JON BONSO AND KENNETH SAMONTE

AWS CERTIFIED DEVOPS ENGINEER PROFESSIONAL

Tutorials Dojo
Study Guide and Cheat Sheets

Types of Blue Green Deployment via ELB, Route 53, Elastic Beanstalk

AWS Elastic Beanstalk Blue/Green

Elastic Beanstalk, by default, performs an in-place update when you deploy a newer version of your application. This can cause a short downtime since your application will be stopped while Elastic Beanstalk performs the application update.

Blue/Green deployments allow you to deploy without application downtime.

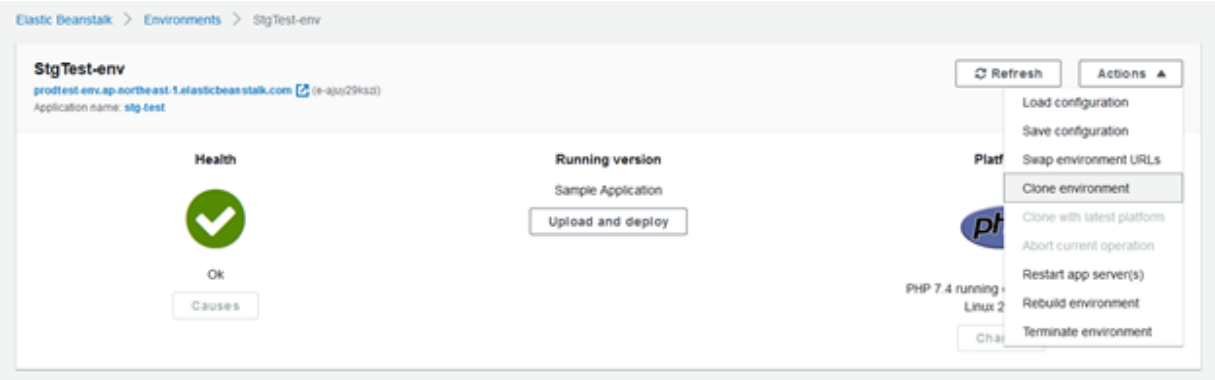
DevOps Exam Notes:

Remember these key points on when to use blue/green deployments:

- No downtime during deployment because you are deploying the newer version on a separate environment
- CNAMEs of the environment URLs are swapped to redirect traffic to the newer version.
- Route 53 will swap the CNAMEs of the application endpoints.
- Fast deployment time and quick rollback since both old and new versions are running at the same time, you just have to swap back the URLs if you need to rollback.
- Useful if your newer version is incompatible with the current platform version of your application. (ex. Jumping from major versions of NodeJS, Python, Ruby, PHP, etc.)
- Your RDS Database instance should be on a separate stack because the data will not transfer to your second environment. You should decouple your database from the web server stack.

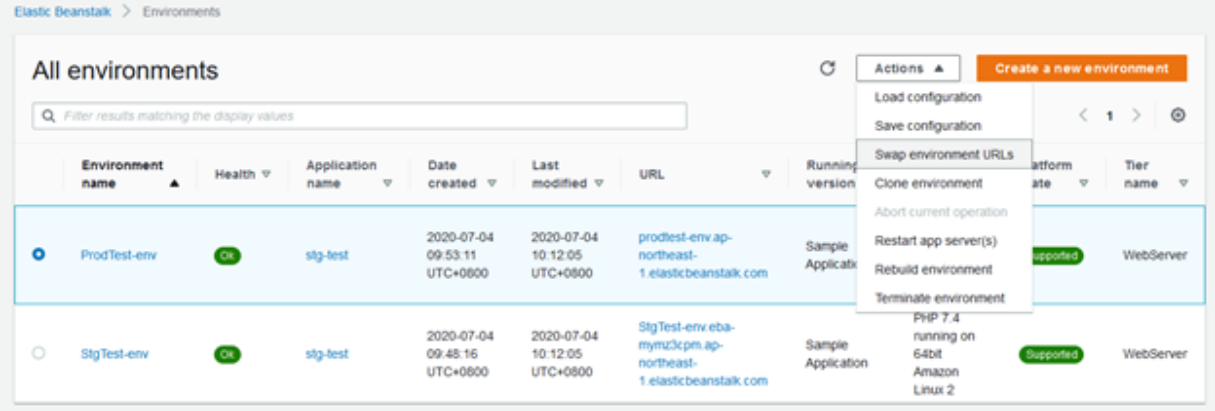
To implement a Blue/Green deployment for your Elastic Beanstalk application, you can perform the following steps:

1. Create another environment on which you will deploy the newer version of your application. You can clone your current environment for easier creation.

