Elastic Beanstalk

Let us be clear with one point before we learn & jump into Elastic Beanstalk

AWS categorizes Elastic Beanstalk as a **compute service** because its primary role is to manage compute resources for deploying and running applications. However, it is also recognized as a **PaaS** because of its developer-friendly features and ability to abstract infrastructure management, simplifying the development and deployment process. ***Both interpretations are accurate, but they focus on different facets of the service.***

Elastic Beanstalk – **Compute Service?** or **Development Kit?**

Both perspectives are correct, but they address different aspects of what Elastic Beanstalk provides:

- **As a Compute Service**: This categorization is correct because Elastic Beanstalk fundamentally manages compute resources such as EC2 instances, which are the core elements that actually execute the application code.
- **As a Development Platform (PaaS)**: This perspective is also valid as Elastic Beanstalk provides an easy-to-use platform for deploying, managing, and scaling applications, focusing on the development workflow.

Elastic Beanstalk

Intro

AWS Elastic Beanstalk is a fully managed service that makes it easy for developers to deploy, manage, and scale web applications and services. It supports several programming languages, including Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker, allowing you to upload your code, and Elastic Beanstalk automatically handles the deployment, from capacity provisioning, load balancing, and auto-scaling to application health monitoring.

Key Features of AWS Elastic Beanstalk:

1. **Ease of Use**: Elastic Beanstalk abstracts much of the underlying infrastructure management, allowing developers to focus on writing code rather than managing servers or configuring load balancers.
2. **Support for Multiple Languages and Platforms**: It supports various development environments, including popular languages and platforms.
3. **Automated Scaling**: Elastic Beanstalk automatically scales your application up and down based on its specific needs using auto-scaling rules.
4. **Environment Control**: While Elastic Beanstalk manages the environment, you still have full control over the AWS resources powering your application, including the ability to customize or configure resources like Amazon EC2 instances.
5. **Integrated Monitoring**: The service integrates with Amazon CloudWatch, allowing you to monitor application health, performance, and resource usage.
6. **Consistent Environment Management**: You can deploy your application across different environments, ensuring consistent application settings.

Elastic Beanstalk

Why Use AWS Elastic Beanstalk?

1. **Simplified Deployment and Management**: It simplifies the process of deploying applications by managing the infrastructure and platform stack (OS, application server, and runtime).
2. **Cost Efficiency**: You only pay for the underlying AWS resources you use. Elastic Beanstalk itself is free, meaning there are no additional management fees.
3. **Scalability**: Elastic Beanstalk can automatically scale applications based on demand, which helps maintain performance during traffic spikes and reduce costs when demand decreases.
4. **Customization and Control**: Even though Elastic Beanstalk manages many aspects of deployment, you can still customize the environment and control the AWS resources.
5. **Quick Prototyping and Development**: It is ideal for quickly deploying applications without spending a lot of time configuring and managing the environment. This feature is great for rapid prototyping and iterative development.
6. **Integrated Environment Management**: With Elastic Beanstalk, you can manage environments (development, staging, production) more consistently and efficiently, ensuring smooth transitions between different stages of development.

Elastic Beanstalk

Basic Terms we need to know

1. Application

An **application** in Elastic Beanstalk is a logical container for components such as environments, versions, and configurations. It serves as a higher-level entity that groups together multiple environments (e.g., development, testing, and production) under a single umbrella.

