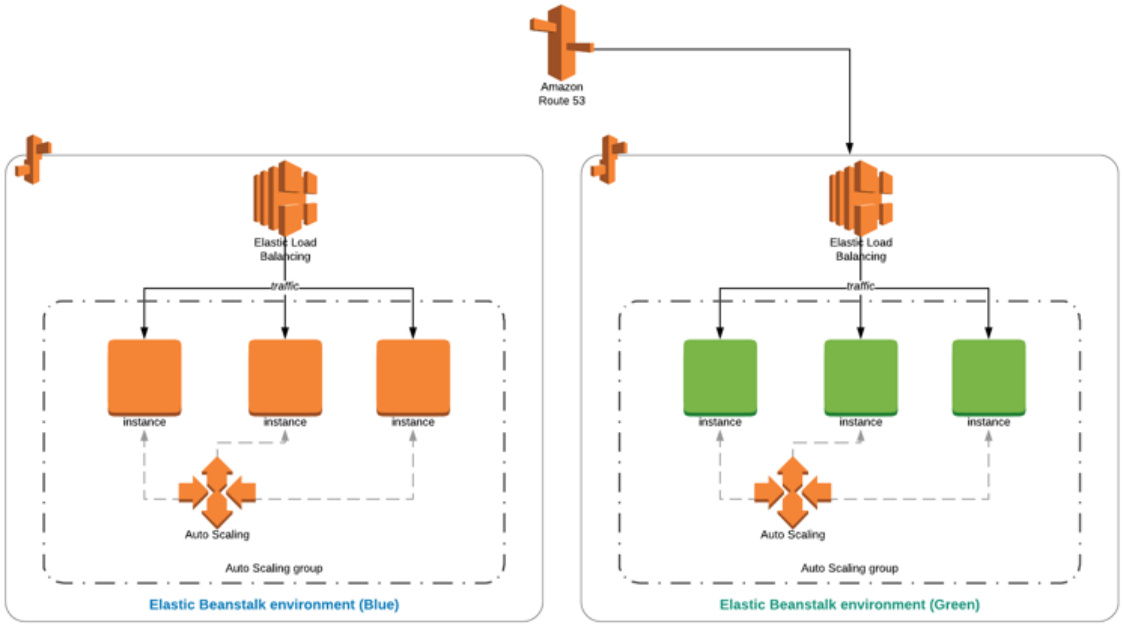
Which AWS Elastic Beanstalk Deployment Method Should You Use?



Let's say that you are a developer building awesome applications using Node.js or Python but lacking the knowledge and experience necessary to configure AWS environments. No worries! Elastic Beanstalk can make your life easier by handling configuration details. It uses preconfigured CloudFormation templates and provisions a scalable, load-balanced, and reliable environment for your application. It supports lots of programming languages, giving you less control but less worry in return. You can start from a single instance and make your architecture grow into a multi-instance cluster.

Although it may sound simple until now, actually Elastic Beanstalk is more than this. It provides you many deployment options, and you can select one of them depending on your environment and use case. In this post, I will explain the options and when to consider them.

Some words on Elastic Beanstalk

Elastic Beanstalk currently supports multiple programming platforms such as Go, Java SE, .NET on Windows Server with IIS, Node.js, PHP, Python, Ruby… Each of these platforms supports multiple versions and configurations. For example, you can choose `Ruby 2.5 with Puma version 2.8.4` or `Ruby 2.3 with Passenger version 2.8.4` or another configuration, whichever suits your needs. In addition, it supports Docker containers allowing you to host custom applications using Docker. You can also create custom platforms or customize your environment using .ebextensions.

You have two types of environments on Elastic Beanstalk: Web server environments and worker environments. The web server environment is designed to serve web applications on

the Internet, whereas worker environments are intended for background SQS message processing for decoupling your applications. In this post, our examples will be for a web server environment.

As usual, you have an application on Elastic Beanstalk and its versions. You can deploy the same version to multiple environments, but an environment can have only one application version running. When you create an environment, Elastic Beanstalk creates a CloudFormation stack behind the scenes, and this stack is visible on the CloudFormation console. Similarly, when you update an environment configuration or make a deployment, it creates additional temporary stacks or updates the current stack accordingly.