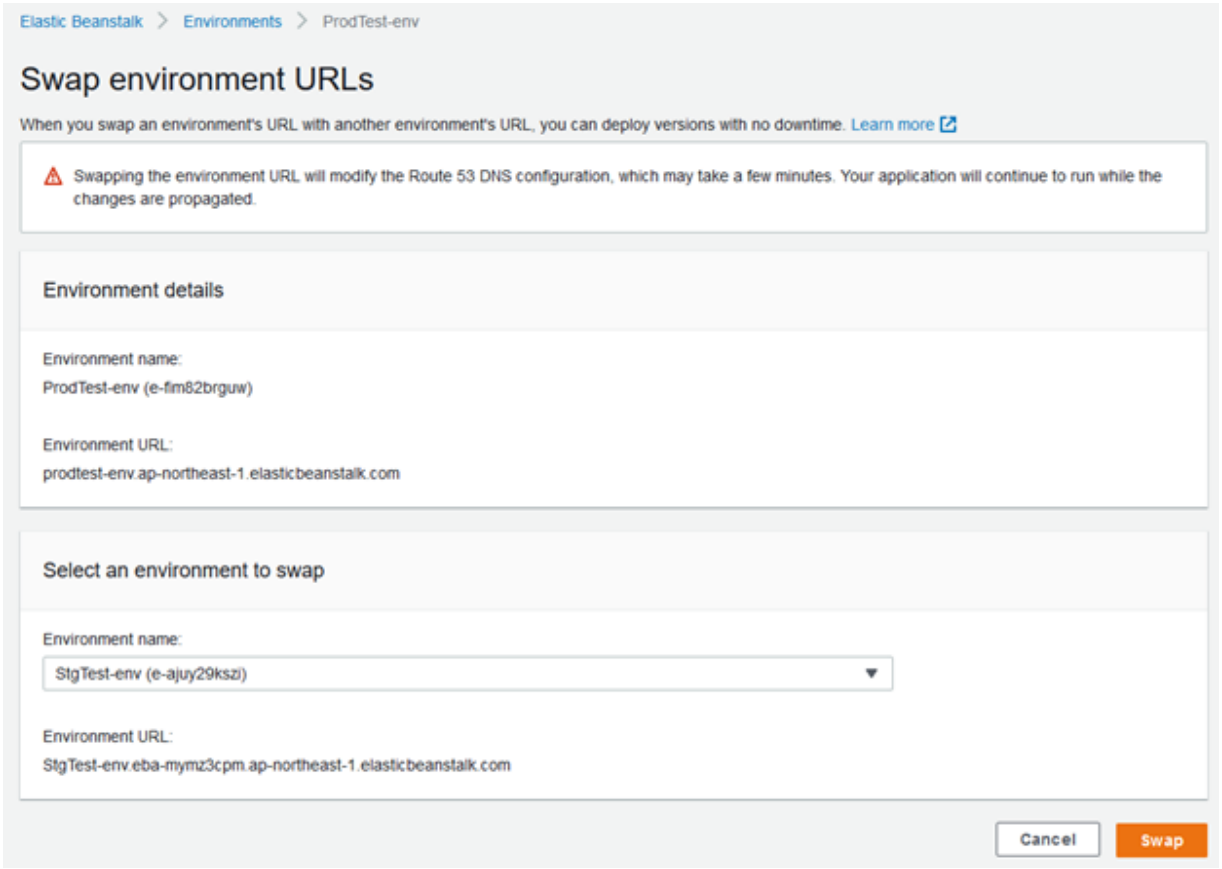


2. Once the new environment is ready, deploy a new version of your application. Perform your tests on the URL endpoint of your new environment.

