AWS Elastic Beanstalk — Choosing the right Deployment Option

Elastic BeanStalk helps to quickly deploy applications by automatically handling the infrastructure provisioning, scaling, load balancing and monitoring application health. This reduces a lot of overhead on the developers allowing them to focus on the business needs.

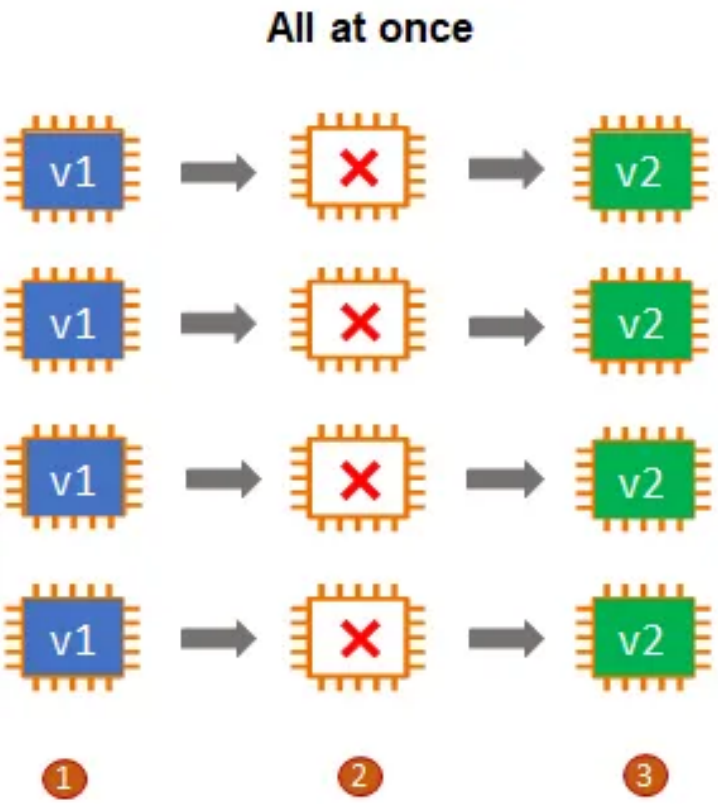
Given the above advantages offered by Elastic Beanstalk, choosing the optimal application deployment option while configuring Beanstalk is equally important. Deployment configurations determine availability and cost impacts when updating the application codebase.

In this article, we will look at the various deployment options available in Beanstalk along with pros and cons for each that will provide guidelines for choosing them when using for applications.

Elastic Beanstalk provides below deployment options for updates
1. All at once
2. Rolling
3. Rolling with additional batches
4. Immutable
5. Traffic Splitting
6. Blue Green

Let us look at each of them in detail.

