Blue-Green Deployment Guide Using Two Backend Pools and a Load Balancer

Overview of Blue-Green Deployment

Blue-Green deployment allows for seamless application updates with minimal downtime. It involves two separate environments:

- **Blue Environment**: Hosts the current production version of the application.
- **Green Environment**: Hosts the new version of the application.
- **Load Balancer**: Manages traffic distribution between these environments.

This strategy ensures a smooth upgrade by gradually shifting traffic from the Blue (current) version to the Green (new) version, thoroughly testing the new version before full deployment.

Setup Architecture

Two Backend Pools

- **Backend Pool 1 (Blue)**: Hosts the current version of the application.
- **Backend Pool 2 (Green)**: Hosts the new version of the application.

Application Deployment

- Deploy **version A** of your application to the Blue backend pool (e.g., running on servers/containers with Tomcat on Ubuntu or Windows).
- Deploy **version B** to the Green backend pool, potentially in the same or a different environment.

Load Balancer

- Configure the load balancer to route traffic between the backend pools.
- Enable weighted traffic distribution, starting with 100% to Blue and 0% to Green.

Deployment Steps

Backend Pool Setup

- **Blue Pool**: Assign servers running version A of your application and configure health probes to monitor their availability.
- **Green Pool**: Assign servers running version B of your application and thoroughly test its functionality in isolation before allowing production traffic.

Load Balancer Configuration

- **Frontend Listener**: Configure the load balancer with a listener on the desired port (e.g., 80 or 443).
- **Routing Rules**: Define rules to distribute traffic between the Blue and Green backend pools.
- **Initial Traffic Distribution**: Start with 100% traffic routed to Blue and 0% to Green.
- **Health Probes**: Set up probes to monitor the health and availability of both backend pools.

Gradual Traffic Shift

- Gradually shift traffic from Blue to Green using weighted routing:
    - Start by updating the distribution (e.g., 80% Blue, 20% Green).
    - Monitor the Green pool for errors or issues during this phase.
    - Incrementally increase traffic to the Green pool until it handles 100% of the load.

Rollback Procedure

- If any issues are detected with the Green deployment, route all traffic back to the Blue pool by resetting its weight to 100%. This ensures quick recovery without service disruption.

Example Configuration

Blue Pool (Version A)

- Backend Servers: 192.168.1.101, 192.168.1.102
- Application Version: 1.0
- Health Probe Endpoint: `/health`

Green Pool (Version B)

- Backend Servers: 192.168.1.103, 192.168.1.104
- Application Version: 2.0
- Health Probe Endpoint: `/health`

Traffic Rules

Backend Pool	Initial Weight	Final Weight
Blue	100%	0%
Green	0%	100%

Key Considerations

- **Testing**: Always test the Green backend pool thoroughly before shifting production traffic.
- **Monitoring**: Use monitoring tools to track the health, performance, and logs of both pools during and after the deployment process.
- **Automation**: Incorporate CI/CD pipelines to streamline deployments, testing, and traffic shifts.

Traffic Shifting Using Azure CLI with a Bash Script

Steps to Traffic Shift with Azure CLI

1. Prerequisites

- **Install Azure CLI**: Ensure the Azure CLI is installed on your system.
- **Login to Azure**:
  ```
  az login
  ```
- **Set Default Subscription** (if needed):
  ```
  az account set --subscription "YourSubscriptionName"
  ```

2. Identify Required Resources

- **Load Balancer Name**: The name of your load balancer.
- **Resource Group**: The resource group containing the load balancer.
- **Backend Pools**: Names of the Blue (current) and Green (new) backend pools.
- **Rule Name**: The load balancing rule managing traffic distribution.

3. Modify Traffic Weights

Azure Load Balancer doesn't support direct traffic weighting. Instead, traffic shifting can be achieved by:

- Adding or removing backend pool members.
- Modifying health probe configurations.
- Switching backend pools for the load balancer rule.

Example Bash Script: Traffic Shifting

Scenario:

- Load Balancer Name: `myLoadBalancer`
- Resource Group: `myResourceGroup`
- Backend Pool Blue: `blueBackendPool`
- Backend Pool Green: `greenBackendPool`
- Rule Name: `httpRule`

```
#!/bin/bash

# Variables
```

```
RESOURCE_GROUP="myResourceGroup"
LOAD_BALANCER="myLoadBalancer"
BLUE_POOL="blueBackendPool"
GREEN_POOL="greenBackendPool"
RULE_NAME="httpRule"

# Function to associate a rule with a specific backend pool
update_backend_pool() {
    local BACKEND_POOL=$1
    echo "Updating Load Balancer Rule to use Backend Pool: $BACKEND_POOL"

    # Update the load balancer rule
    az network lb rule update \
        --resource-group $RESOURCE_GROUP \
        --lb-name $LOAD_BALANCER \
        --name $RULE_NAME \
        --backend-pool-name $BACKEND_POOL

    if [ $? -eq 0 ]; then
        echo "Traffic successfully shifted to $BACKEND_POOL."
    else
        echo "Failed to update backend pool."
        exit 1
    fi
}

# Check current traffic pool (optional)
echo "Current Load Balancer Rules:"
az network lb rule list \
    --resource-group $RESOURCE_GROUP \
    --lb-name $LOAD_BALANCER \
    --output table

# Shift traffic to Green pool
update_backend_pool $GREEN_POOL

# Optional: Monitor health probes
echo "Monitoring health probes for $GREEN_POOL..."
az network lb probe list \
    --resource-group $RESOURCE_GROUP \
    --lb-name $LOAD_BALANCER \
    --output table
```

4. Explanation of the Script

1. Backend Pool Switching

- The `az network lb rule update` command updates the load balancer rule to point to the specified backend pool.
- Initially, the rule is associated with `blueBackendPool`. Switching to `greenBackendPool` redirects traffic.