3. After testing, select your Production environment, click Actions > Swap environment URLs.



4. On the Swap Environment URLs page, select the newer environment and click Swap to apply the changes.


Elastic Beanstalk > Environments > ProdTest-env
Swap environment URLs
When you swap an environment's URL with another environment's URL, you can deploy versions with no downtime. Learn more
Swapping the environment URL will modify the Route 53 DNS configuration, which may take a few minutes. Your application will continue to run while the changes are propagated.
Environment details
Environment name:
ProdTest-env (e-fim82brguw)
Environment URL:
prodtest-env.ap-northeast-1.elasticbeanstalk.com
Select an environment to swap
Environment name:
StgTest-env (e-ajuy29kszi)
Environment URL:
StgTest-env.eba-mymz3cpm.ap-northeast-1.elasticbeanstalk.com
Cancel
Swap

AWS Lambda Blue/Green

You can also implement Blue/Green deployments on your Lambda functions. The concept is the same as in Elastic beanstalk blue/green deployment i.e. you will need to create two versions of your Lambda function and use function Aliases to swap the traffic flow.

Lambda versions – lets you publish a new version of a function that you can test without affecting the current application accessed by users. You can create multiple versions as needed for your testing environments. The ARN of Lambda version is the same as the ARN of the Lambda function with added version suffix.

arn:aws:lambda:aws-region:acct-id:function:helloworld:$LATEST

Lambda aliases – Aliases are merely pointers to specific Lambda versions. You can't select a Lambda alias and edit the function. You need to select the LATEST version if you want to edit the function. Aliases are helpful for blue/green deployments because it allows you to use a fixed ARN and point it to a particular Lambda version that you want to deploy.

DevOps Exam Notes:

Remember the difference between Lambda $LATEST, Lambda Versions and Lambda Aliases:

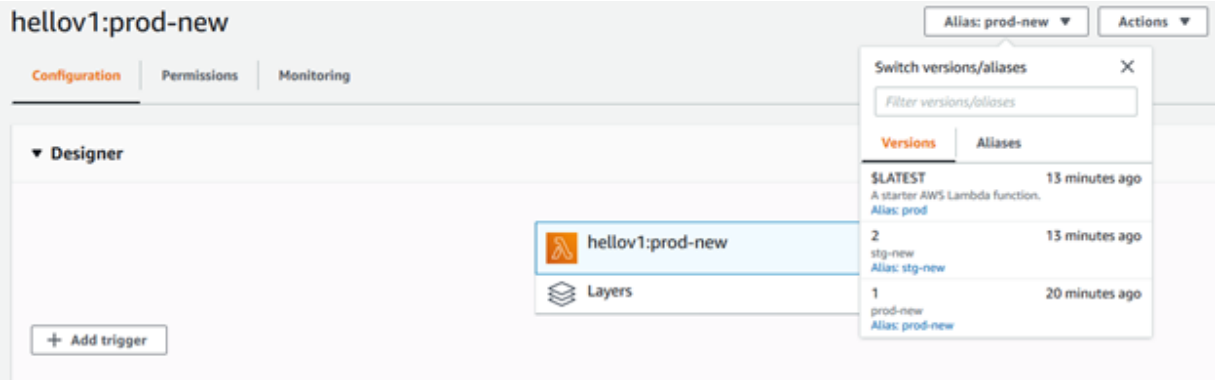
$LATEST - this is the latest version of your Lambda function. You can freely edit this version.

Lambda Version - fixed version of your function. You can't edit this directly.