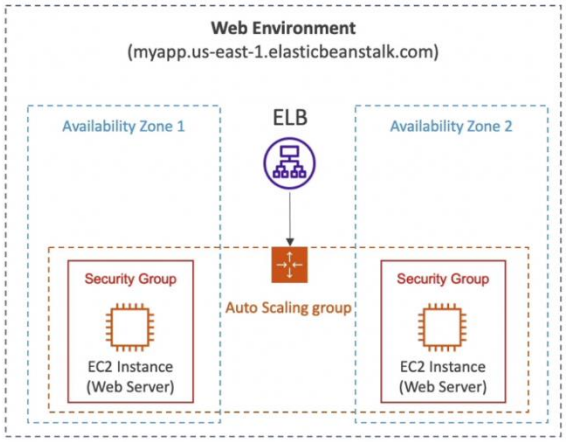
2. Environment

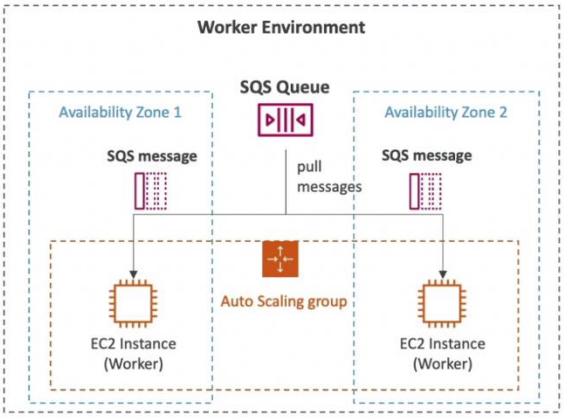
An **environment** is a collection of AWS resources created and managed by Elastic Beanstalk. Each environment runs a specific version of an application, which can include resources like EC2 instances, load balancers, and Auto Scaling groups. Environments are isolated from each other, meaning you can have different environments for development, staging, and production, each with its own configuration and resources.

3. Environment Tier

An **environment tier** determines the purpose of an environment, such as:

- **Web Server Environment Tier**: Used for web applications that handle HTTP requests. Elastic Beanstalk manages resources like EC2 instances, load balancers, and scaling for the application.
- **Worker Environment Tier**: Used for background tasks that do not handle HTTP requests directly. Elastic Beanstalk provisions an SQS queue and worker instances to process jobs from the queue.





Elastic Beanstalk

4. Environment Configuration

Environment configuration refers to the settings that control the resources and behaviour of an environment. This includes instance types, scaling settings, software configurations, environment variables, and other settings that define how the environment operates.

5. Application Version

An **application version** is a specific, labelled iteration of deployable code for an Elastic Beanstalk application. Each version is unique and can be deployed to one or more environments. Application versions are stored in Amazon S3 as application source bundles (ZIP or WAR files).

6. Platform

A **platform** is the combination of an operating system, programming language runtime, web server, application server, and Elastic Beanstalk components that Elastic Beanstalk uses to run applications. For example, there are platforms for Java, Python, Node.js, Docker, and many more. The platform provides the necessary runtime environment for the application to run.

7. Platform Version

A **platform version** represents a specific version of a platform that Elastic Beanstalk supports. AWS regularly updates platform versions to include new runtime versions, patch updates, and improvements.

8. Configuration Template

A **configuration template** is a starting point for creating environments with predefined settings. It includes a combination of environment configurations and can be reused for creating new environments with the same setup.

Elastic Beanstalk

9. Deployment Policy

A **deployment policy** determines how Elastic Beanstalk updates the environment with new application versions. Some common deployment policies are:

- **All at Once**: Deploys the new version to all instances at once, which causes downtime during deployment.
- **Rolling**: Updates instances in batches, which maintains availability but has reduced capacity during deployment.
- **Rolling with Additional Batch**: Adds a new batch of instances to handle traffic while others are being updated, ensuring no capacity reduction.
- **Immutable**: Deploys the new version to a new set of instances and swaps them with the old instances once the deployment is successful, ensuring minimal downtime and safe rollback.

10. Health Monitoring

Health monitoring is the process by which Elastic Beanstalk tracks the status and performance of the environment's resources. It uses health checks (like pinging EC2 instances) and metrics from Amazon CloudWatch to determine the overall health status, which can be one of several states: OK, Warning, Degraded, Severe, or Info.

11. Managed Updates

Managed updates are an Elastic Beanstalk feature that automatically applies platform updates to an environment on a scheduled basis. This ensures the environment uses the latest patches and improvements without manual intervention, maintaining security and performance.

12. Elastic Beanstalk CLI (EB CLI)

The **EB CLI** is a command-line interface specifically for Elastic Beanstalk. It allows developers to interact with Elastic Beanstalk from the terminal, manage environments, deploy applications, and perform other tasks.

13. Saved Configurations

Saved configurations are snapshots of environment settings that you can reuse. When you save a configuration, Elastic Beanstalk stores a copy of your environment's current settings, allowing you to apply the same configuration to new or existing environments later.