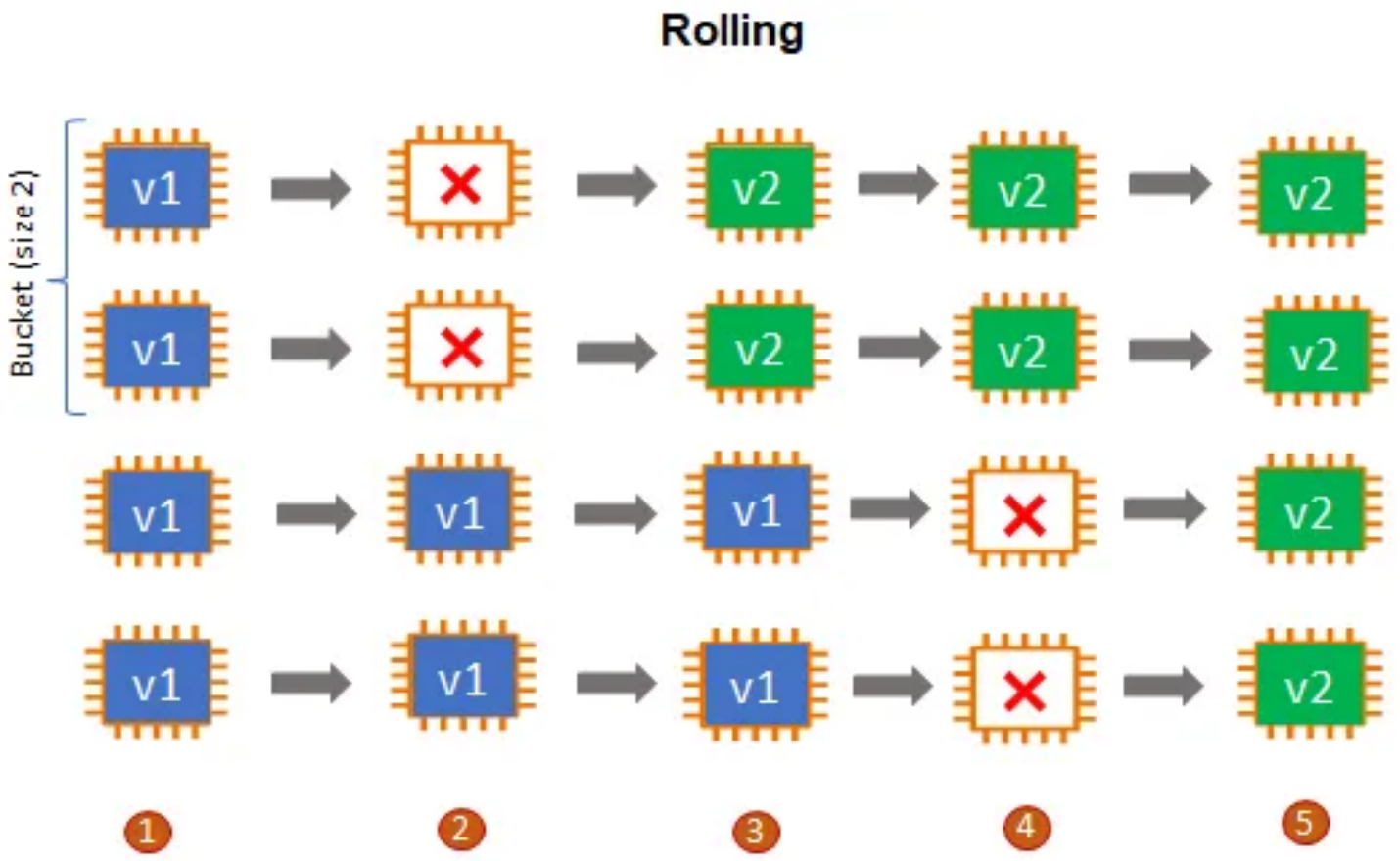
1. **All at once :**
   In this type, all the instances are stopped at once and the new version is deployed to all of them.



The above figure depicts the various stages:
1. All instances running old version v1
2. All instances are stopped, which means there is no application running
3. Version v2 is deployed on all instances. All instances are running new version v2.

Pros:
- Fastest Deployment
- Quick deployments, suitable if we can accept unavailability of environment for small time, can be used in dev environment
- No additional cost
Cons:
- Application is completely down until the deployments are done

2. Rolling :
This method involves deploying the new version in batches. Each batch is taken out of service and replaced, reducing environment's capacity by the number of instances in a batch.