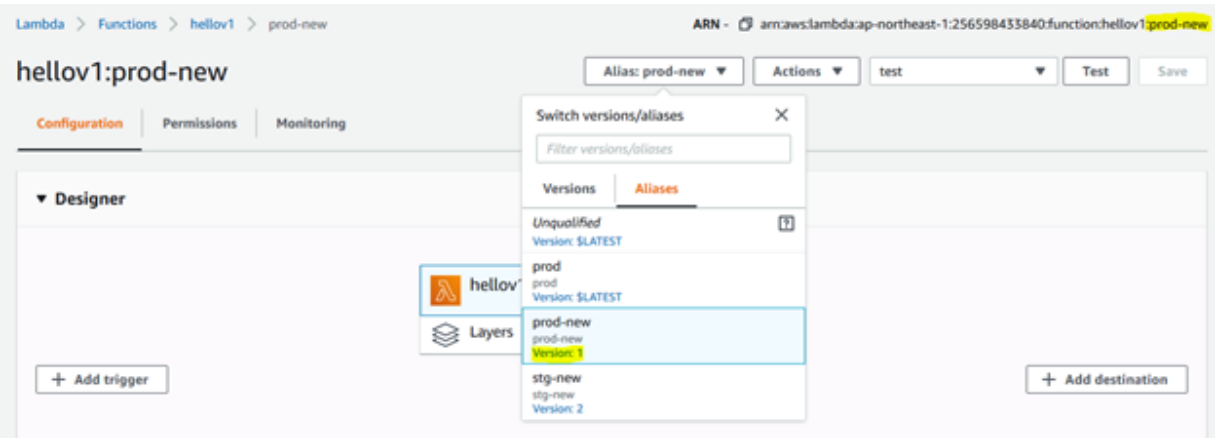
Lambda Alias - a pointer to a specific Lambda version. You can perform blue/green deployment with Aliases by pointing to a newer version.

The following steps will show how blue/green deployment can be done on Lambda functions.

1. The current version of your Lambda function is deployed on Version 1. Create another version and make your changes, this will be Version 2.



2. Create an Alias that will point to the current production version. Use this alias as your fixed production ARN.



3. Create another Alias that you will use for your newer version. Perform your testing and validation on this newer version. Once testing is complete, edit the production alias to point to the newer version. Traffic will now instantly be shifted from the previous version to the newer version.

Sources:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.CNAMESwap.html
https://docs.aws.amazon.com/lambda/latest/dg/configuration-aliases.html
https://docs.aws.amazon.com/lambda/latest/dg/configuration-versions.html

Elastic Beanstalk - Deployment Policies and Settings

AWS Elastic Beanstalk provides several options for how deployments are processed, including deployment policies (All at once, Rolling, Rolling with additional batch, Immutable, and Traffic splitting) and options that let you configure batch size and health check behavior during deployments. By default, your environment uses all-at-once deployments.

All at once – 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.

Rolling deployments - 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. 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 deployment with additional batch - launches new batches during the deployment. To maintain full capacity during deployments, you can configure your environment to launch a new batch of instances before taking any instances out of service. When the deployment completes, Elastic Beanstalk terminates the additional batch of instances.

Immutable deployments - perform an immutable update to launch a full set of new instances running the new version of the application in a separate Auto Scaling group, alongside the instances running the old version. Immutable deployments can prevent issues caused by partially completed rolling deployments. If the new instances don't pass health checks, Elastic Beanstalk terminates them, leaving the original instances untouched.

Traffic-splitting deployments - let you perform canary testing as part of your 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 don't pass health checks, or if you choose to abort the deployment, Elastic Beanstalk moves traffic back to the old instances and terminates the new ones.

Here's a summary of the deployment methods, how long each deployment takes, and how rollback is handled.

Method	Impact of failed deployment	Deploy time	Zero downtime	No DNS change	Rollback process	Code deployed to
All at once	Downtime	🕒	No	Yes	Manual redeploy	Existing instances
Rolling	Single batch out of service; any successful batches before failure running new application version	🕒 🕒 †	Yes	Yes	Manual redeploy	Existing instances
Rolling with an additional batch	Minimal if first batch fails; otherwise, similar to **Rolling**	🕒 🕒 🕒 †	Yes	Yes	Manual redeploy	New and existing instances
Immutable	Minimal	🕒 🕒 🕒 🕒	Yes	Yes	Terminate new instances	New instances
Traffic splitting	Percentage of client traffic routed to new version temporarily impacted	🕒 🕒 🕒 🕒 ††	Yes	Yes	Reroute traffic and terminate new instances	New instances
Blue/green	Minimal	🕒 🕒 🕒 🕒	Yes	No	Swap URL	New instances

