😊 Blue Green Deployment

Blue-green deployment is a release management strategy designed to minimize downtime and reduce risks associated with deploying new versions of software. The technique involves maintaining two identical production environments, referred to as "blue" and "green." Only one environment serves live production traffic at any given time.

How Blue-Green Deployment Works:

1. **Two Environments**:
    - **Blue Environment**: The current live production environment, serving all user traffic.
    - **Green Environment**: An identical environment that is not currently serving user traffic but is ready for deployment.
2. **Deployment to the Green Environment**:
    - The new version of the software is deployed to the green environment.
    - Extensive testing is performed on the green environment to ensure the new release is stable and functions as expected. This includes integration, user acceptance, and performance testing.
3. **Traffic Switch**:
    - Once testing is successful, traffic is redirected from the blue environment to the green environment.
    - This switch can be instantaneous or gradual (a canary release can be used to slowly direct a percentage of traffic to the green environment before fully switching over).
4. **Monitoring**:
    - After the switch, the green environment is monitored closely for any issues or bugs.
    - If problems arise, traffic can be quickly redirected back to the blue environment, which still holds the previous stable version of the software.
5. **Clean Up**:
    - If the new version in the green environment is confirmed to be stable and performing well, the blue environment can be updated with the next release or decommissioned.
    - This alternating pattern continues for future deployments.

Advantages:

- **Zero Downtime**: Because the switch between environments is seamless, downtime is either nonexistent or minimal, providing continuous availability.

- **Easy Rollback**: If a problem is detected with the new version, reverting to the old version is as simple as redirecting traffic back to the blue environment.
- **Thorough Testing**: The new version is tested in a production-like environment (green) before it goes live, reducing the chances of unforeseen issues.

Challenges:

- **Resource Intensive**: Maintaining two fully operational environments can be costly in terms of infrastructure, especially for large-scale applications.
- **Operational Complexity**: Ensuring both environments are identical and keeping them synchronized can be complex, especially when dealing with databases and other stateful services.

Use Cases:

- **High-Availability Systems**: Systems that require near-zero downtime, such as online services, e-commerce platforms, and financial services.
- **Frequent Updates**: Environments where new features or updates are released frequently and need to be quickly and safely deployed.

Blue-green deployment is often used in conjunction with other DevOps practices, such as continuous integration and continuous deployment (CI/CD), to streamline the release process and ensure smooth, reliable deployments.

The process described involves updating a web application using AWS CodeDeploy with a blue-green deployment strategy. The goal is to release a new version of the application with minimal downtime and risk.

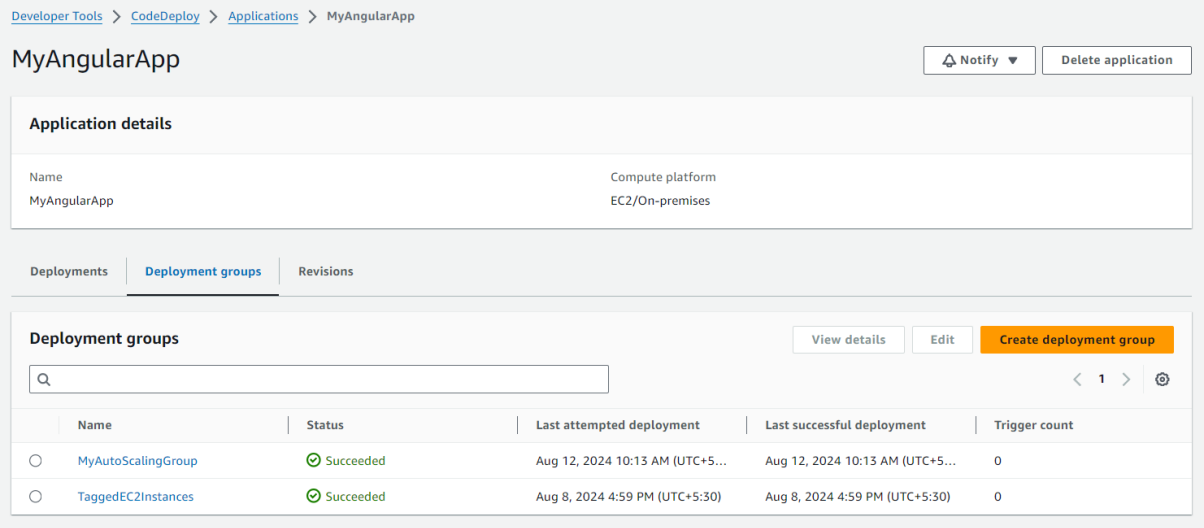
First, the deployment settings are configured for blue-green deployment, ensuring that the new version (green) is deployed while the current version (blue) remains live. After making code changes in the application, the new version is pushed to the repository, triggering the deployment pipeline. The deployment process then creates a new set of instances (green fleet) identical to the existing ones (blue fleet) and deploys the new version to them.

Once the green fleet is up and running, traffic is gradually shifted from the blue fleet to the green fleet, ensuring a smooth transition. Finally, the old blue fleet is decommissioned, leaving only the new green fleet serving the updated application.

