Yash Anil Mutatkar Nov 9 2024

Objective: Implement a blue-green deployment strategy using AWS EC2 instances and an AWS Application Load Balancer (ALB), automated through Terraform and GitHub Actions.

Overview
I successfully implemented a blue-green deployment strategy to ensure zero downtime during application updates. By using two identical EC2 instances (blue and green environments) and an ALB, I set up a system where traffic is routed to one environment (blue) while updates are made to the other (green). GitHub Actions were utilized to automate deployment, allowing for seamless traffic switching and rollbacks in case of any issues.

Tools Used

- **AWS EC2**: Provisioned two identical EC2 instances to represent blue and green environments.
- **AWS Application Load Balancer (ALB)**: Used to route traffic between the blue and green EC2 instances.
- **Terraform**: Used to automate the provisioning and management of the infrastructure (EC2 instances, ALB, etc.).
- **GitHub Actions**: Automated deployment processes to update the blue environment and switch traffic between environments.

Implementation Steps

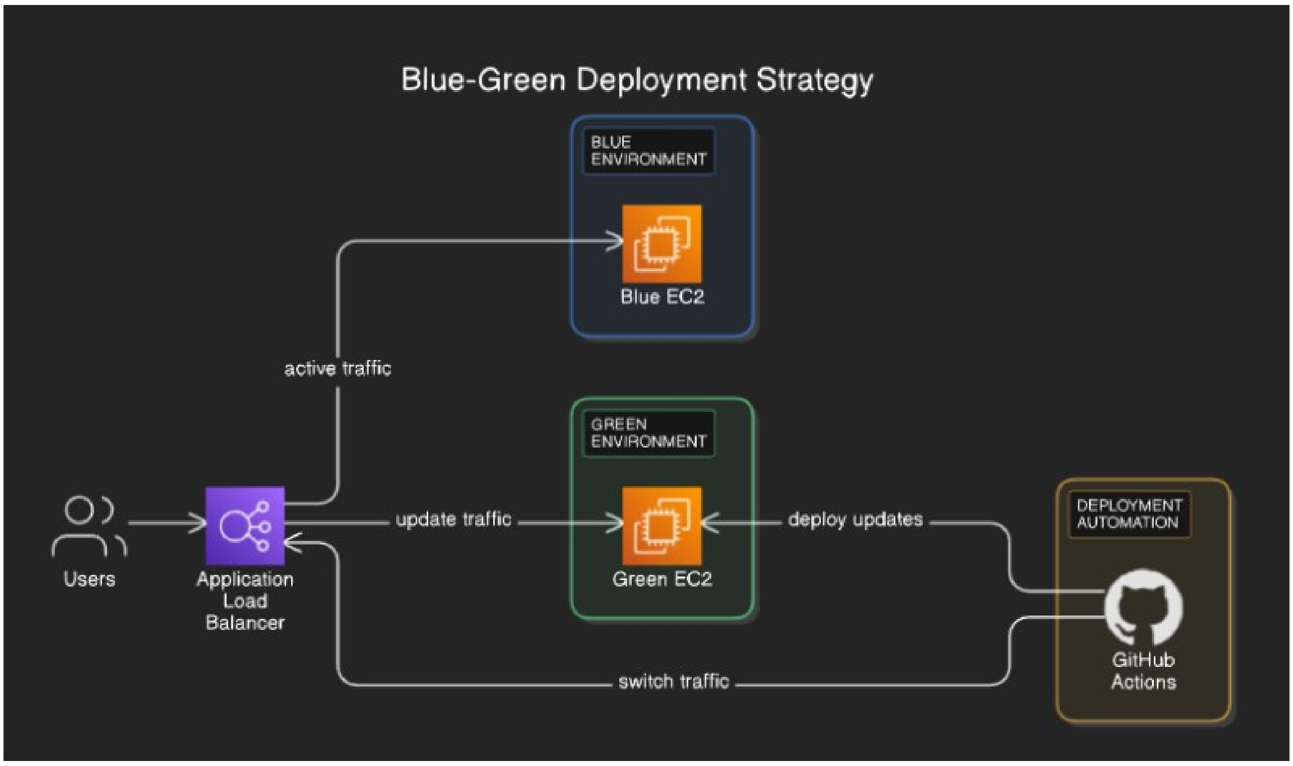
1. **Provision EC2 Instances Using Terraform**
    - First, I used Terraform to create two identical EC2 instances across different subnets, representing the **blue** and **green** environments.
    - These instances were placed in two separate availability zones to ensure high availability.
2. **Set Up Application Load Balancer (ALB)**
    - I configured an **Application Load Balancer (ALB)** to distribute traffic between the blue and green EC2 instances.
    - I created **two target groups** (one for blue and one for green) and registered the EC2 instances under the appropriate target group.
3. **Configure Health Checks**
    - I set up health checks for both target groups to ensure that the load balancer only routes traffic to healthy instances.
    - The health check checks the status of the web service running on both the blue and green EC2 instances.
4. **Automate Deployment Using GitHub Actions**
    - I configured **GitHub Actions** to automate the deployment process.
    - Whenever new changes are pushed, the action triggers, updates the blue environment with the latest application version, and performs a health check to confirm that the deployment is successful.

- If the blue environment is updated without issues, I manually switched traffic from the blue to the green environment. This switch was achieved by modifying the load balancer's target group routing.

5. **Test Traffic Switch**
   - Once the deployment was complete and the environments were set up, I tested the traffic switch through the **ALB DNS name**.
   - Requests were made to the ALB DNS name, and I observed that traffic was routed first to the blue environment.
   - After a successful deployment to the green environment, I updated the ALB to route traffic to the green environment, confirming the seamless switch.

Conclusion

The blue-green deployment strategy I implemented successfully ensured that I could deploy updates to the application with minimal downtime. By leveraging AWS EC2, ALB, Terraform, and GitHub Actions, I automated the entire process from provisioning to deployment, making it easier to manage updates while ensuring high availability. The process was tested thoroughly by switching traffic between environments, ensuring that the blue-green deployment strategy worked as expected without any disruptions to the end users.


Blue-Green Deployment Strategy
BLUE ENVIRONMENT
Blue EC2
active traffic
GREEN ENVIRONMENT
Green EC2
Users
Application Load Balancer
update traffic
deploy updates
DEPLOYMENT AUTOMATION
GitHub Actions
switch traffic