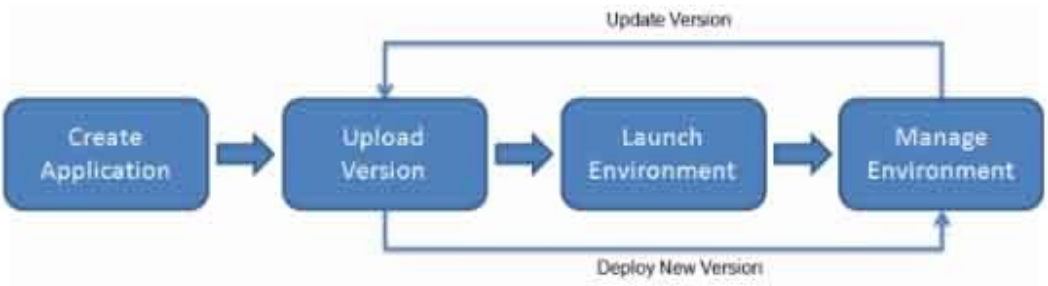
† *Varies depending on batch size.*

†† *Varies depending on* ***evaluation time*** *option setting.*

Sources:
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.rolling-version-deploy.html
https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html

AWS Elastic Beanstalk

- Allows you to quickly deploy and manage applications in the AWS Cloud without worrying about the infrastructure that runs those applications.
- Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring for your applications.
- Elastic Beanstalk supports Docker containers.
- Elastic Beanstalk Workflow



- Your application's domain name is in the format: *subdomain.region*.elasticbeanstalk.com

Elastic Beanstalk Concepts

- **Application** - a logical collection of Elastic Beanstalk components, including environments, versions, and environment configurations. It is conceptually similar to a folder.
- **Application Version** - refers to a specific, labeled iteration of deployable code for a web application. An application version points to an Amazon S3 object that contains the deployable code. Applications can have many versions and each application version is unique.
- **Environment** - a version that is deployed on to AWS resources. Each environment runs only a single application version at a time, however you can run the same version or different versions in many environments at the same time.
- **Environment Tier** - determines whether Elastic Beanstalk provisions resources to support an application that handles HTTP requests or an application that pulls tasks from a queue. An application that serves HTTP requests runs in a **web server environment**. An environment that pulls tasks from an Amazon SQS queue runs in a **worker environment**.
- **Environment Configuration** - identifies a collection of parameters and settings that define how an environment and its associated resources behave.
- **Configuration Template** - a starting point for creating unique environment configurations.
- There is a limit to the number of application versions you can have. You can avoid hitting the limit by applying an *application version lifecycle policy* to your applications to tell 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.

AWS CodePipeline

- A fully managed **continuous delivery service** that helps you automate your release pipelines for application and infrastructure updates.
- You can easily integrate AWS CodePipeline with third-party services such as GitHub or with your own custom plugin.
- **Concepts**
    - A **pipeline** defines your release process workflow, and describes how a new code change progresses through your release process.
    - A pipeline comprises a series of **stages** (e.g., build, test, and deploy), which act as logical divisions in your workflow. Each stage is made up of a sequence of actions, which are tasks such as building code or deploying to test environments.
        - Pipelines must have **at least two stages**. The first stage of a pipeline is required to be a source stage, and the pipeline is required to additionally have at least one other stage that is a build or deployment stage.
    - Define your pipeline structure through a **declarative JSON** document that specifies your release workflow and its stages and actions. These documents enable you to update existing pipelines as well as provide starting templates for creating new pipelines.
    - A **revision** is a change made to the source location defined for your pipeline. It can include source code, build output, configuration, or data. A pipeline can have multiple revisions flowing through it at the same time.
    - A **stage** is a group of one or more actions. A pipeline can have two or more stages.
    - An **action** is a task performed on a revision. Pipeline actions occur in a specified order, in serial or in parallel, as determined in the configuration of the stage.
        - You can add actions to your pipeline that are in an AWS Region different from your pipeline.
        - There are six types of actions
            - Source
            - Build
            - Test
            - Deploy
            - Approval
            - Invoke
    - When an action runs, it acts upon a file or set of files called **artifacts**. These artifacts can be worked upon by later actions in the pipeline. You have an artifact store which is an S3 bucket in the same AWS Region as the pipeline to store items for all pipelines in that Region associated with your account.
    - The stages in a pipeline are connected by **transitions.** Transitions can be disabled or enabled between stages. If all transitions are enabled, the pipeline runs continuously.

