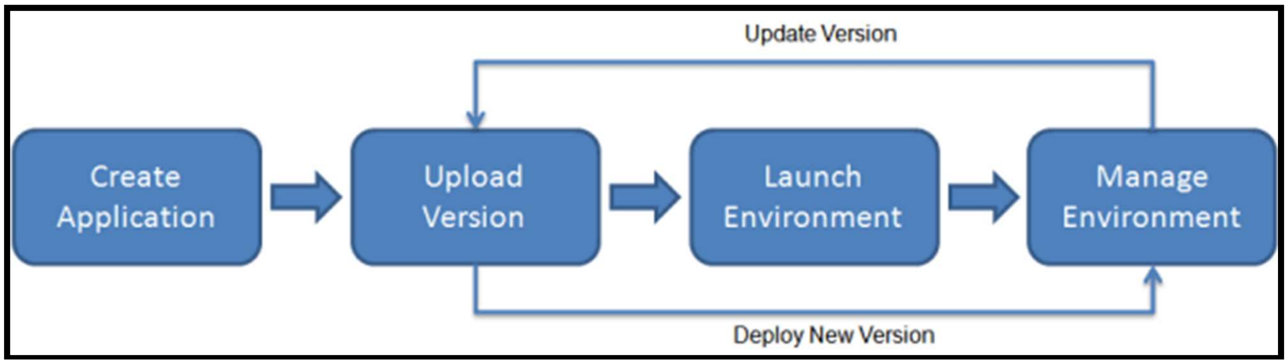
Elastic Beanstalk

- AWS Elastic Beanstalk is a compute service that makes it easier for the developers to quickly deploy and manage applications that we upload to the AWS cloud.
- Developers simply upload their application to the AWS cloud, and then let the AWS Beanstalk provision and handle the configuration for us.
- It automatically handles deployment details like capacity provisioning, load balancing, auto scaling, and application health monitoring.
- With Elastic Beanstalk, you can quickly deploy and manage applications in the AWS Cloud without having to learn about the infrastructure that runs those applications.
- You simply upload your application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.
- Elastic Beanstalk supports applications developed in Go, Java, .NET, Node.js, PHP, Python, and Ruby.
- When you deploy your application, Elastic Beanstalk builds the selected supported platform version and provisions one or more AWS resources, such as Amazon EC2 instances, to run your application.
- To use Elastic Beanstalk, you create an application, upload an application version in the form of an application source bundle (for example, a Java .war file) to Elastic Beanstalk, and then provide some information about the application.
- Elastic Beanstalk automatically launches an environment and creates and configures the AWS resources needed to run your code.
- After your environment is launched, you can then manage your environment and deploy new application versions.
- The following diagram illustrates the workflow of Elastic Beanstalk.



- Before using Amazon elastic beanstalk service, we have to create a local application of any platform. It can be Python, PHP, Node.js, etc.
- After that, we have to create an application in Elastic Beanstalk with an environment where we can upload our local application. Then we deploy it and use the URL provided for it to launch it.