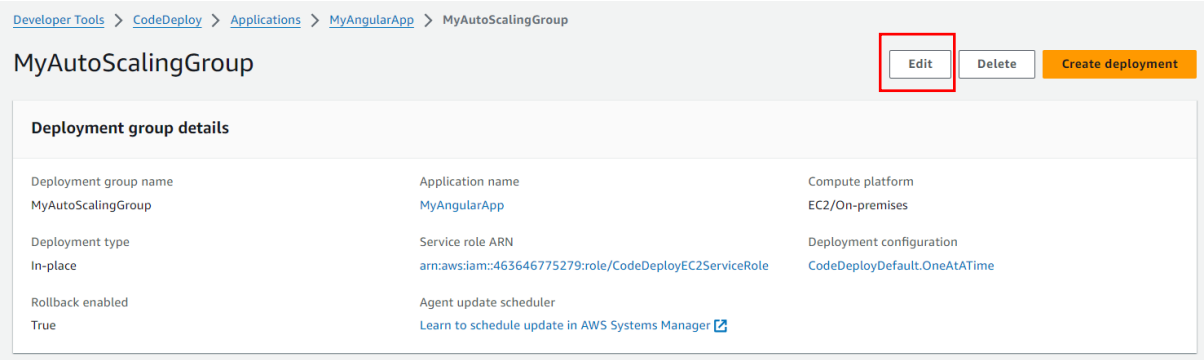
End Goal: Successfully update the web application to a new version with zero downtime, ensuring that users experience a seamless transition without any interruptions.

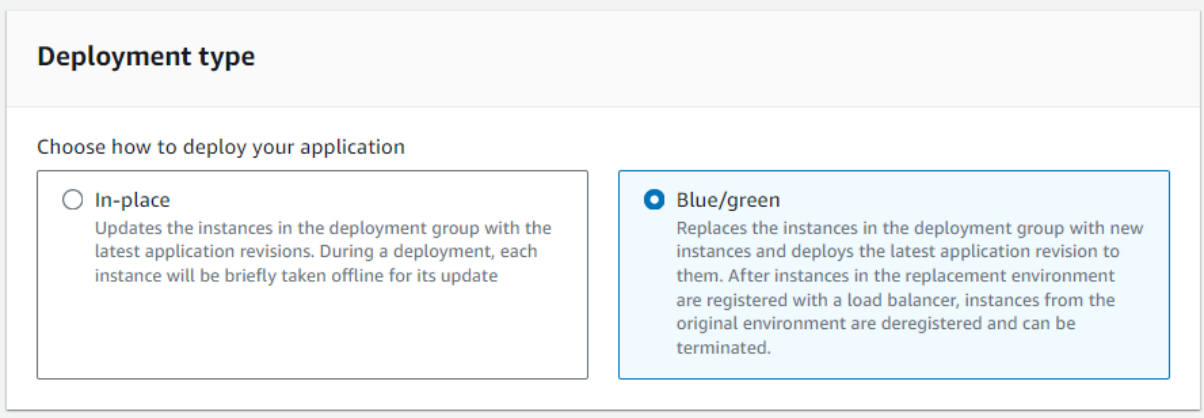
😄 To begin with the Lab:

1. Now in your AWS Console go to CodeDeploy Applications and open the deployment group. So, when you open your Application, you can see your Deployment groups open the auto-scaling deployment group.