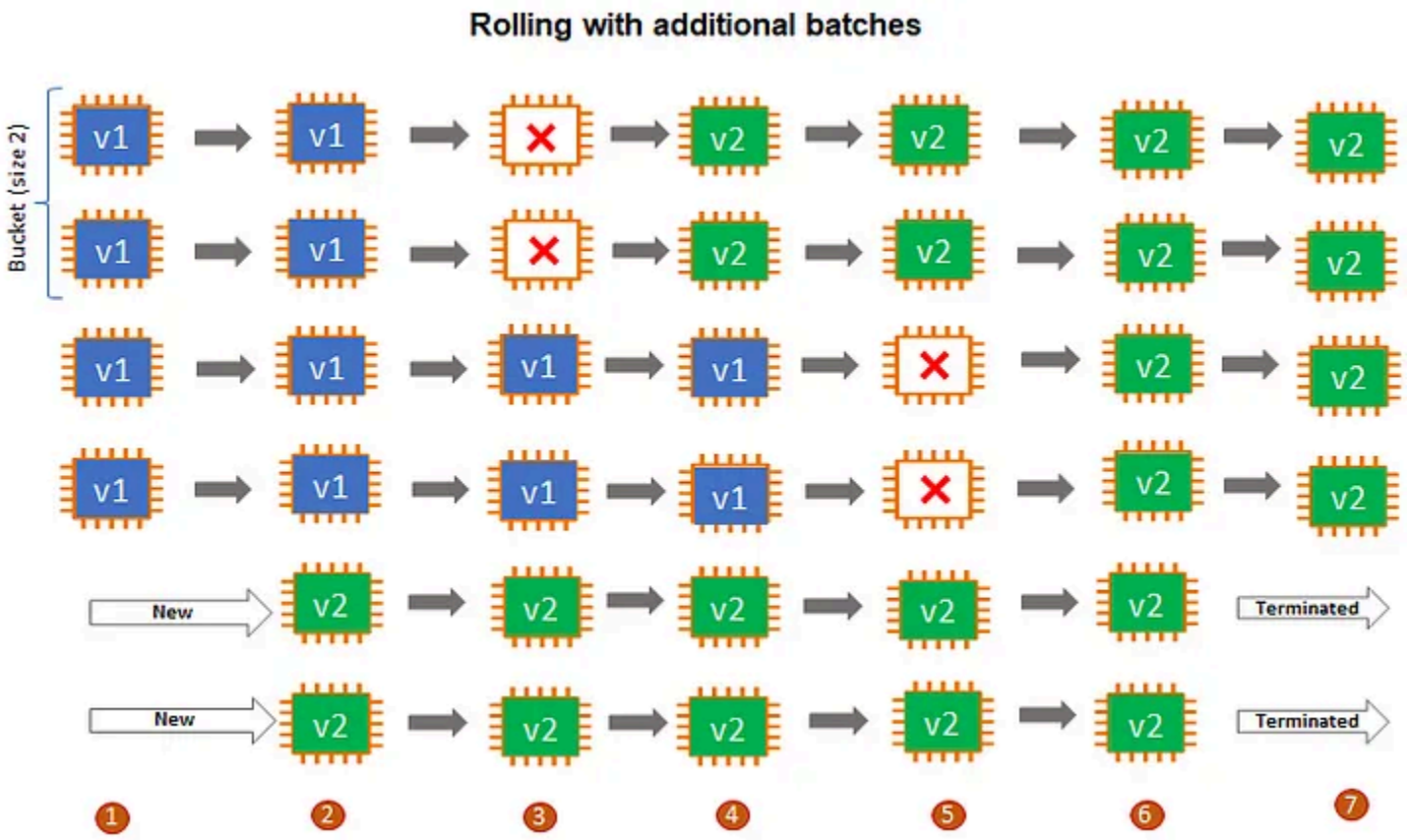


The above figure depicts the various stages:
1. All instances are running old version v1.
2. The rolling bucket size is selected as 2. The top 2 instances are stopped and move into stopped state.
3. Version v2 is deployed on the 2 stopped instances. The remaining instances are running on v1. Here we have both versions running in parallel.
4. The next 2 instances which were running v1 are stopped. At this step, we have only v2 running with decreased capacity.
5. Version v2 is deployed on the 2 stopped instances. All instances are now running v2.

Pros:
- No additional cost
- Incremental updates based on bucket size
- Running 2 versions simultaneously can facilitate testing and rollback of buckets in case of failure

Cons:
- Application is running below capacity
- Longer time for deployment

3. Rolling with additional batches :
This option involves deploying the new version in batches, however by first launching a new batch of instances to ensure full capacity during the deployment process.