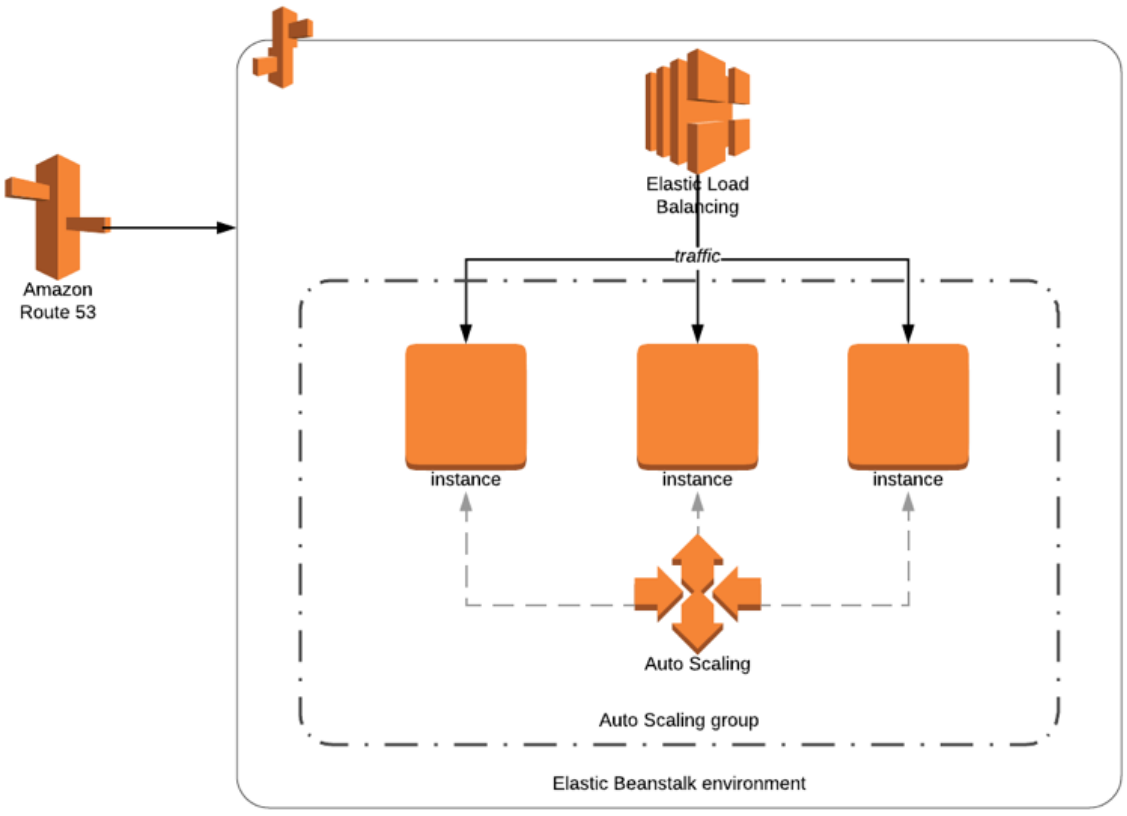
As you can see, the primary purpose of Elastic Beanstalk is to simplify the provisioning and deployment processes for developers. However, it also supports complex deployment flows as I will describe below.

Base Scenario

Before continuing with the first deployment method, let's describe our sample environment component:

- An Elastic Load Balancer distributing traffic to the instances
- 3 EC2 instances behind EC2 Autoscaling configured to scale a minimum of 3 and a maximum of 6 instances.

You can see the diagram below.