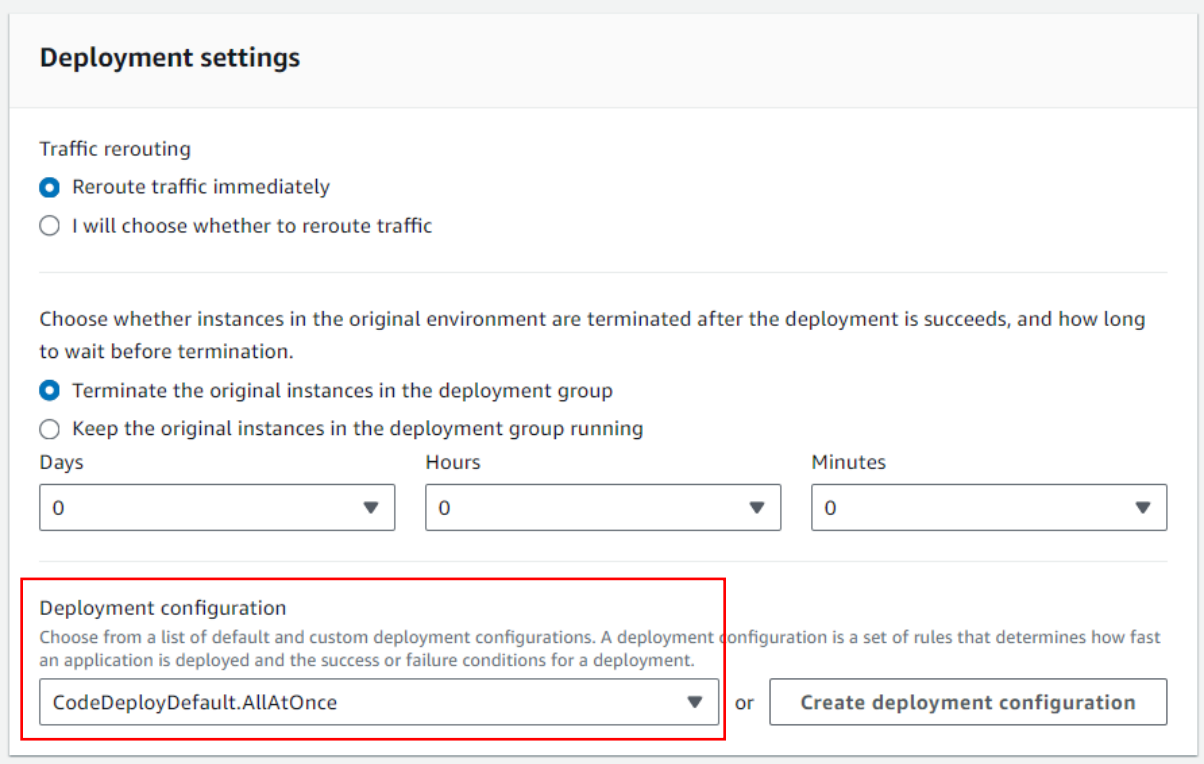


2. Now you need to click on Edit, then scroll down to deployment type and choose Blue/green deployment.

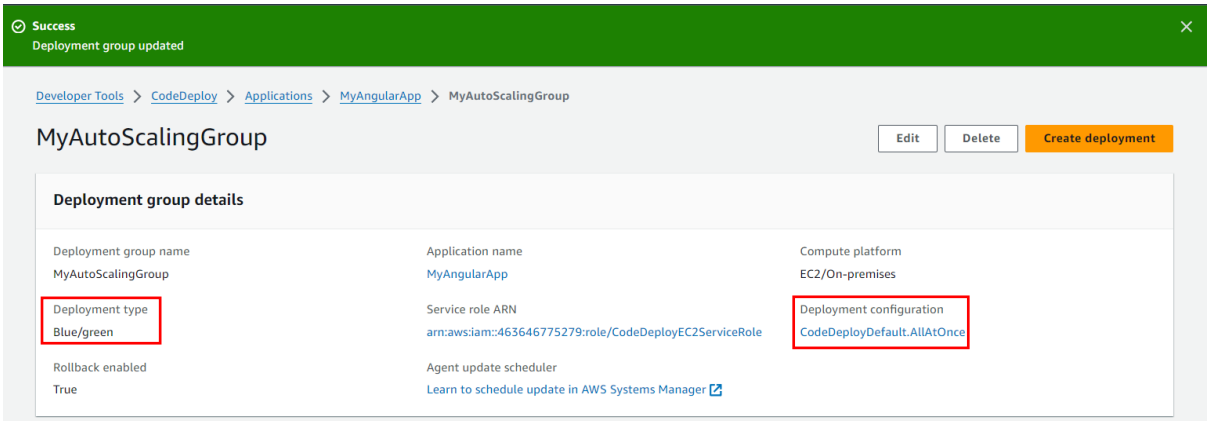




3. Now scroll down to deployment settings and choose deployment configuration as all at once.
4. After that save your changes.



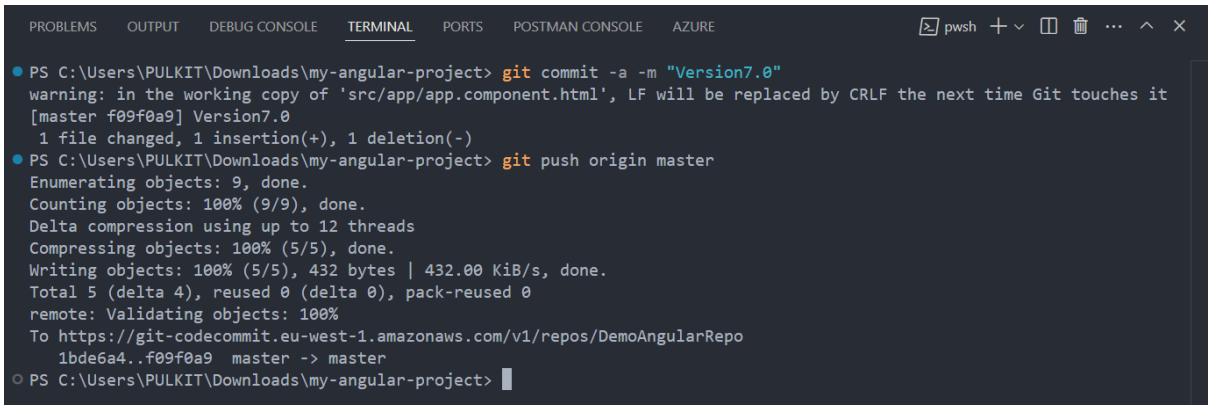
5. Below you can see that our changes have been saved.



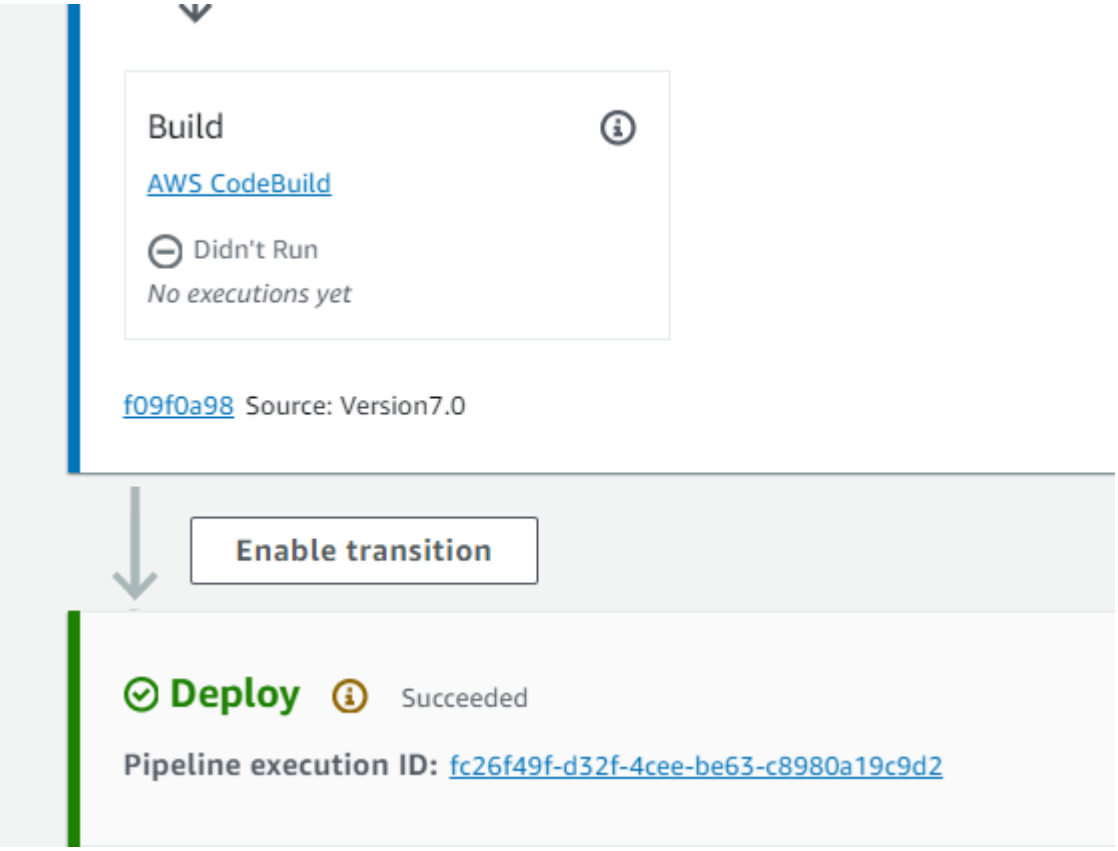
6. Next, we'll release a new version of our web application and trigger a deployment to see how it works.
7. So, open your Angular application in VS Code. Now open the app.component.html file and change the version.

