datetime.strptime(timestamp_str, "%Y%m%d_%H%M%S")
                if backup_time < cutoff_time:
                    shutil.rmtree(item_path)
                    print(f"Deleted old backup: {item_path}")
                    deleted_count += 1
        except (ValueError, OSError) as e:
            print(f"Error processing item '{item_name}' for cleanup: {e}")

    if deleted_count > 0:
        print(f"Cleaned up {deleted_count} old backups.")
    else:
        print("No old backups found to cleanup.")

def main():
    """Automates the backup process with optional cleanup."""
    source_directory = "/path/to/your/source"  # Replace with your source directory
    destination_directory = "/path/to/your/backups"  # Replace with your backup directory
    retention_period_days = 7

    # Ensure backup directory exists
    os.makedirs(destination_directory, exist_ok=True)

    create_backup(source_directory, destination_directory)
    cleanup_old_backups(destination_directory, retention_period_days)
```

```
if __name__ == "__main__":
    main()
```

To use this:

1. ***Important: Replace `/path/to/your/source` and `/path/to/your/backups` with the actual paths on your system.***
2. ***Run the Python script. It will create a timestamped copy of your source directory in the backup directory. It will also then clean up any backups older than 7 days (you can adjust `retention_period_days`).***

Scenario 5: Reading Infrastructure Details from YAML and Generating a CSV Report

This script reads infrastructure details (like server names, IP addresses, roles) from a YAML file and generates a CSV report containing specific information.

```
import yaml
import csv
import os
from datetime import datetime

def load_infrastructure_config(config_file):
    """Loads infrastructure configuration from a YAML file."""
    try:
        with open(config_file, 'r') as f:
            return yaml.safe_load(f)
    except FileNotFoundError:
        print(f"Error: Configuration file '{config_file}' not found.")
        return None
    except yaml.YAMLError as e:
        print(f"Error parsing YAML file: {e}")
        return None

def generate_infrastructure_report(config_data, output_csv):
    """Generates a CSV report from the infrastructure configuration data."""
    if not config_data:
        return

    report_data = [["Server Name", "IP Address", "Role", "Region"]]

    for server in config_data.get('servers', []):
        server_name = server.get('name', 'N/A')
        ip_address = server.get('ip', 'N/A')
        role = server.get('role', 'N/A')
        region = server.get('location', {}).get('region', 'N/A')
        report_data.append([server_name, ip_address, role, region])

    try:
        with open(output_csv, 'w', newline='') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerows(report_data)
        print(f"Infrastructure report generated at '{output_csv}'")
    except Exception as e:
        print(f"Error writing to CSV file: {e}")

def main():
    """Loads infrastructure config and generates a CSV report."""
    config_file = "infrastructure.yaml"
    output_file = "infrastructure_report.csv"

    infrastructure_config = load_infrastructure_config(config_file)
    if infrastructure_config:
        generate_infrastructure_report(infrastructure_config, output_file)

if __name__ == "__main__":
    main()
```

To use this:

Create a YAML file named `infrastructure.yaml` with details about your infrastructure. For example:

YAML

```
servers:
  - name: web-server-01
    ip: 192.168.1.10
    role: web
    location:
      datacenter: dc-east
      region: us-east-1
  - name: db-server-01
    ip: 192.168.1.20
    role: database
    location:
      datacenter: dc-east
      region: us-east-1
  - name: app-server-01
    ip: 10.0.1.5
    role: application
    location:
      datacenter: dc-west
      region: eu-west-2
```

Run the Python script. It will read the YAML file and generate a CSV report named `infrastructure_report.csv` ***containing the server names,***