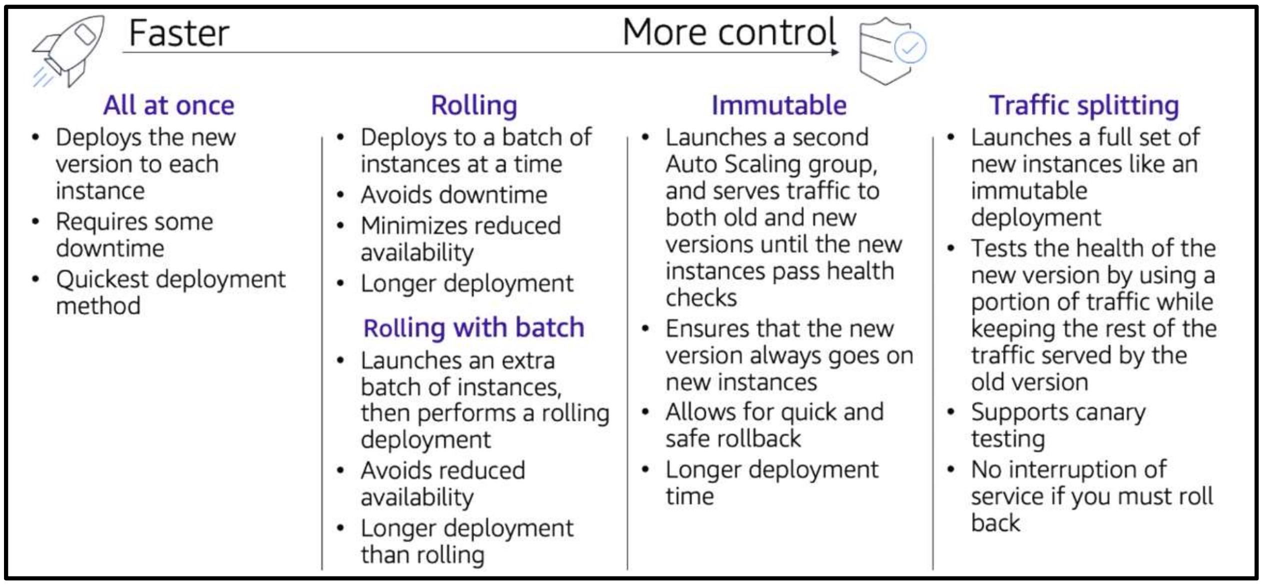
Components of Elastic Beanstalk

- **Application:** An Elastic Beanstalk application is logically considered equivalent to a folder/container containing the source code. In Amazon Elastic Beanstalk, we upload our application as a zip file with all the contents in it.
- **Application Version:** Application Version refers to the web application which we have uploaded and will upload its next upgraded version. For example, we upload our application to AWS Beanstalk at first and then we have updated the source code of our application. Instead of overwriting our previous version of the application, we can give it a new version name which we could use to compare them both.
- **Environment**: The collection of AWS resources is an environment and an environment can only run one application version at a time. We may run multiple applications in multiple environments at the same time. Whenever an environment is created, Amazon Elastic Beanstalk automatically provisions required EC2 instances and S3 buckets.
- **Environment Tier**: Basically, there are two environment tiers available and they are Web Server Environment and Worker Environment. An application using PHP or requires HTTP requests runs in a Web Server Environment. An application using Amazon Simple Queue Service (SQS) runs in a Worker Environment.
- **Environment Configuration**: The configuration of an environment is a set of parameters like security group, Instance type, and platform version. If we change the configuration, Amazon Elastic Beanstalk implements it dynamically. It either applies the new changes or deletes and deploys new resources.
- **Saved Configuration:** We can create a template that contains the basic functionalities and use it as a starting point for our environment configurations. We can also modify the configurations whenever we need them and use them while creating a new environment.
- **Platform**: A platform is a combination of all the AWS Beanstalk components, an Operating system, a programming language runtime, and a webserver to run the applications. We can choose our platform while creating our application or environment. We don't need to update it or include software patches, AWS takes care of that. Just try to make your application as good as possible.

Elastic Beanstalk Deployment Modes

- Elastic Beanstalk provides several options for processing deployments, including deployment policies (all at once, rolling, rolling with additional batch, immutable, and traffic splitting).
- It also includes options that you can use to configure batch size and health check behavior during deployments.

- Choose the method that makes the most sense for your use case.
- Consider how fast you can make updates compared to how much tolerance your users have for issues with a new version and downtime during updates.
- All at once deployments are the fastest but require at least some downtime.
- Immutable deployments are the safest but require a longer deployment time.
- With traffic splitting, you can test a small amount of traffic to the new version while still sending most traffic to the existing version.
- Then, you can automatically shift all traffic to the new version after a period if health checks are successful.
- By default, your environment uses all at once deployments. If you created the environment with the CLI and it's a scalable environment (you didn't specify the -- single option), it uses rolling deployments.


Faster
More control
All at once
Deploys the new version to each instance
Requires some downtime
Quickest deployment method
Rolling
Deploys to a batch of instances at a time
Avoids downtime
Minimizes reduced availability
Longer deployment
Rolling with batch
Launches an extra batch of instances, then performs a rolling deployment
Avoids reduced availability
Longer deployment than rolling
Immutable
Launches a second Auto Scaling group, and serves traffic to both old and new versions until the new instances pass health checks
Ensures that the new version always goes on new instances
Allows for quick and safe rollback
Longer deployment time
Traffic splitting
Launches a full set of new instances like an immutable deployment
Tests the health of the new version by using a portion of traffic while keeping the rest of the traffic served by the old version
Supports canary testing
No interruption of service if you must roll back

- **By default, our environment uses all-at-once deployments.** The quickest deployment method. Suitable if you can accept a short loss of service, and if quick deployments are important to you. With this method, Elastic Beanstalk deploys the new application version to each instance. As a result, your application might be unavailable to users (or have low availability) for a short time.
- **With rolling deployments**, avoids downtime and minimizes reduced availability. Elastic Beanstalk splits the environment's Amazon EC2 instances into batches and deploys the new version of the application to one batch at a time. It leaves the rest of the instances in the environment running the old version of the application. During a rolling deployment, some instances serve requests with the old version of the application, while instances in completed batches serve other requests with the new version.
- **Rolling deployments with additional batch:** To maintain full capacity during deployments, we can configure our environment to launch a new batch of instances before taking any instances out of service. This option is known as a rolling deployment with an additional batch. With this method, Elastic Beanstalk launches an extra batch of instances, then performs a rolling deployment. Launching the extra batch takes time.