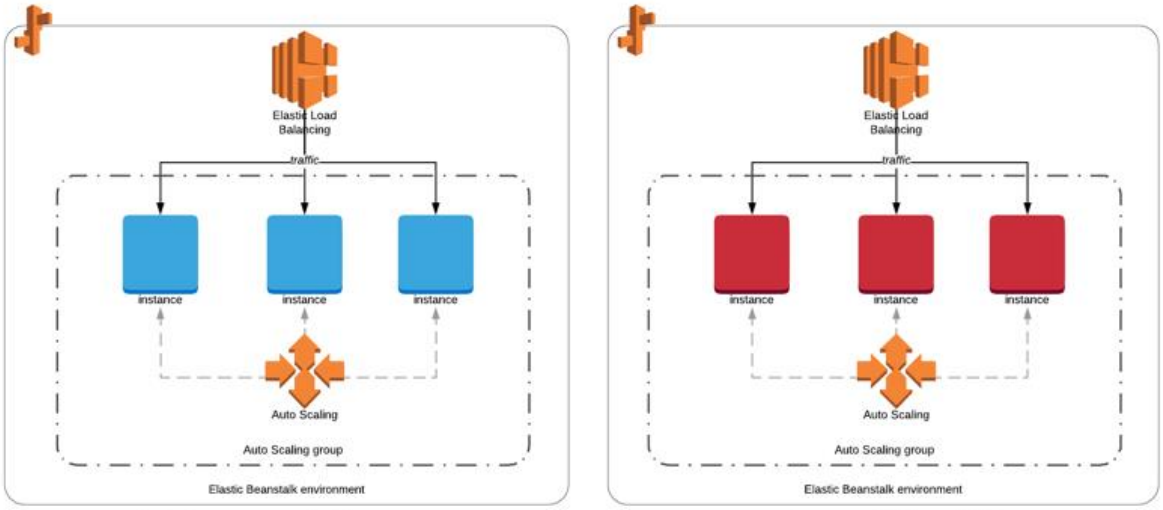


Now, let start describing the deployment methods Elastic Beanstalk provides to us.

All At Once: Let's deploy them to all!

As the name suggests, when you deploy your application in `All At Once` mode in Elastic Beanstalk, it starts the deployment on all EC2 instances simultaneously. In our case, all three instances start deploying.

It can be helpful for fast deployments, but what if your deployment is faulty and fails? It can always happen and will happen. Then, all your instances will fail, and your load balancer will not be able to serve traffic because of downtime. Both of them are a disaster if you host a production application. In this case, you need to deploy the previous version from application versions and wait for them to complete.



1. Deployment starts in all instances

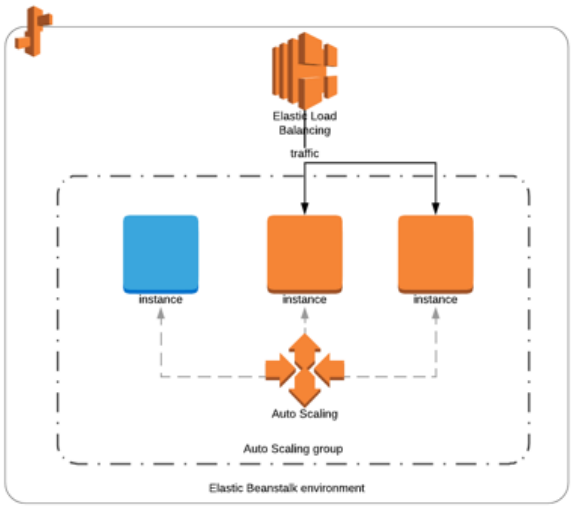
2. In case of failure, all fails!

Why use `All At Once`? I can only think for development and test environment purposes. Downtimes due to failed deployments should not affect your business use case.

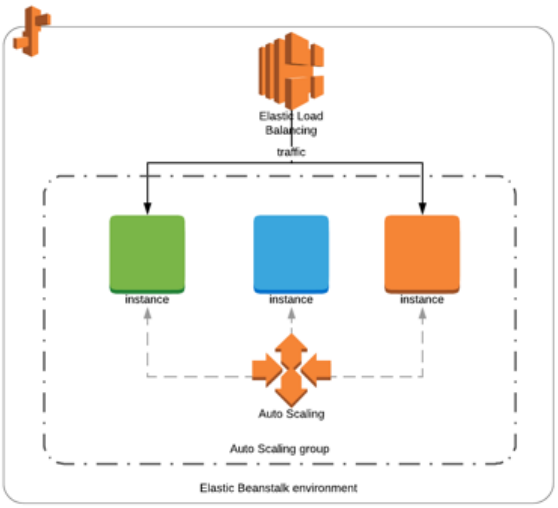
Rolling: Let's be more cautious and deploy one by one!

You can mitigate `all fail` mode of `all at once` by rolling deployments and switch to `only one batch fails` mode. When you deploy in the `Rolling` mode, you also define the number of instances to be grouped in a batch. For example, let's define the batch size as 1.

In the beginning, Elastic Beanstalk deploys the application version to the first batch and proceeds to next after it succeeds.