8. Now save the changes and open the terminal, then you need to commit the changes and push them to your repository.

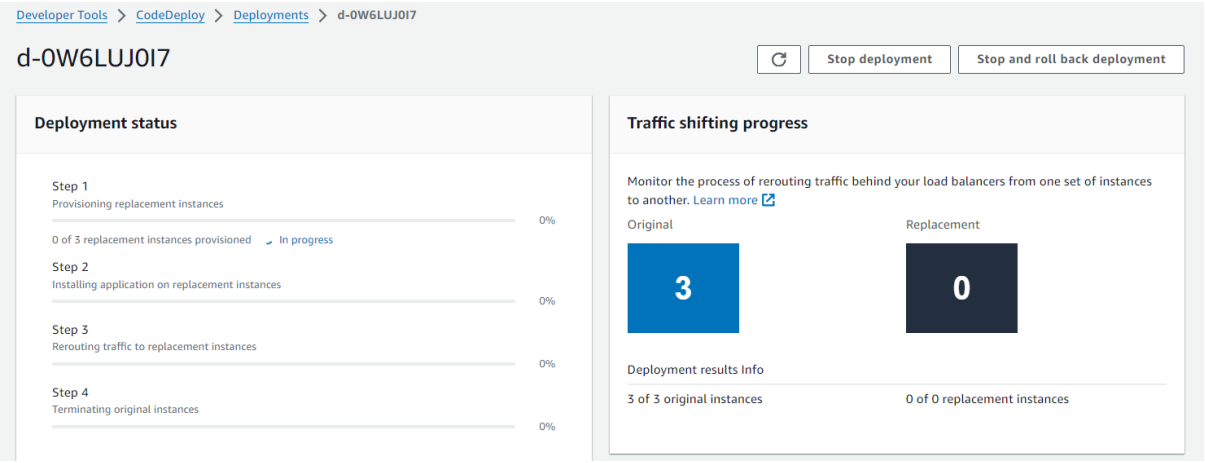
   git commit -a -m “Version7.0”
   git push origin master

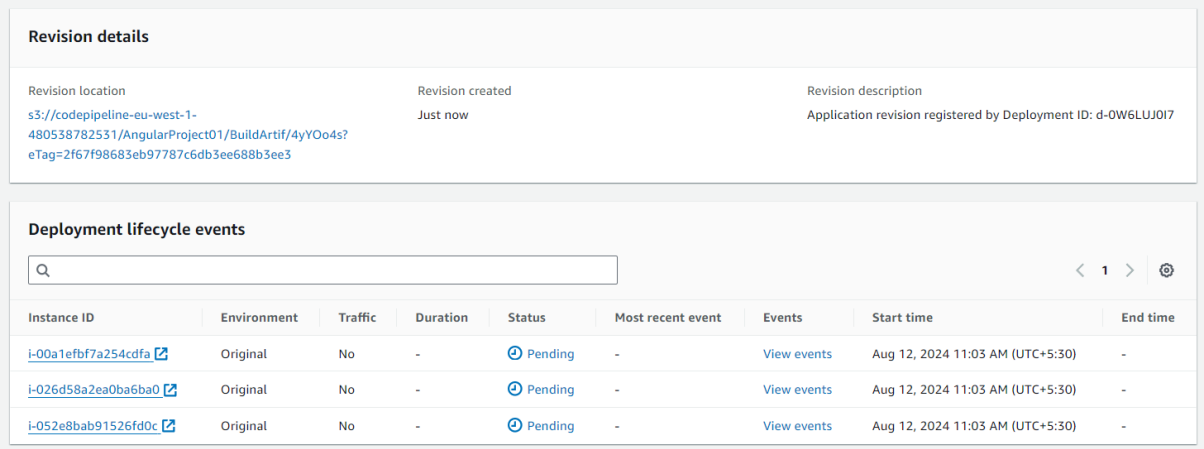


9. After that go to your Pipeline and as soon as it starts you need to disable the transition between your build stage and deploy. So, that you can enable it later and check the deployment for yourself like we did in our previous labs.
10. So, when the build stage is executed, you need to enable the transition and wait until the view details option is available in the deployment stage. Then you need to click on it and go to CodeDeploy.