Elastic Beanstalk

14. Scaling

Scaling refers to adjusting the number of Amazon EC2 instances in an environment to handle the load. Elastic Beanstalk supports **Auto Scaling**, which automatically adjusts the number of instances based on metrics like CPU utilization or request count.

15. Extensions (.ebextensions)

.ebextensions are configuration files (written in YAML or JSON) that allow you to customize the environment beyond what is provided by the Elastic Beanstalk console or API. They can be used to install packages, run commands, configure resources, and more during environment creation or updates.

Now we will look into few topics in deep to understand – how it works

Elastic Beanstalk

Lifecycle Policy

- Lifecycle policy to phase out old versions based on:
    - **Days** (delete versions older than x days)
    - **Count** (keep latest x versions)
- Up to 1000 application version deployed simultaneously
- Option to retain the old version's source bundle in S3

EB Extensions

- Configure EB environment through config files
- Config files should be present in .ebextensions/ directory in the root of the source code
- Config files should have .config extension and should be in YAML or JSON format
- Lifecycle of resources managed by EB Extensions are tied to the EB environment
- Through EB Extensions, we have the ability to add resources (RDS, ElastiCache, etc.) through CloudFormation, which cannot be done from the EB console.

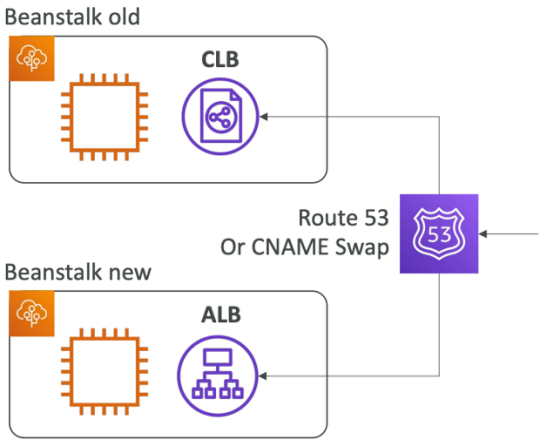
EB Cloning

- Clone an **environment** with the **same configuration**
- Useful to deploy a test version of your application
- Data present in DB will not be cloned, only the configuration will be cloned
- After cloning an environment, we can change the configuration

Elastic Beanstalk

EB Migration

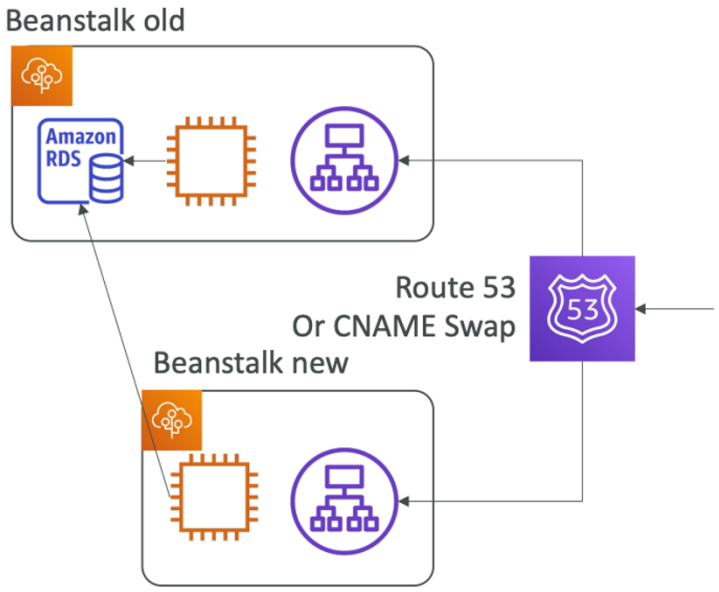
- **If some change cannot be done in an environment** (eg. changing the load balancer type)**, we need to migrate our environment.**
- Steps for migration
    - Create a new application environment with the same configuration except the required change (cannot clone)
    - Perform a CNAME swap or Route 53 update to route all traffic to the new environment