Avoids any reduced availability. When the deployment completes, Elastic Beanstalk terminates the additional batch of instances.

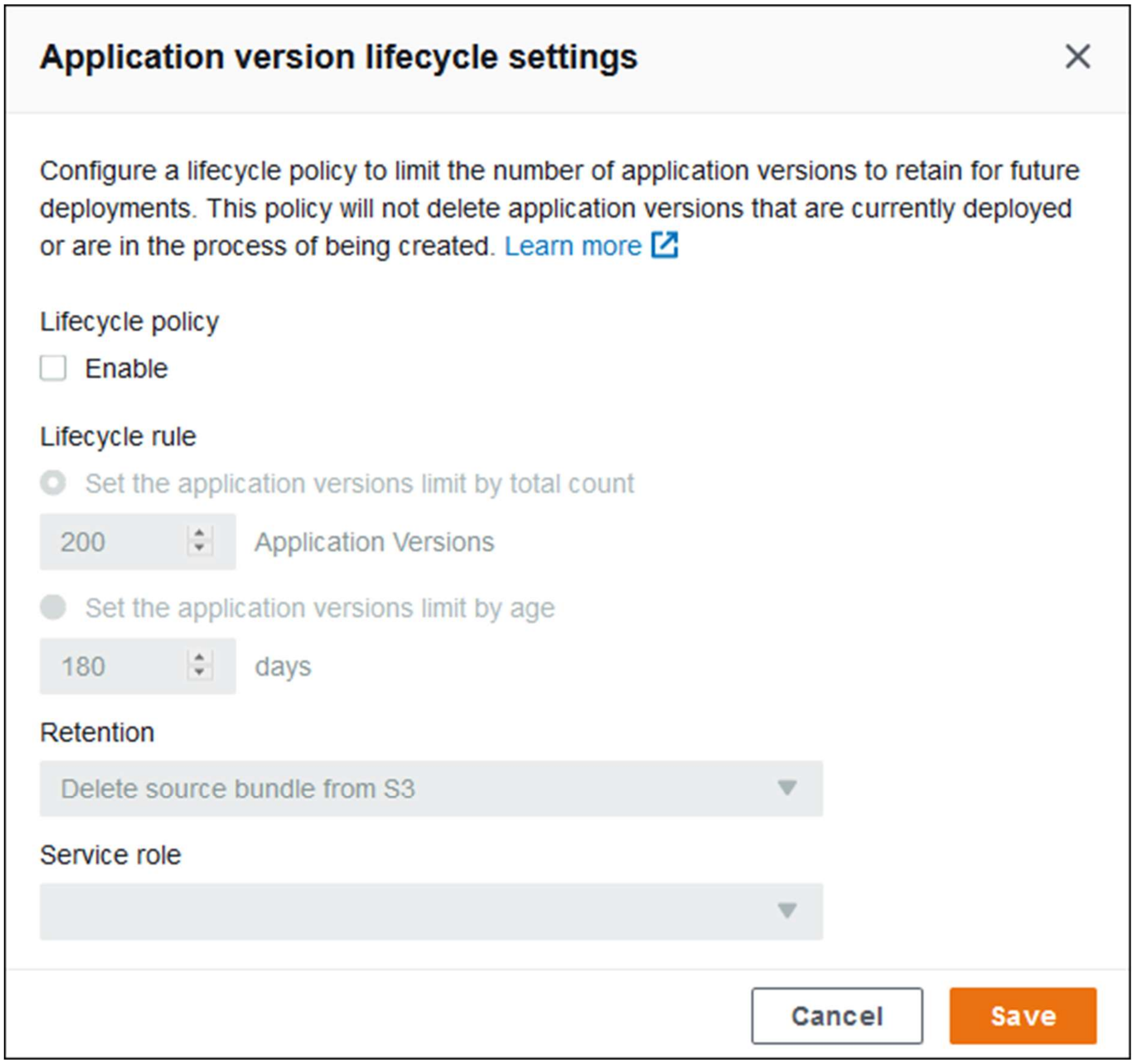
- **Immutable deployments:** A slower deployment method, that ensures your new application version is always deployed to new instances, instead of updating existing instances. It also has the additional advantage of a quick and safe rollback in case the deployment fails.
- **Traffic-splitting deployments** let we perform canary testing as part of our application deployment. In a traffic-splitting deployment, Elastic Beanstalk launches a full set of new instances just like during an immutable deployment. It then forwards a specified percentage of incoming client traffic to the new application version for a specified evaluation period. If the new instances stay healthy, Elastic Beanstalk forwards all traffic to them and terminates the old ones. If the new instances do not pass health checks, or if we choose to abort the deployment, Elastic Beanstalk moves traffic back to the old instances and terminates the new ones. There is never any service interruption. Suitable if you want to test the health of your new application version using a portion of incoming traffic, while keeping the rest of the traffic served by the old application version.