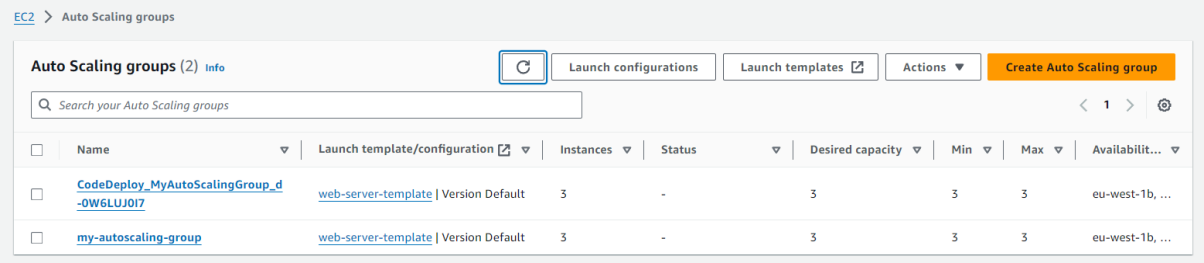


11. Now we have enabled the transition, and we are in CodeDeploy by clicking on view in the CodeDeploy option from our deploy stage.
12. As you see, this time the 'Deployment status' section is different. We see 4 steps of action in the deployment status. Now, we are in the first step. The deployment is provisioning new instances for replacement, or in other words, the green group. Let's scroll down. In the 'Deployment lifecycle events' section, as you see, we have only the original instances at start, the blue group. And they are all in 'Pending' state.
13. They will remain in this state until the new instances in the green fleet are registered to the load balancer.

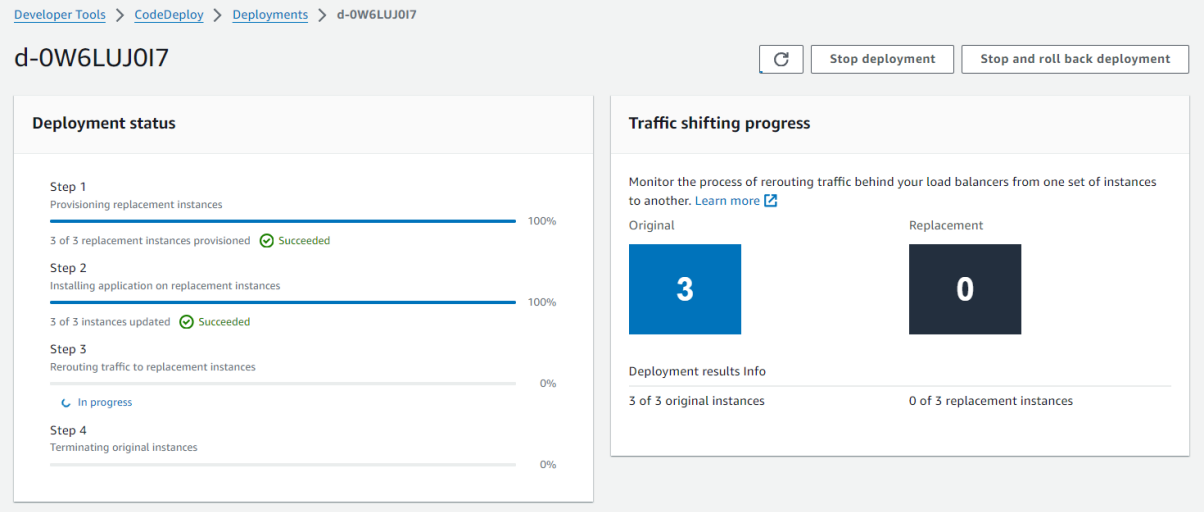




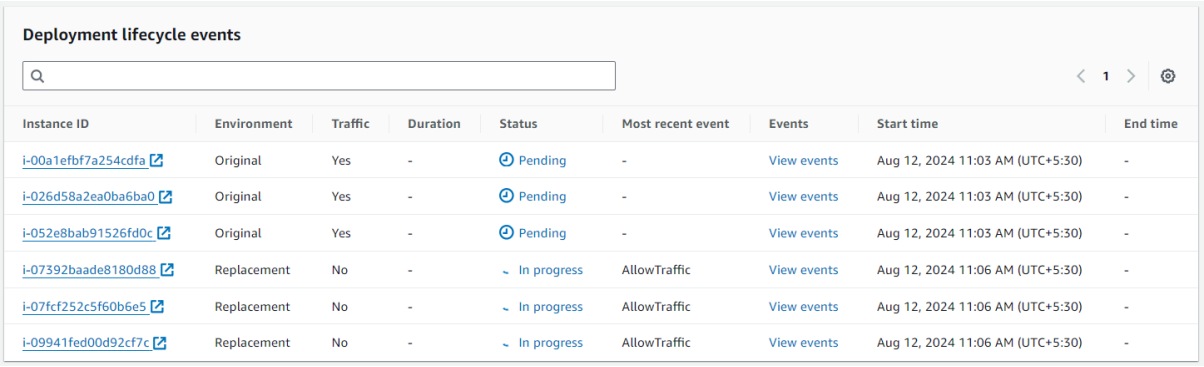
14. Now, let's switch to the EC2 auto-scaling groups to see what happens there. As you see, now we have a new auto-scaling group. Let's select it from the list. And then, let's open the 'Instances' tab. Yes, we see that it has 3 instances just like the previous one. So, this group will be the green fleet of our blue/green deployment. It was created automatically by CodeDeploy by duplicating our existing auto-scaling group. Besides, the name of the new auto-scaling group starts with CodeDeploy and contains both the deployment group name and the ID of the deployment that created it.