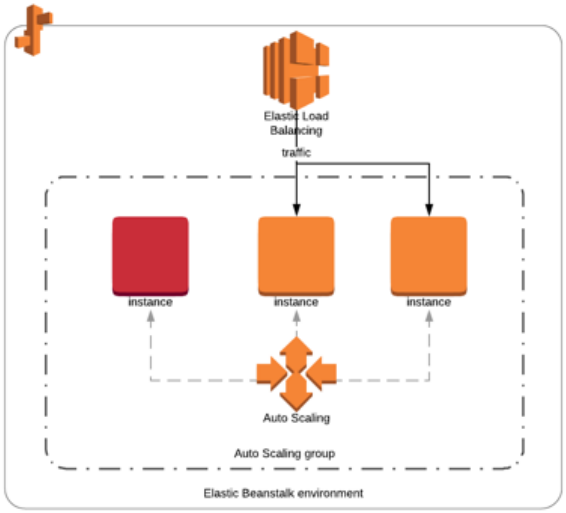
1. Deployment starts in the first batch.



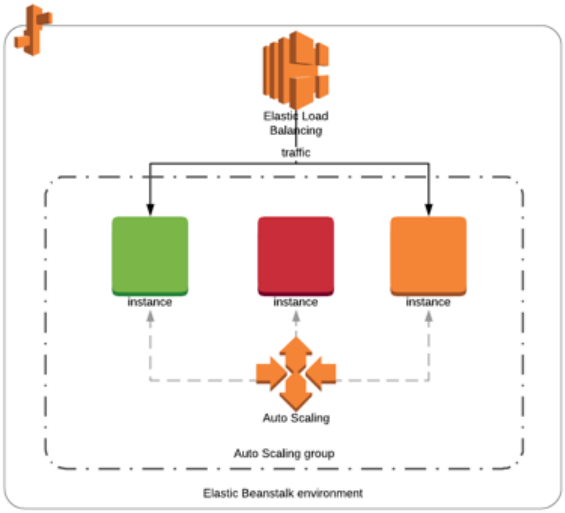
2. Deployment continues with the next batch.

What happens if the deployment fails? Only failed instance will be effected and the effect on whole environment depends on when it failed.

- If it failed in the first batch, the first instance will be down, and the remaining instances will continue to serve the previous version.
- If it failed in later batches, the failed instance will become unhealthy and there will be two versions served by your environment: The new version will be served by successful deployments until the failure and the previous version will be served by instances the deployment cancelled.



In case of the first batch failure, other instances continue to serve the previous version.



In case of the second batch failure, there are two versions served.

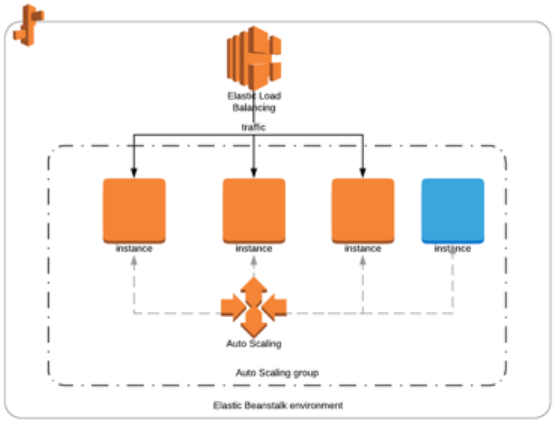
In addition, your fleet size serving the application will be reduced by the batch size, as failed instances will not be able to serve traffic. But, you will not have downtime when compared to `all at once` deployments.

To rollback, you need to cancel deployment using AWS Management Console or AWS CLI or EB CLI and start redeployment of the known previous healthy version on the environment.

Rolling with additional batch: Let’s launch new instances and deploy on them!

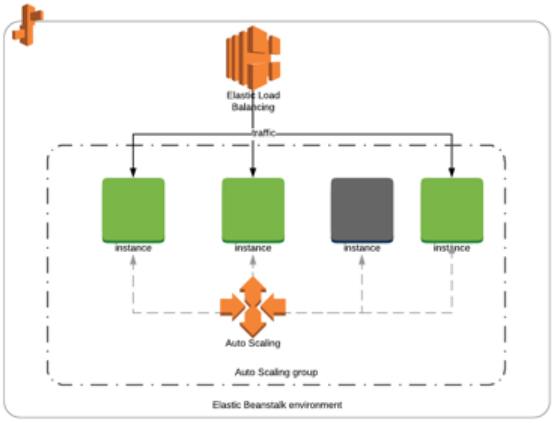
As we saw in the previous section, in `Rolling` deployments, the fleet size serving traffic is reduced during deployments and in case of failures. To solve this problem, you can use `Rolling with additional batch`.