    - An **approval action** prevents a pipeline from transitioning to the next action until permission is granted. This is useful when you are performing code reviews before code is deployed to the next stage.
- **Features**
    - AWS CodePipeline provides you with a graphical user interface to create, configure, and manage your pipeline and its various stages and actions.
    - A pipeline starts automatically (default) when a change is made in the source location, or when you manually start the pipeline. You can also set up a rule in CloudWatch to automatically start a pipeline when events you specify occur.
    - You can model your build, test, and deployment actions to run **in parallel** in order to increase your workflow speeds.
    - AWS CodePipeline can pull source code for your pipeline directly from AWS CodeCommit, GitHub, Amazon ECR, or Amazon S3.
    - It can run builds and unit tests in AWS CodeBuild.
    - It can deploy your changes using AWS CodeDeploy, AWS Elastic Beanstalk, Amazon ECS, AWS Fargate, Amazon S3, AWS Service Catalog, AWS CloudFormation, and/or AWS OpsWorks Stacks.
    - You can use the CodePipeline Jenkins plugin to easily register your existing build servers as a custom action.
    - When you use the console to create or edit a pipeline that has a GitHub source, CodePipeline creates a **webhook**. A webhook is an HTTP notification that detects events in another tool, such as a GitHub repository, and connects those external events to a pipeline. CodePipeline deletes your webhook when you delete your pipeline.
- As a best practice, when you use a Jenkins build provider for your pipeline's build or test action, install Jenkins on an Amazon EC2 instance and configure a separate EC2 instance profile. Make sure the instance profile grants Jenkins only the AWS permissions required to perform tasks for your project, such as retrieving files from Amazon S3.

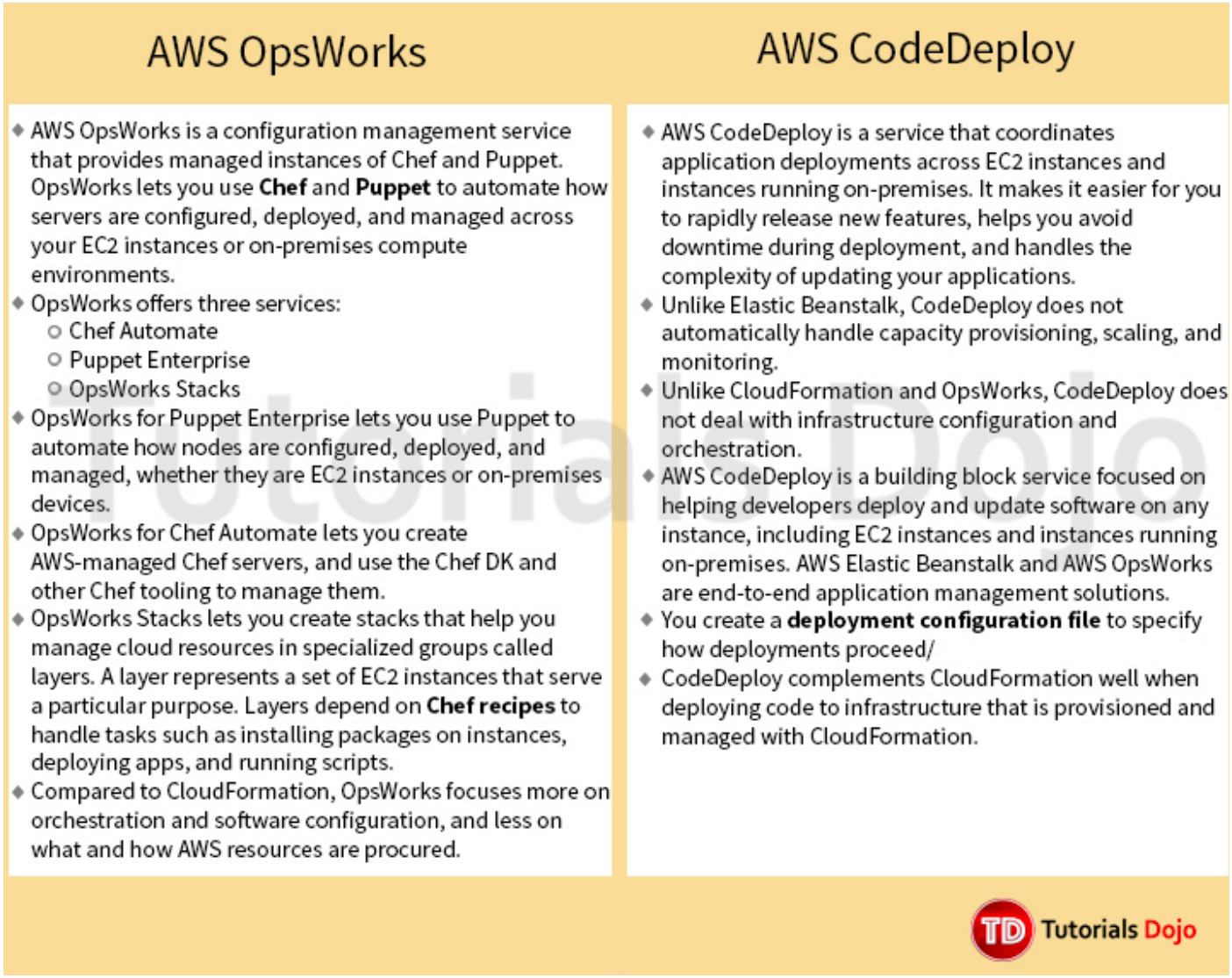
Elastic Beanstalk vs CloudFormation vs OpsWorks vs CodeDeploy