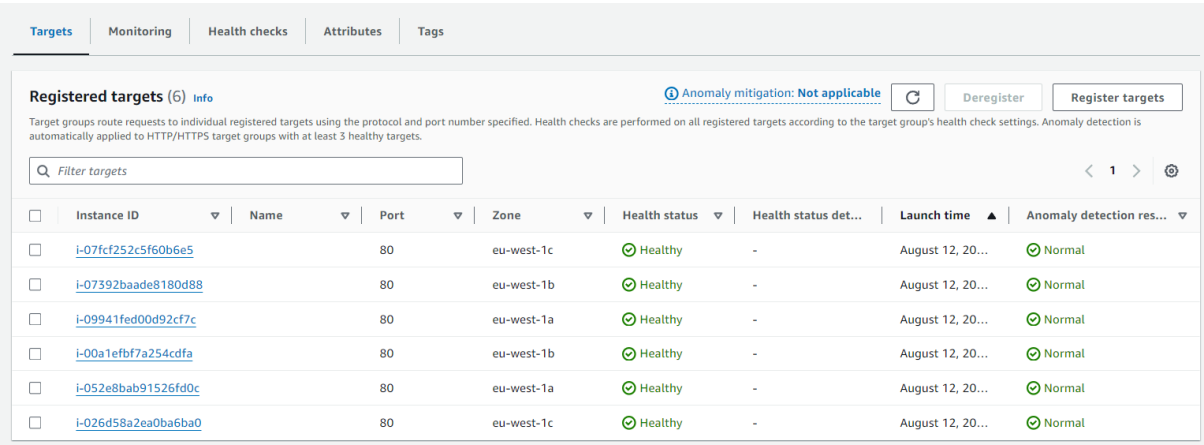


15. So, the first step succeeded. The replacement auto-scaling group and its instances were provisioned successfully. And now, we're in the second step. The new revision will be deployed to the new instances, the green fleet. In the 'Deployment lifecycle events' list, we continue to see only the older instances, the blue fleet.
16. In step 3, the traffic will be shifted from the blue fleet to the green fleet. To achieve this, firstly, the replacement instances, the green fleet will be registered to the load balancer and then the older instances, the blue fleet will be deregistered from it.

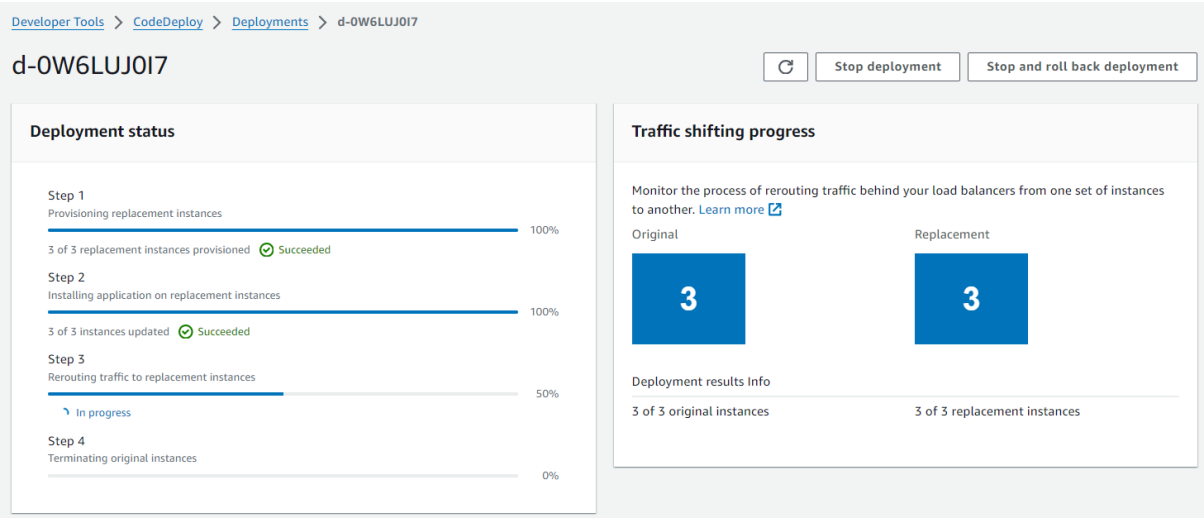


17. As you see, now we have 6 instances in the target group and the green fleet instances also seem registered successfully. Because, all the instances here are healthy. So, at this stage, until the deregistration of the blue fleet starts you will see 2 versions in production.