In this deployment, Elastic Beanstalk provisions a new batch of instances and starts by deploying the latest version to them. Then, it will register the new instances to the load balancer and continues with the next batch of instances. After deployment is completed in all batches, it decreases the instances desired limit to the state where deployment started. In other words, it terminates the number of instances equal to the batch size it launched.



1. Deployment starts by launching new instances for the first batch.

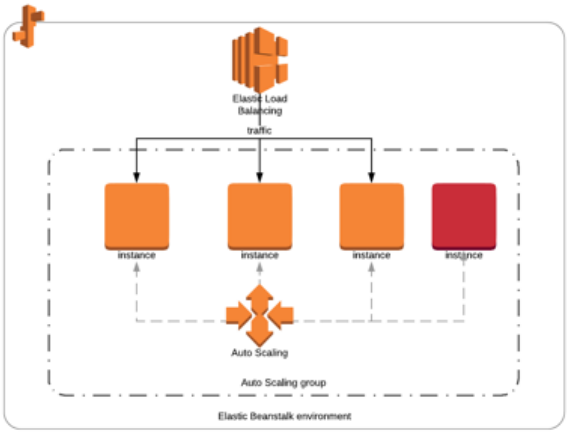
2. Deployment continues with the next batch.



3. After the final batch, excess instance is terminated.

What happens if this deployment fails? If the deployment fails in the first batch, it terminates the failed instances when you cancel the deployment. Because deployment was performed on the additional batch, nothing will be affected. Your fleet size will remain as it is, and you do not need to do anything. Awesome! The effect is minimal.

However, if the deployment fails after the first batch, the failed batch will be terminated. Then, two versions of your application will be served, like in the failures in `Rolling` deployments without additional instances. You need to deploy the previous version again to roll back.



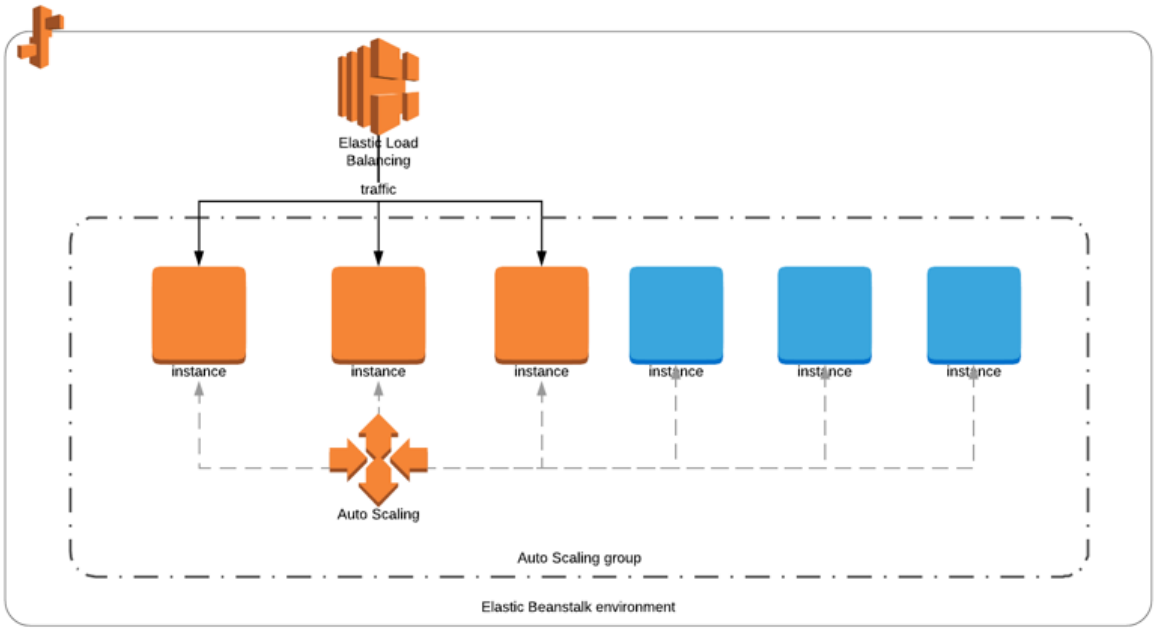
In case of the first batch failure, the new instance is terminated. The fleet size is not effected.