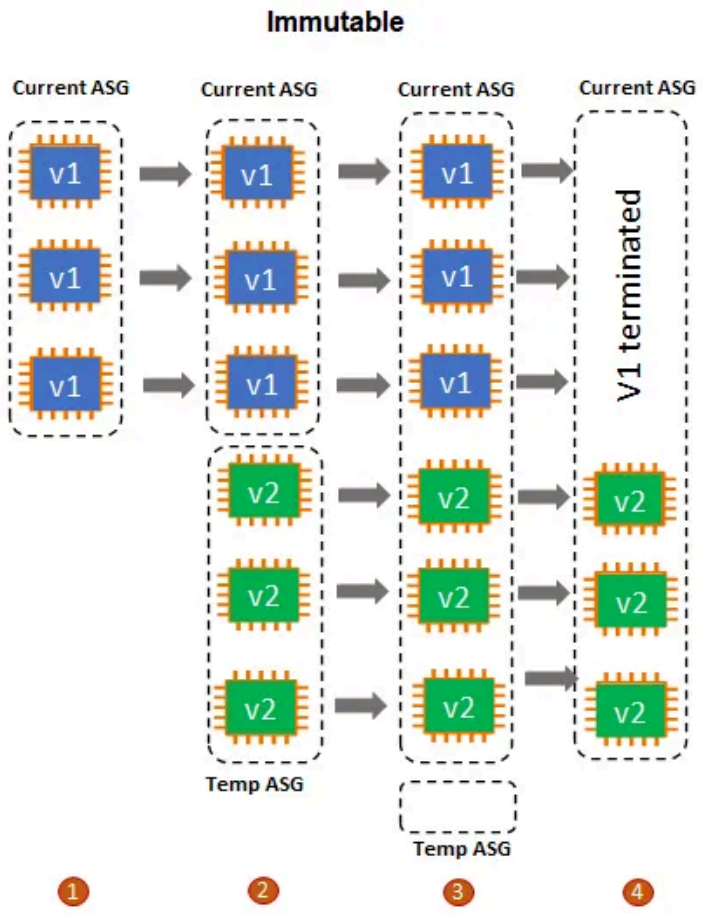
The above figure depicts the various stages:
1. All instances are running old version v1.
2. The rolling batch size is selected as 2. Hence 2 new instances are created and version v2 is deployed into the new instances.
3. 2 instances running the old version v1 are stopped. We still have the 4 instances running so the application is running at full capacity however with 2 different versions.
4. version v2 is deployed on the 2 instances stopped from the existing stack.
5. The remaining 2 instances or next batch is selected from existing instances running v1 is selected and stopped
6. These instances are deployed with v2 version. So now we have all existing instances replaced with v2 version.

7. The new instances added in the beginning are now terminated since we already have all the 4 existing instances replaced with V2 version.

Pros:
- Application is running at full capacity
- Incremental updates based on bucket size. Good option for production workloads.
- Running 2 versions simultaneously can facilitate testing and rollback of buckets in case of failure

Cons:
- Small additional cost due to addition of new instances
- Longer deployment

4. **Immutable :**
This option involves deploying the new version to a fresh group of instances and then terminating the old instances.