Elastic Beanstalk

Decoupling DB from EB Environment

- **Production environments should not have DB as a part of the environment as its lifecycle gets tied to the environment.**
- Steps to decouple an RDS DB already present in an EB environment:
    - Create a snapshot of the DB (for safety)
    - Go to RDS console → Protect RDS from deletion
    - Create a new EB environment without RDS and connect the application to the existing RDS (using connection string passed in the environment variable)
    - Perform a CNAME swap or Route 53 update to route all traffic to the new environment
    - Terminate the old environment (RDS won't be deleted)
    - Delete CF stack (which is in DELETE_FAILED state)



Elastic Beanstalk

Running containerized applications on EB

Single Container

- **Running the application as a single container does not use orchestration solutions like ECS.** It runs it directly on an EC2 instance.
- Need to provide either:
    - Dockerfile - EB will build and run the Docker container (doesn't require a pre-built docker image in a container repository)
    - Dockerrun.aws.json (v1) - describes where the prebuilt Docker image is along with how to run it (ports, volumes, etc.)

Multi Container

- **Running the application in a multi-container setup creates an ECS cluster along with task definition and executions.** It also provisions ELB in **high availability mode**.
- Runs multiple containers in each EC2 instance.
- Requires Dockerrun.aws.json (v2) at the root of the source code. It is used by EB to generate the ECS task definition.
- The Docker images must be pre-built and stored in a container repository.

HTTPS on Elastic Beanstalk

- TLS certificate can be loaded on to the load balancer in either of the two ways:
    - From the EB console under load balancer configuration
    - In the .ebextensions/securelistener-alb.config file
- SSL cert can be provisioned using ACM or CLI
- Must configure a SG rule to allow incoming traffic on port 443

Redirect HTTP to HTTPS on Beanstalk

- Can be done in either of two ways:
    - Configure EC2 instances to redirect HTTP to HTTPS
    - Configure ALB with a rule (preferred)
- Health Checks should not be redirected

Elastic Beanstalk

Custom Platform

- If your app's language is incompatible with Beanstalk and does not use Docker, we can use Beanstalk and need to create a custom platform.
- Steps to create custom platform:
    - Define an AMI using Platform.yaml file
    - Build the platform using **Packer** (open-source tool to create AMIs)

Elastic Beanstalk

Working Process

How Elastic Beanstalk Works: Simple Steps

1. **Create an application:**
    - You start by creating an Elastic Beanstalk application. This application serves as a container for your environments (like development, testing, or production).
2. **Upload Your Code:**
    - You upload your application code (e.g., a ZIP file containing your web application) to Elastic Beanstalk. The platform supports various formats and languages.
3. **Choose a Platform:**
    - Select the platform that matches your application, such as Node.js, Python, Java, or Docker. Elastic Beanstalk provides pre-configured environments for these platforms.
4. **Deploy the Application:**
    - Elastic Beanstalk automatically handles the deployment of your code. It sets up the environment, including EC2 instances, load balancers, auto-scaling groups, and databases if needed.
5. **Environment Configuration:**
    - You can customize the environment settings, such as instance type, environment variables, and scaling policies, either during or after deployment. Elastic Beanstalk provides a console and CLI for configuration.
6. **Automatic Scaling and Load Balancing:**
    - Elastic Beanstalk monitors the health of your application and automatically scales the number of instances up or down based on demand. It also uses load balancers to distribute traffic among instances to ensure high availability.
7. **Monitor and Manage the Application:**
    - You can monitor the application's health, logs, and metrics using the Elastic Beanstalk dashboard. Elastic Beanstalk also provides notifications for environment changes and issues.
8. **Update or Rollback:**
    - You can easily update your application by uploading a new version of your code. If something goes wrong, Elastic Beanstalk allows you to quickly roll back to a previous version.

Elastic Beanstalk

9. **Terminate the Environment:**
    - When your application is no longer needed, you can terminate the environment, which will delete all associated resources, such as EC2 instances, load balancers, and auto-scaling groups.

Real-Time Use Case: Deploying a Web Application

Scenario: Imagine you have developed a simple web application using Node.js to manage a blog.