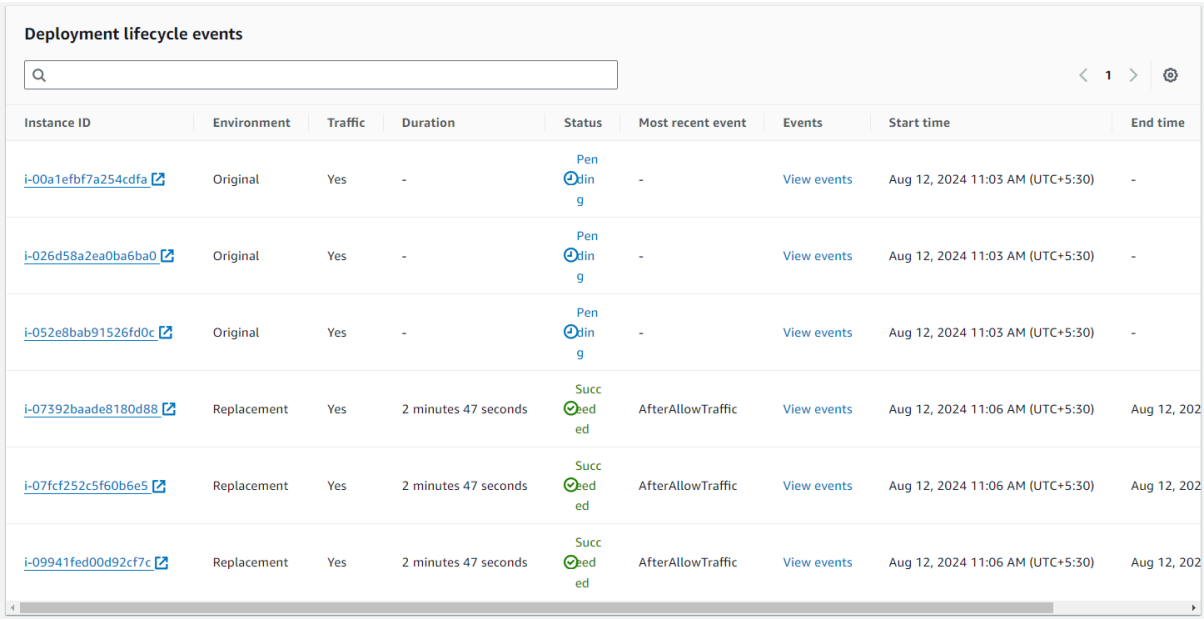




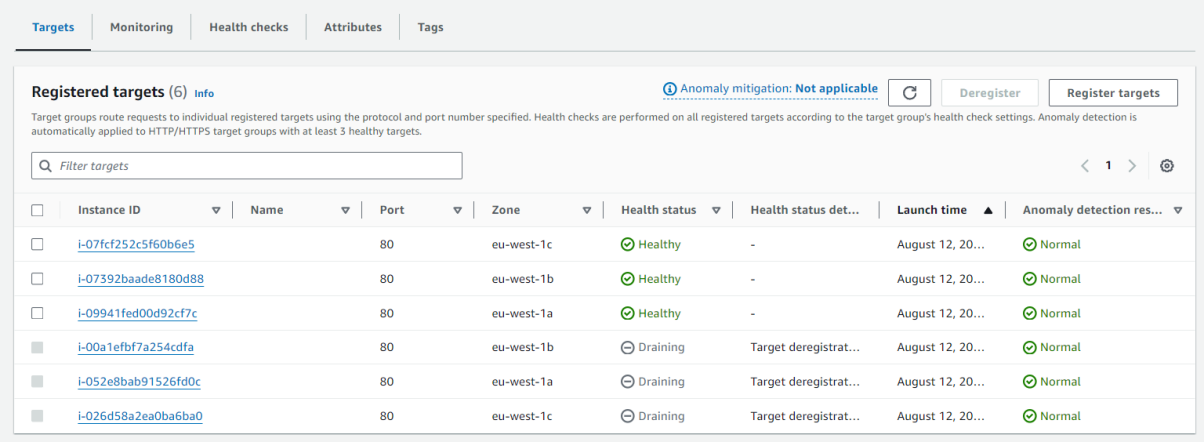
18. Now, step 3 has 50% progress. Besides, on the 'Traffic shifting progress' section the 'Replacement' box became blue as well. So, currently, both the original and the replacement instances are receiving traffic. Let's scroll down to see what is happening on the instances.



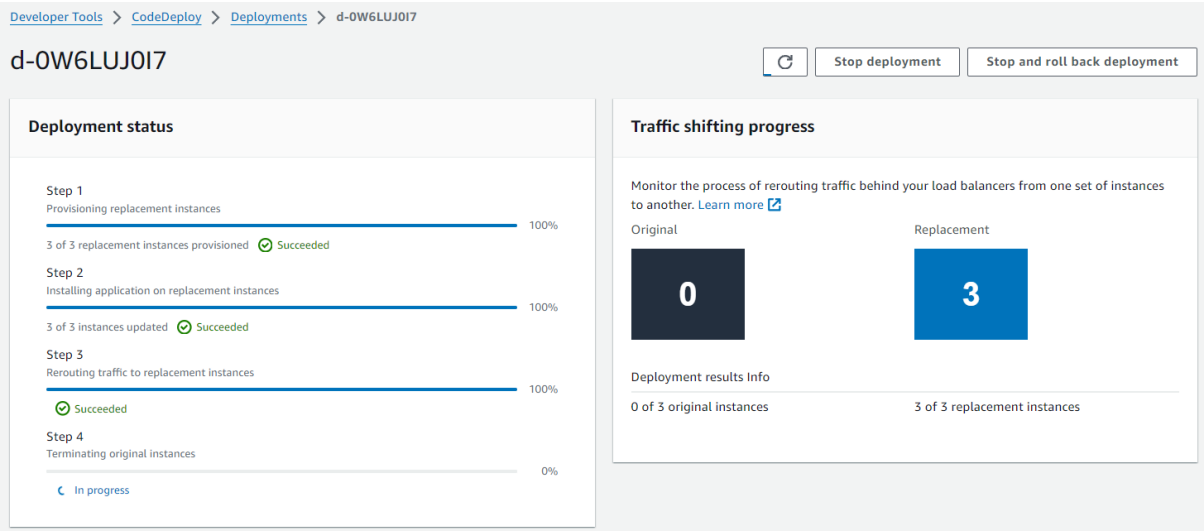
19. Now, as you see, the original instances are in progress. Their last events are BeforeBlockTraffic. So, they are being taken offline. The deployments on the replacement instances, our green gleet succeeded as well.