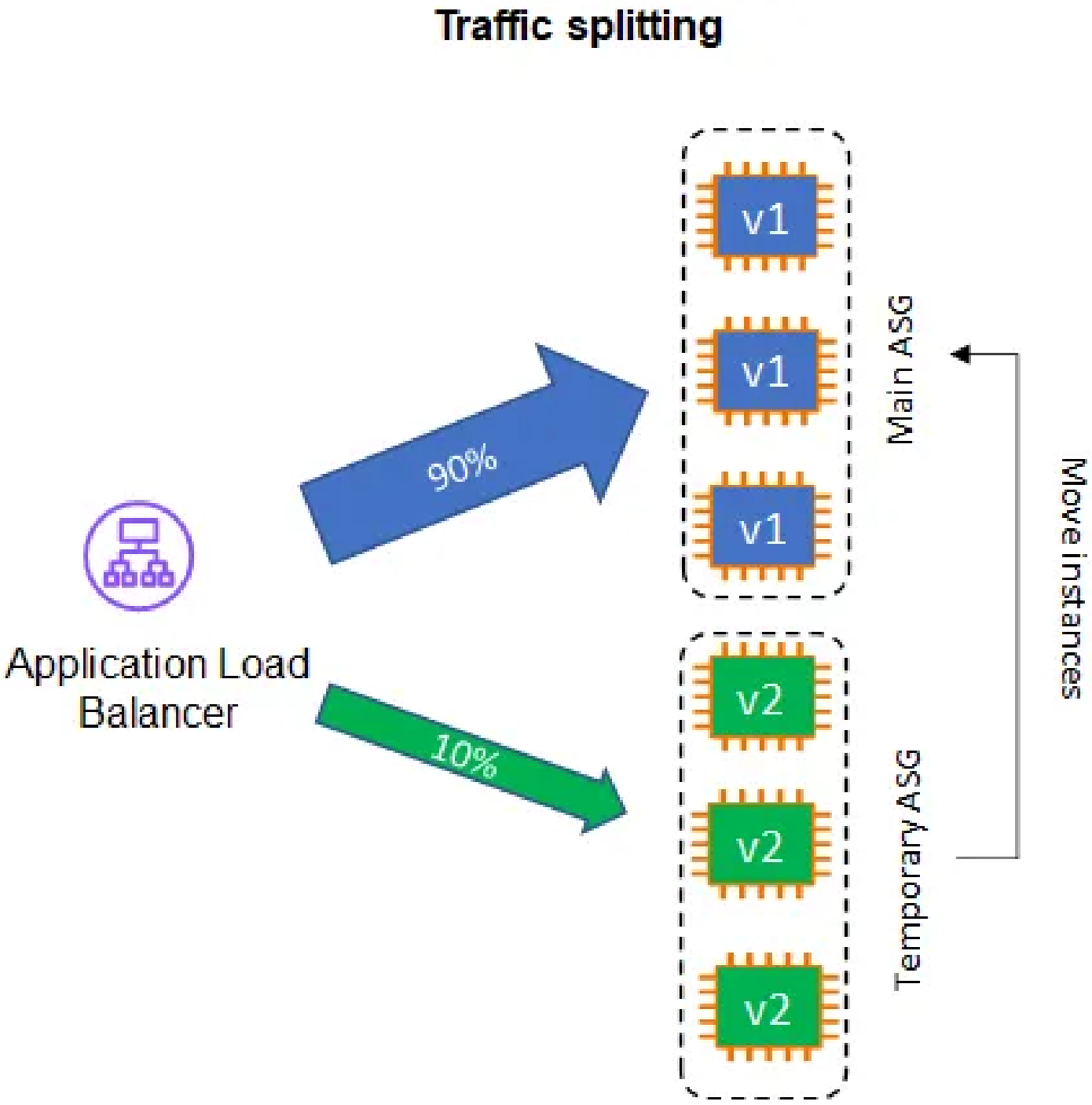


The above figure depicts the various stages:
1. All instances are running old version v1.
2. Create a new temporary ASG (AutoScaling Group), with new instances of equal capacity as old. Deploy version v2 on the new instances.
3. Test the new instances and once they are working fine, move them into current (original) ASG.
4. Terminate the instances that are running version v1. Now we will have version v2 running on the same number of instances as original capacity.

Pros:
- Zero downtime , suitable for production workloads
- Quick rollback in case of failures, terminate the Temp ASG on failure.

Cons:
- Highest cost, since it requires double capacity
- Longest time for deployment

5. Traffic splitting
This option involves deploying the new version to a fresh group of instances and temporarily splitting incoming client traffic between the existing application version and the new one.



The above figure depicts the deployment:

1. Create a new ASG. New application version v2 is deployed to instances in a temporary ASG.
2. A small percentage (such as 10%) of traffic is sent to the temporary ASG for a configurable amount of time.
3. The heath of deployment is monitored for temporary ASG.
4. If the deployment fails, automatic rollback is triggered quickly.
5. If everything looks good with the new version v2 deployment, instances from temporary ASG are moved to existing/original ASG.
6. Instances running old version v1 are terminated.

Pros:
- Allows for canary testing
- Rollback is automatic and quick on failure, can be used for production

Cons:
- Requires provisioning of entirely new environment
- Increased cost due to duplicate environment