If the second batch fails, there will be two versions running as in rolling deployments without additional batch.

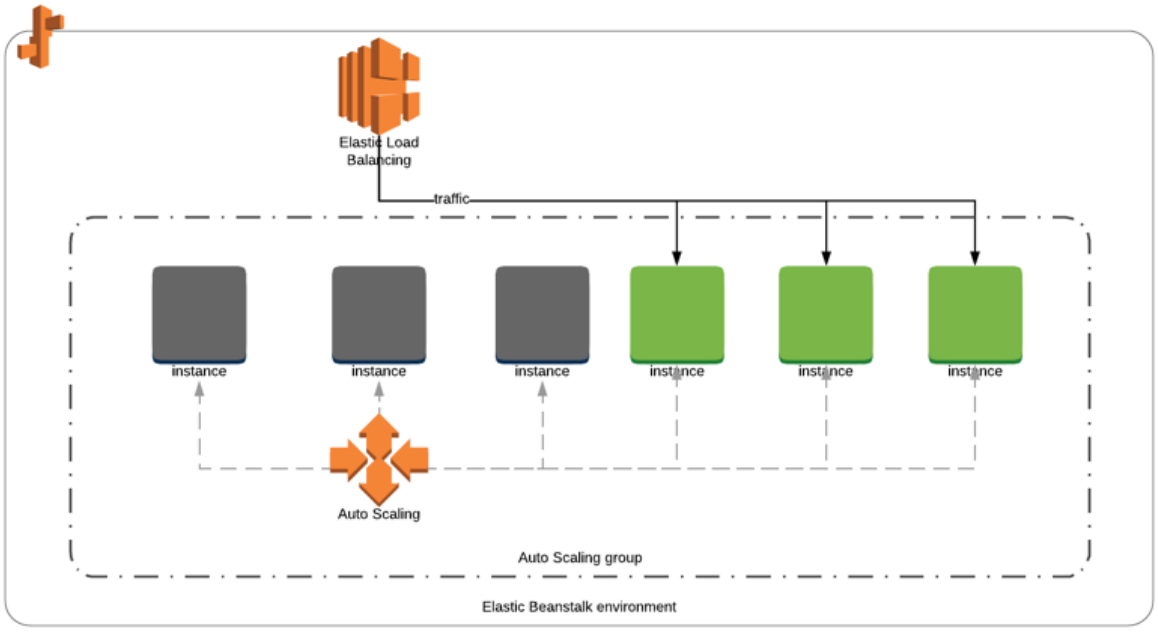
When I tried this deployment method, I found it safe when it failed in the first batch because failed instances were replaced automatically. But, I could not simulate failures after the first batch, but it is possible.

Immutable: Let's launch new instances and deploy on them!

This deployment method regards your instances as `immutable` as they are not updated but replaced. When you deploy using the `immutable` deployment type, Elastic Beanstalk will launch new instances and duplicate the size of your fleet for a short period. Then, it will deploy the latest version in new instances and terminate the old group after success.