Steps Using Elastic Beanstalk:

1. **Develop Your Application:**
    - You've built your Node.js blog application on your local machine, and it's ready for deployment.
2. **Create an Application on Elastic Beanstalk:**
    - Log in to the AWS Management Console, navigate to Elastic Beanstalk, and create a new application. Name it "MyBlogApp."
3. **Upload Your Code:**
    - Zip your Node.js project files and upload them to Elastic Beanstalk as a new application version.
4. **Select a Platform:**
    - Choose the Node.js platform from the list of supported platforms.
5. **Deploy the Application:**
    - Click "Create Environment" and select "Web server environment." Elastic Beanstalk will automatically set up the necessary EC2 instances, load balancers, security groups, and other infrastructure components required to run your blog application.
6. **Configure Environment Settings:**
    - Customize the environment settings, such as specifying a custom domain name for your blog, setting environment variables (e.g., database connection strings), and choosing the instance type.

Elastic Beanstalk

7. **Automatic Scaling:**
    - As your blog becomes popular and traffic increases, Elastic Beanstalk automatically scales the EC2 instances to handle more requests. It uses the load balancer to distribute traffic among all running instances.
8. **Monitor the Application:**
    - Use the Elastic Beanstalk dashboard to monitor the health and performance of your application. You can view logs, check metrics like CPU utilization, and set up alerts for any potential issues.
9. **Update the Blog Application:**
    - Suppose you want to add a new feature, like a comments section. You make the changes in your code locally, zip the new version, and upload it to Elastic Beanstalk. It deploys the updated version seamlessly, with zero downtime.
10. **Rollback if Needed:**

- If there's an issue with the new feature, you can quickly roll back to the previous version with a few clicks in the Elastic Beanstalk console.

11. **Terminate When Done:**

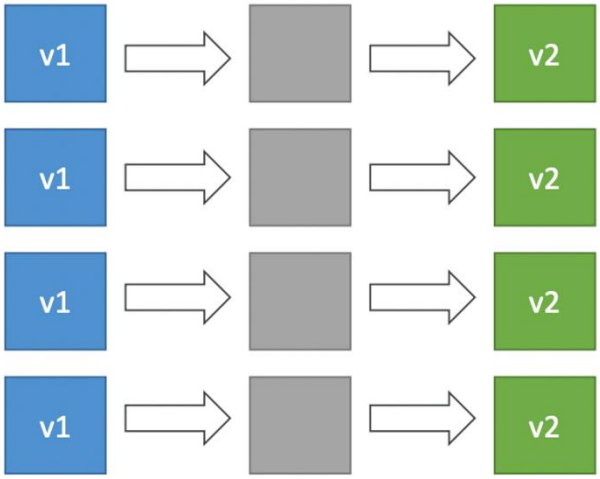
- If you decide to shut down the blog, you can terminate the Elastic Beanstalk environment, and all associated resources will be automatically deleted, ensuring you are not billed for unused resources.

Elastic Beanstalk

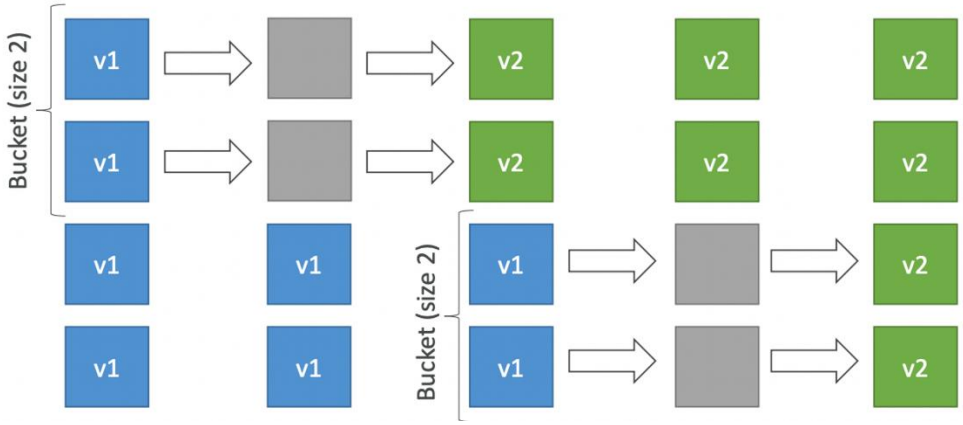
Deployment options for Update

Deployment Options in Elastic Beanstalk

1. **All at Once (Immune Deployment):**
    - **How It Works:** Deploys the new version to all instances at the same time.
    - **Pros:** Fastest deployment method; all instances are updated immediately.
    - **Cons:** Causes downtime because all instances are replaced at once. If there is an issue with the new version, the whole application might be affected.
    - **Use Case:** Best for development or test environments where downtime is acceptable, and you want to quickly see the effects of your updates.