2. Health Probes Monitoring

- The `az network lb probe list` command displays the health status of backend instances in the load balancer.

3. Flexibility

- The script can be extended to enable gradual traffic shifts by modifying backend pool configurations or instance availability.

5. Considerations

- **Azure Application Gateway**: For Azure Application Gateway, Weighted Traffic Routing and Rewrite Rules can facilitate gradual shifts.
- **Downtime Prevention**: Configure health probes to avoid routing traffic to unhealthy instances.
- **Monitoring**: Use tools like Azure Monitor or Application Insights for real-time feedback during deployment.

Gradual Traffic Shift Script for Azure Load Balancer

Steps for Gradual Traffic Shifting with Log Monitoring

1. **Initial Setup**:
2. Begin with App A (Blue) serving 100% of traffic while App B (Green) is idle.
3. **Gradual Traffic Shifting**:

   Incrementally decrease traffic to App A and increase it for App B using weighted traffic rules.
4. **Log Monitoring**:

   Verify App B's health and performance through application logs or monitoring endpoints.
5. **Finalize Shift**:

   If App B is healthy, reduce App A's traffic to 0%, fully shifting to App B.

Script: Gradual Traffic Shift with Log Monitoring

Scenario:

- **Load Balancer Name**: `myLoadBalancer`
- **Resource Group**: `myResourceGroup`
- **App A (Blue)**: `blueBackendPool`
- **App B (Green)**: `greenBackendPool`
- **Rule Name**: `httpRule`
- **Log Endpoint**: `http://app-b.example.com/logs`

```
#!/bin/bash

# Variables
RESOURCE_GROUP="myResourceGroup"
LOAD_BALANCER="myLoadBalancer"
BLUE_POOL="blueBackendPool"
GREEN_POOL="greenBackendPool"
RULE_NAME="httpRule"
LOG_ENDPOINT="http://app-b.example.com/logs"
```

```
HEALTH_CHECK_RETRIES=5
TRAFFIC_INCREMENT=20

# Function to monitor App B's logs
check_logs() {
    local retries=0

    echo "Checking logs for App B..."

    while [ $retries -lt $HEALTH_CHECK_RETRIES ]; do
        # Replace with a real log check; for example, fetching logs via curl or other
monitoring tools
        response=$(curl -s -o /dev/null -w "%{http_code}" $LOG_ENDPOINT)

        if [ "$response" == "200" ]; then
            echo "App B is healthy (Log endpoint returned 200)."
            return 0
        fi

        echo "Log check failed. Retrying... ($retries/$HEALTH_CHECK_RETRIES)"
        retries=$((retries + 1))
        sleep 5
    done

    echo "App B logs indicate failure. Aborting traffic shift."
    return 1
}

# Function to update traffic distribution
update_traffic() {
    local blue_weight=$1
    local green_weight=$2

    echo "Updating traffic: Blue = ${blue_weight}%, Green = ${green_weight}%"
```

```
    az network lb rule update \
        --resource-group $RESOURCE_GROUP \
        --lb-name $LOAD_BALANCER \
        --name $RULE_NAME \
        --backend-pool-name $BLUE_POOL \
        --set backend_address_pool.weights.blue=$blue_weight \
        --set backend_address_pool.weights.green=$green_weight

    if [ $? -eq 0 ]; then
        echo "Traffic successfully updated."
    else
        echo "Failed to update traffic. Exiting."
        exit 1
    fi
}

# Initial traffic split
blue_weight=100
green_weight=0

# Start gradual traffic shift
while [ $blue_weight -gt 0 ]; do
    # Shift traffic
    update_traffic $blue_weight $green_weight

    # Check App B logs
    check_logs
    if [ $? -ne 0 ]; then
        echo "Reverting traffic to App A only."
        update_traffic 100 0
        exit 1
    fi

    # Adjust weights
    blue_weight=$((blue_weight - TRAFFIC_INCREMENT))
    green_weight=$((green_weight + TRAFFIC_INCREMENT))

    # Wait for traffic to stabilize
    sleep 10
done

echo "Traffic fully shifted to App B."
```

Steps Explained

1. **Log Monitoring**:
    a. The `check_logs` function verifies App B's health by checking the response from a log endpoint (`$LOG_ENDPOINT`).
    b. Retries are configurable through `$HEALTH_CHECK_RETRIES`.
2. **Gradual Traffic Shift**:
    a. Traffic is adjusted incrementally using `$TRAFFIC_INCREMENT`.
    b. The script modifies backend pool weights with `az network lb rule update`.
3. **Rollback**:
    a. If App B fails the log check, the script restores 100% traffic to App A (Blue).
4. **Traffic Weights**:
    a. Gradual changes in traffic percentages (e.g., 80%-20%, 60%-40%) ensure stability during the transition.

Requirements

1. **Weighted Traffic Support**:

Confirm that your load balancer supports weighted traffic routing (e.g., Azure Load Balancer or Application Gateway).

2. **Health Probes**:

Proper health probes must be configured for backend pools.

3. **Log Endpoint**:

A valid endpoint to verify App B's health.