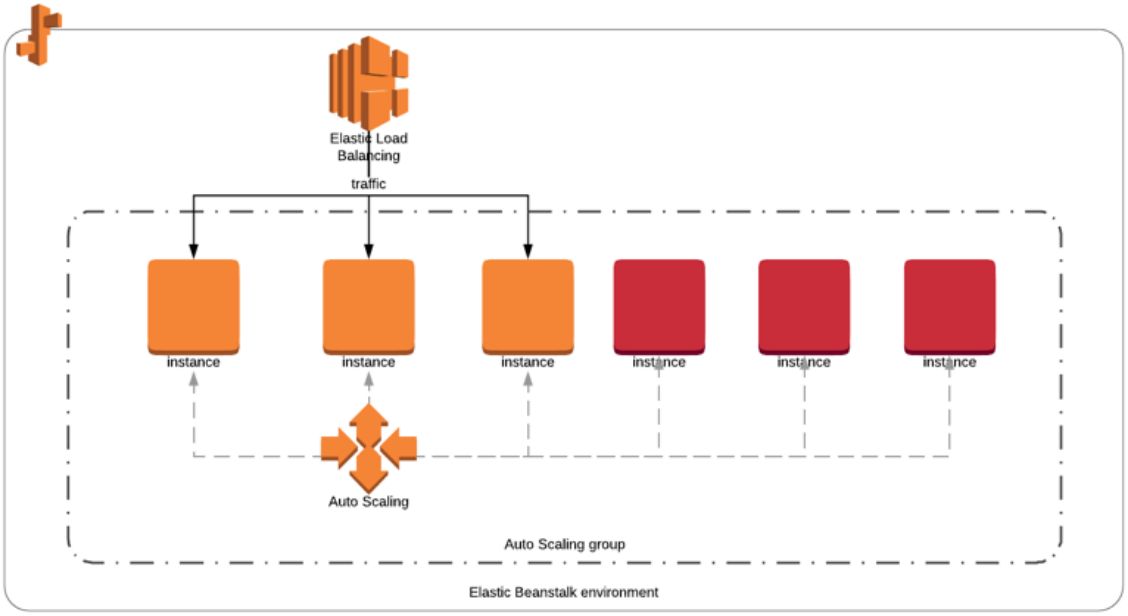


1. Deployment starts by duplicating the instances and deploying the app in new instances



2. After deployment succeeds, older instances are terminated.

As you might have guessed already, in case of a deployment failure, the new instances will be terminated without affecting your availability.



In case of failure, the new instances is terminated and traffic is not effected.

What are the drawbacks? When you duplicate the fleet, you are subject to AWS account limits, and Elastic Beanstalk may fail to launch new instances if your fleet size is too large. Also, it is slightly more costly than other deployment types as you update them in place or only launch a group of new instances during deployment. But on the cloud, this cost effect is minimal as it is expected to last only for a few minutes.

This deployment method is definitely safer than the previous methods.

Blue/Green: Duplicate the whole environment and switch the URLs

Blue/Green deployments simply replicate your current environment (blue), deploy the new application to your new, cloned environment (green), and redirect the traffic to the green one after deployment.

If the deployment fails, you terminate the green environment, and nothing will be affected.

If something goes wrong after deployment, for example, your users experience a problem in the new version, you can simply redirect the traffic back to the old version. Hence, it would be wise to keep the old environment running until you verify that the deployment is successful and DNS propagation completed.

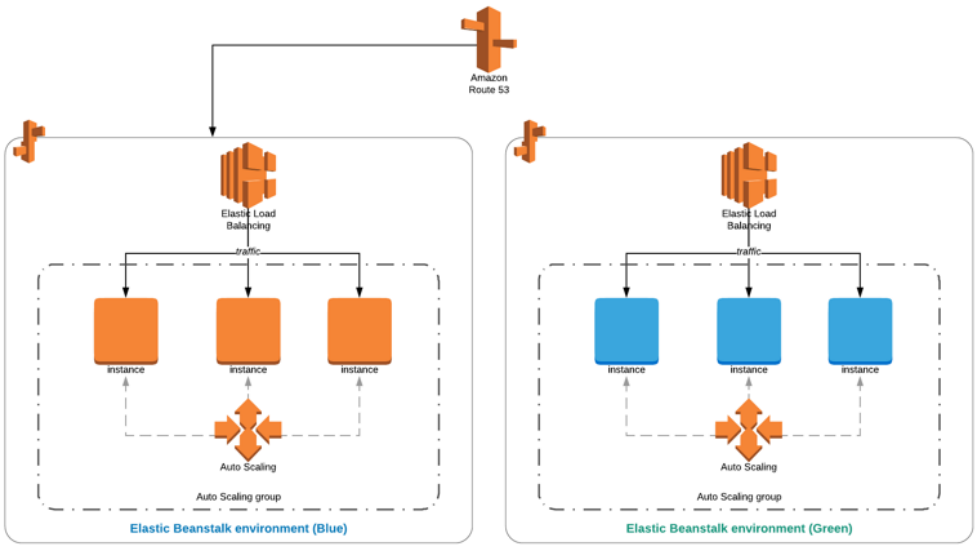
If everything goes well, you terminate the old environment, and your cloned environment becomes your new blue one. You repeat the same process in all new deployments. It is a nearly zero-downtime deployment model and a best practice for mission-critical applications.