AWS Elastic Beanstalk

- AWS Elastic Beanstalk makes it even easier for developers to **quickly deploy and manage applications** in the AWS Cloud. Developers simply upload their application, and Elastic Beanstalk **automatically handles the deployment details** of capacity provisioning, load balancing, auto-scaling, and application health monitoring.
- This **platform-as-a-service solution** is typically for those who want to deploy and manage their applications within minutes in the AWS Cloud without worrying about the underlying infrastructure.
- AWS Elastic Beanstalk supports the following languages and development stacks:
  - Apache Tomcat for Java applications
  - Apache HTTP Server for PHP applications
  - Apache HTTP Server for Python applications
  - Nginx or Apache HTTP Server for Node.js applications
  - Passenger or Puma for Ruby applications
  - Microsoft IIS for .NET applications
  - Java SE
  - Docker
  - Go
- Elastic Beanstalk also supports deployment versioning. It maintains a copy of older deployments so that it is easy for the developer to rollback any changes made on the application.

AWS CloudFormation

- AWS CloudFormation is a service that gives developers and businesses an easy way to create a **collection of related AWS resources** and provision them in an orderly and predictable fashion. This is typically known as **"infrastructure as code".**
- The main difference between CloudFormation and Elastic Beanstalk is that CloudFormation deals more with the AWS infrastructure rather than applications. AWS CloudFormation introduces two concepts:
  - The **template**, a JSON or YAML-format, text-based file that describes all the AWS resources and configurations you need to deploy to run your application.
  - The **stack**, which is the set of AWS resources that are created and managed as a single unit when AWS CloudFormation instantiates a template.
- CloudFormation also supports a rollback feature through template version controls. When you try to update your stack but the deployment failed midway,
- CloudFormation will automatically revert the changes back to their previous working states.
- CloudFormation supports Elastic Beanstalk application environments. This allows you, for example, to create and manage an AWS Elastic Beanstalk–hosted application along with an RDS database to store the application data.
- AWS CloudFormation can be used to bootstrap both Chef (Server and Client) and Puppet (Master and Client) softwares on your EC2 instances.
- CloudFormation also supports OpsWorks. You can now model OpsWorks components (stacks, layers, instances, and applications) inside CloudFormation templates, and provision them as CloudFormation stacks. This enables you to document, version control, and share your OpsWorks configuration.
- AWS CodeDeploy is a recommended adjunct to CloudFormation for managing the application deployments and updates.



Additional Notes:

- Elastic Beanstalk, CloudFormation, or OpsWorks are particularly useful for **blue-green deployment method** as they provide a simple way to clone your running application stack.
- CloudFormation and OpsWorks are best suited for the **prebaking AMIs**.
- CodeDeploy and OpsWorks are best suited for performing **in-place application upgrades**. For **disposable upgrades**, you can set up a cloned environment with Elastic Beanstalk, CloudFormation, and OpsWorks.