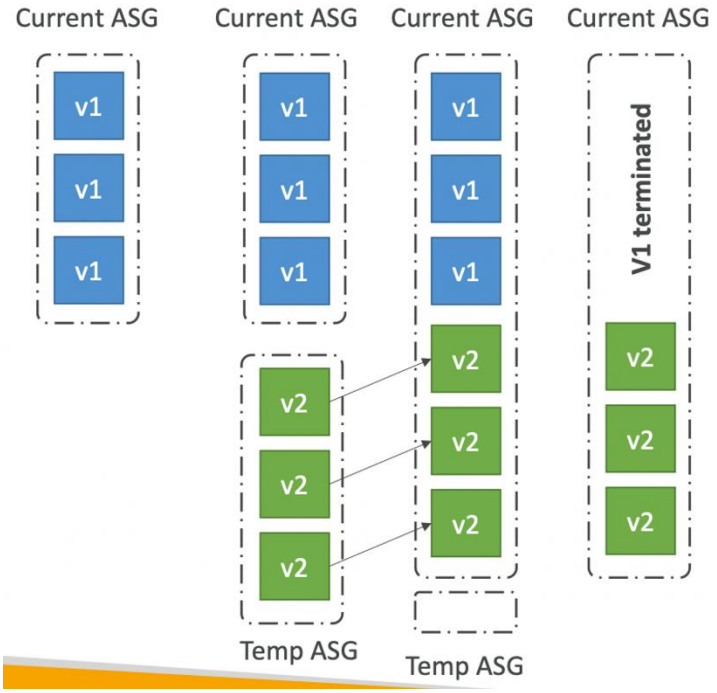
2. **Rolling Deployment**:
    - **How It Works**: Updates a batch of instances at a time, gradually rolling out the new version to all instances.
    - **Pros**: Reduces downtime as only a portion of instances are taken offline at any time.
    - **Cons**: Takes longer to complete the deployment. The application runs with both old and new versions until the deployment is finished.
    - **Use Case**: Useful for environments where minimizing downtime is important but having instances temporarily running different versions is acceptable, such as in staging or internal tools.



3. **Rolling with Additional Batch:**
    - **How It Works:** Similar to Rolling Deployment, but adds a temporary batch of instances to maintain full capacity during deployment.
    - **Pros:** Ensures full capacity during deployment, reducing impact on performance. It minimizes downtime and maintains stability.
    - **Cons:** Increases costs temporarily due to the additional instances.
    - **Use Case:** Ideal for production environments were maintaining full capacity and reducing downtime are critical, such as customer-facing applications.