How to perform Blue/Green deployments on AWS Elastic Beanstalk?

As you deploy application versions on a selected environment, Blue/Green does not exist as a deployment type on Elastic Beanstalk deployments because you need to duplicate the environment as a whole, including Elastic Load Balancers. However, this is very simple on Elastic Beanstalk, and actually, this simplicity is one of its strengths:

1) You clone the current environment using AWS Management Console or AWS CLI, or EB CLI. There is a feature for this. It will create a replica of your environment alongside Elastic Load Balancers, Autoscaling Groups, and other resources and deploy the current version on the new instances.

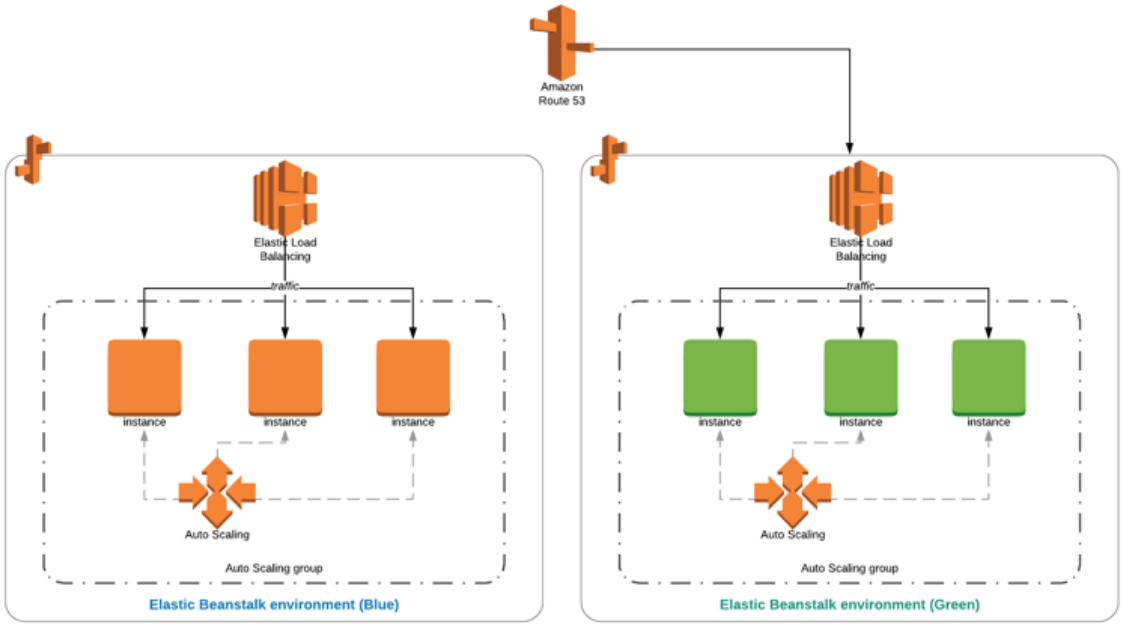
2) After your new environment is ready, you deploy the new version on this environment and verify that the deployment is successful.



Old environment continues to serve the traffic during deployments on the cloned environment. In case of failure, it will not be effected.

Deployments are run on the cloned environment

3) Once you are sure that everything is fine, you swap the URLs of the two environments using AWS Management Console or AWS CLI, or EB CLI. Again, Elastic Beanstalk provides a specific action for this. Then, the traffic will start to flow to your new environment after the DNS propagation completes.



What are the prerequisites to apply Blue/Green?

First of all, you are duplicating your environment and subject to AWS account limits as in the immutable deployments. Hence, your EC2 instance limits should cover the new instances launched.

Secondly and even more importantly, your environment should not contain any RDS instances managed by Elastic Beanstalk because the data will not be copied during the clone operation. Even it had clonned, the two databases would not be in sync during the swap phase. So, swapping URLs and pointing to an empty database would not be feasible if you have an RDS database resource in your Elastic Beanstalk environment.

Conclusion

As you see, Elastic Beanstalk is a powerful platform to deploy your applications without worrying about the infrastructure. You can apply DevOps principles using the Blue/Green deployment method and other deployment types. After reading this post, I hope you have a clear picture of the deployment types Elastic Beanstalk provides.