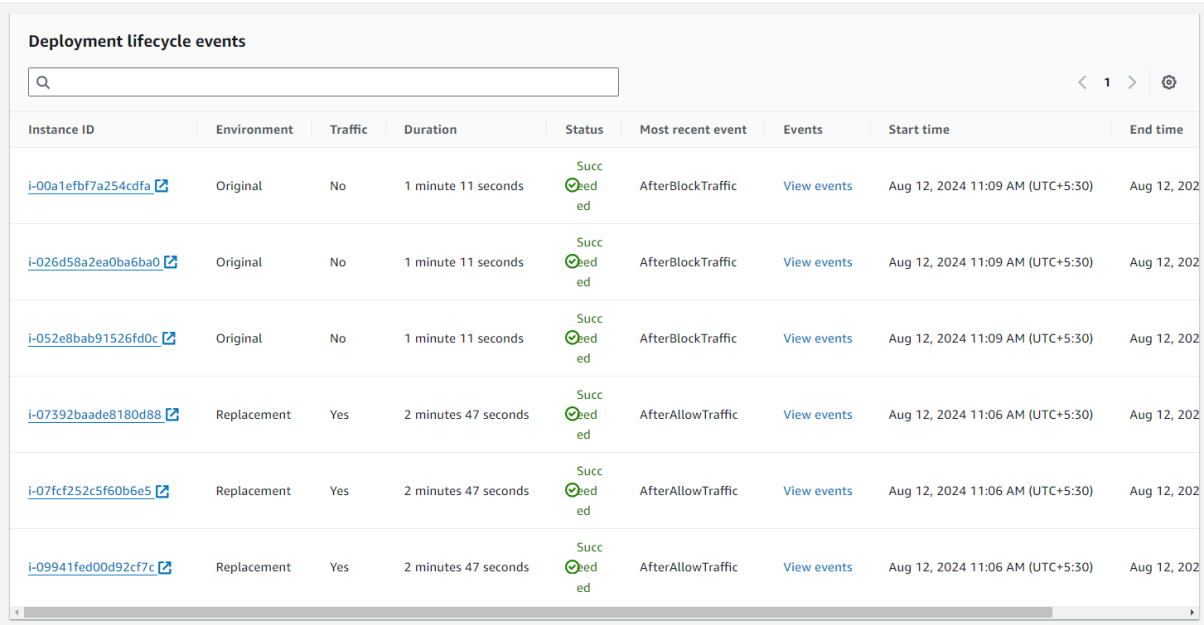


20. Now, we have 3 instances in draining state. These are the original instances, our blue fleet, and the replacement instances in our green fleet are healthy. So, currently, our website is serving only the new version. The switch from blue to green almost completed.

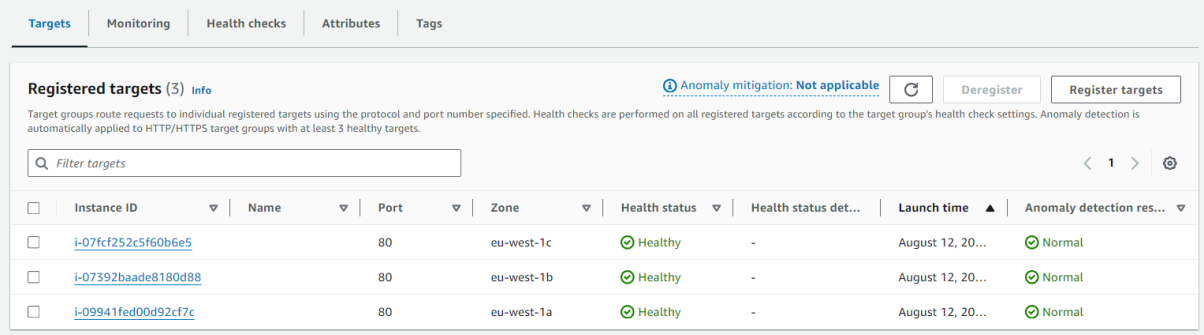


21. Step 3 also finished and our last step started afterwards. Well, it also succeeded. So, our deployment finished. Let's scroll down to the lifecycle events. As you see, now all instances are in 'Succeeded' state.

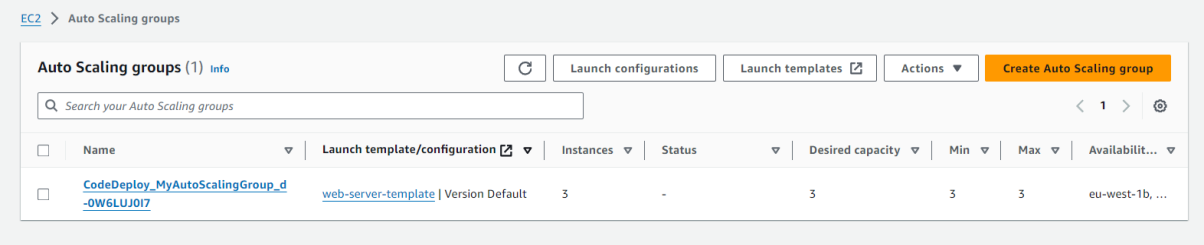




22. Now we see only the replacement instances here. And they are all healthy.



23. As you see, the original auto-scaling group we created has been deleted. The deployment just submitted its request to Amazon EC2 service and completed step 4. All instances in this group will be terminated eventually. So, now we have only one auto-scaling group remaining, the green group, which contains the name of our deployment group and the ID of our last deployment in its name.



24. Below you can see that our website is up and running.