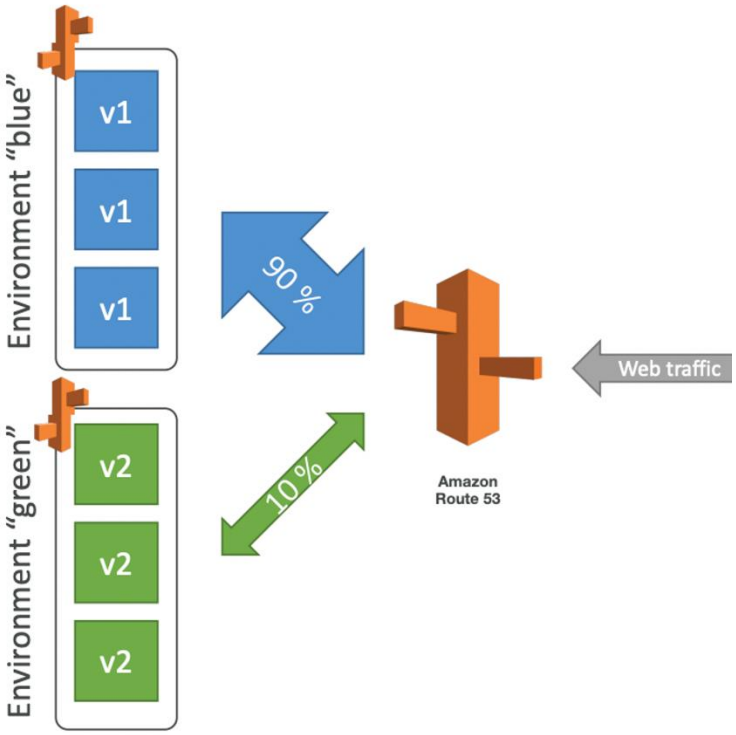
Elastic Beanstalk

4. **Immutable Deployment**:
    - **How It Works**: Creates a new set of instances with the new version and swaps them in only after they are fully ready.
    - **Pros**: Zero downtime and reduced risk. If something goes wrong, the old instances remain unaffected, allowing an easy rollback.
    - **Cons**: Slower deployment and temporarily doubles the number of instances, which increases costs.
    - **Use Case**: Best for mission-critical applications where downtime is unacceptable, and you want to minimize risk during updates, such as financial services or healthcare applications.



Elastic Beanstalk

5. **Blue/Green Deployment**:
    - **How It Works**: Deploys the new version to a separate environment (green environment) and then switches the traffic from the old environment (blue environment) to the new one.
    - **Pros**: No downtime and easy rollback by switching back to the old environment. It provides a safe way to test the new version in a production-like environment.
    - **Cons**: Requires managing multiple environments, which can be complex and increase costs.
    - **Use Case**: Ideal for applications where new features need to be tested thoroughly in production conditions before full rollout, such as e-commerce sites during a major sale event.
    -



Elastic Beanstalk

Real-Time Use Cases

1. **All at Once Deployment**:
    - **Scenario**: You're working in a development environment on a new feature for your blog application, and you want to quickly deploy and test changes.
    - **Action**: Use the "All at Once" deployment option to immediately update all instances. Since it's a development environment, downtime is acceptable.
2. **Rolling Deployment**:
    - **Scenario**: You have an internal company tool with regular users, but it's not customer-facing. You need to deploy updates without taking the entire tool offline.
    - **Action**: Use the "Rolling" deployment option to update a few instances at a time. This way, some users might experience minor disruption, but the tool remains mostly available.
3. **Rolling with Additional Batch**:
    - **Scenario**: Your web application is customer-facing, and any downtime can lead to lost sales or unhappy users.
    - **Action**: Choose "Rolling with Additional Batch" to ensure that the application maintains full capacity while deploying updates, minimizing any impact on users.
4. **Immutable Deployment**:
    - **Scenario**: You manage a healthcare application where any downtime or errors could critically impact users.
    - **Action**: Use "Immutable" deployment to create new instances with the updated application. If the update fails, you can easily roll back by terminating the new instances, ensuring high availability and reliability.
5. **Blue/Green Deployment**:
    - **Scenario**: You are launching a major update to your e-commerce site ahead of Black Friday. You need to ensure the new version works flawlessly under production load conditions.
    - **Action**: Deploy the new version to a separate green environment. Test it thoroughly, and once confirmed stable, switch traffic from the blue (old) environment to the green environment with zero downtime. If issues arise, you can quickly switch back to the blue environment.

Elastic Beanstalk

Keep in mind

- To deploy a new application version through the console, you'll need to upload a source bundle that meets the following requirements:
    - Consist of a **single** ZIP file or WAR file
    - Not exceed 512 MB
    - Not include a parent folder or top-level directory (subdirectories are fine)
- To deploy a worker application that processes periodic background tasks, the application bundle must include a cron.yaml file.
- EBS can configure EC2, CloudWatch and ALB. It cannot configure **Lambda** or CloudFront.
- Environment variables can be defined in env.yaml present in the root of the source bundle.
- To deploy a new version of the application, package your application as a zip or war file and deploy it using eb deploy command.
- To migrate an EB environment between accounts, create a saved configuration in the first account and download it to your local machine. Make the account-specific parameter changes and upload to the S3 bucket in second account. From Elastic Beanstalk console, create an application from **Saved Configurations**.

Elastic Beanstalk

Scenario based Questions.

https://lisireddy.medium.com/aws-elastic-beanstalk-scenario-based-questions-a7969e07b1e6

Wish you the best …! Happy Learning ..!

Yours' Love (@lisireddy across all the platforms)