Beanstalk Lifecycle Policy

- Each time you upload a new version of your application with the Elastic Beanstalk console or the EB CLI, Elastic Beanstalk creates an application version.
- If you don't delete versions that you no longer use, you will eventually reach the application version quota and be unable to create new versions of that application.
- You can avoid hitting the quota by applying an *application version lifecycle policy* to your applications.
- A lifecycle policy tells Elastic Beanstalk to delete application versions that are old, or to delete application versions when the total number of versions for an application exceeds a specified number.
- Elastic Beanstalk applies an application's lifecycle policy each time you create a new application version, and deletes up to 100 versions each time the lifecycle policy is applied.
- Elastic Beanstalk deletes old versions after creating the new version, and does not count the new version towards the maximum number of versions defined in the policy.
- Elastic Beanstalk does not delete application versions that are currently being used by an environment, or application versions deployed to environments that were terminated less than ten weeks before the policy was triggered.
- The application version quota applies across all applications in a region.
- If you have several applications, configure each application with a lifecycle policy appropriate to avoid reaching the quota.

To specify your application lifecycle settings

1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.
2. In the navigation pane, choose **Applications**, and then choose your application's name from the list.
3. In the navigation pane, find your application's name and choose **Application versions**.
4. Choose **Settings**.
5. Use the on-screen form to configure application lifecycle settings.
6. Choose **Save**.


Application version lifecycle settings

Configure a lifecycle policy to limit the number of application versions to retain for future deployments. This policy will not delete application versions that are currently deployed or are in the process of being created. Learn more

Lifecycle policy

☐ Enable

Lifecycle rule

○ Set the application versions limit by total count

200 Application Versions

○ Set the application versions limit by age

180 days

Retention

Delete source bundle from S3

Service role

Beanstalk Extensions

- You can add AWS Elastic Beanstalk configuration files (.ebextensions) to your web application's source code to configure your environment and customize the AWS resources that it contains.
- Configuration files are YAML- or JSON-formatted documents with a .config file extension that you place in a folder named .ebextensions and deploy in your application source bundle.
- **Example .ebextensions/network-load-balancer.config**

  This example makes a simple configuration change. It modifies a configuration option to set the type of your environment's load balancer to Network Load Balancer.

```
option_settings:
 aws:elasticbeanstalk:environment:
  LoadBalancerType: network
```

- The option_settings section of a configuration file defines values for configuration options. Configuration options let you configure your Elastic Beanstalk environment, the AWS resources in it, and the software that runs your application.
- Configuration files are only one of several ways to set configuration options.
- The Resources section lets you further customize the resources in your application's environment, and define additional AWS resources beyond the functionality provided by configuration options.
- You can add and configure any resources supported by AWS CloudFormation, which Elastic Beanstalk uses to create environments.
- The other sections of a configuration file (packages, sources, files, users, groups, commands, container_commands, and services) let you configure the EC2 instances that are launched in your environment.
- Whenever a server is launched in your environment, Elastic Beanstalk runs the operations defined in these sections to prepare the operating system and storage system for your application.

Beanstalk with Docker

- Elastic Beanstalk supports the deployment of web applications from Docker containers.

- With Docker containers, you can define your own runtime environment. You can also choose your own platform, programming language, and any application dependencies, such as package managers or tools, which typically aren't supported by other platforms.
- Docker containers are self contained and include all the configuration information and software that your web application requires to run.
- All environment variables that are defined in the Elastic Beanstalk console are passed to the containers.
- By using Docker with Elastic Beanstalk, you have an infrastructure that handles all the details of capacity provisioning, load balancing, scaling, and application health monitoring.
- You can easily manage your web application in an environment that supports the range of services that are integrated with Elastic Beanstalk.

Beanstalk Cleanup

- To ensure that you're not charged for any services you aren't using, delete all application versions and terminate the environment.
- This also deletes the AWS resources that the environment created for you.
- **To delete the application and all associated resources**

    1. Delete all application versions.
    2. Terminate the environment.
    3. Delete the getting-started-app application.