Overview of Deployment Options on AWS

Peter Dalbhanjan

March 2015

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Introduction 4
AWS Deployment Services 5
AWS Elastic Beanstalk 5
AWS CloudFormation 6
AWS OpsWorks 6
AWS CodeCommit 6
AWS CodePipeline 6
AWS CodeDeploy 7
Amazon EC2 Container Service 7
Common Features 7
Provision 8
Deploy 9
Configure 9
Scale 9
Monitoring 9
Logging 10
Instance Profiles 10
Custom Variables 11
Other AWS Service Integration 11
Tags 11
Strategies for Updating Your Stacks 12
Prebaking AMIs 12
In-place vs Disposable Method 13
Blue-Green Method 15
Hybrid Deployment Model Approach 17
Conclusion 20

Abstract

Amazon Web Services offers multiple options for provisioning your IT infrastructure and the deployment of your applications. Whether it is a simple three-tier application or a complex set of workloads, the deployment model varies from customer to customer. But with the right techniques, AWS can help you pick the best strategy and tool set for deploying an infrastructure that can handle your workload.

This whitepaper is intended for anyone looking for information on different deployment services in AWS. It lays out common features available on these deployment services, articulates strategies for updating application stacks, and presents few examples of hybrid deployment models for complex workloads

Introduction

AWS caters to multiple customers with several unique requirements. If you are an experienced user working on the AWS platform, you are probably aware of the "one size *doesn't* fit all" philosophy. Whether you work in enterprise computing or hope to create the next big social media or gaming company, AWS provides multiple customization options to serve a broad range of use cases. The AWS platform is designed to address scalability, performance, security, ease of deployment, tools to help migrate applications and an ecosystem of developers and architects who are deeply involved in the growth of its products and services.

For example, several sizing options are available to roll out an application on Amazon Elastic Compute Cloud (EC2) instance along with various scaling mechanics for adding compute and storage capacity.[1] For persistent data storage needs, Amazon Elastic Block Store (EBS) has tiered offerings such as general purpose (SSD), provisioned IOPS (SSD) and magnetic EBS volumes.[2] For data that is static in nature, you can use Amazon Simple Storage Service (S3)[3] and Amazon Glacier[4] for archival purposes. For data that is relational in nature, you can leverage Amazon Relational Database Service (RDS);[5] for data warehousing, you can use Amazon Redshift.[6] If you need storage with pre-defined throughput, you can leverage Amazon DynamoDB[7] and for real-time processing, you can use Amazon Kinesis.[8]

Similarly, when it comes to deployment services, AWS has multiple options too. The following diagram summarizes different deployment services in AWS.

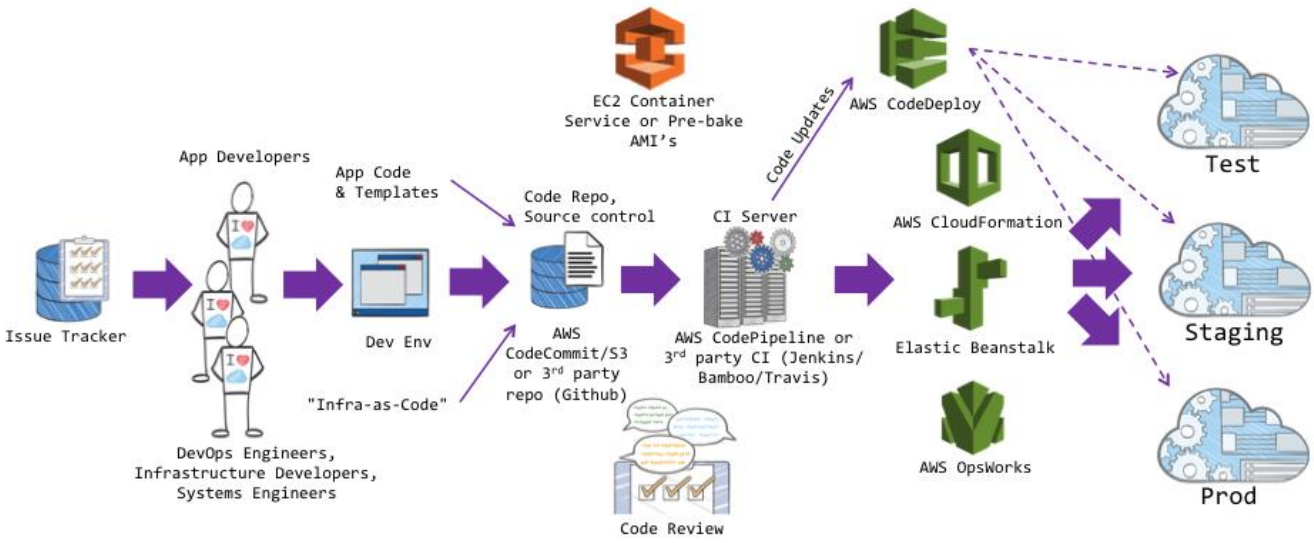

Figure 1: Overview of Deployment Services

AWS Deployment Services

AWS offers multiple strategies for provisioning infrastructure. You could use the building blocks (Amazon EC2, Amazon EBS, Amazon S3, Amazon RDS) and leverage the integration provided by third-party tools to deploy your application. But for even greater flexibility, you can consider the automation provided by the AWS deployment services.

AWS Elastic Beanstalk

AWS Elastic Beanstalk is the fastest and simplest way to get an application up and running on AWS.[9] It is perfect for developers who want to deploy code and not worry about managing the underlying infrastructure. Elastic Beanstalk is ideal if you have a standard three tier PHP, Java, Python, Ruby, Node.js, .NET, Go or Docker application that can run on an app server with a database.[10] Elastic Beanstalk uses Auto Scaling[11] and Elastic Load Balancing[12] to easily support highly variable amounts of traffic and works for you if you want to start small and scale up. Common use cases include web apps, content management systems (CMS), and API back ends.

AWS CloudFormation

AWS CloudFormation provides the sysadmin, network architect, and other IT personnel the ability to provision and manage stacks of AWS resources based on templates you create to model your infrastructure architecture.[13] You can manage anything from a single Amazon EC2 instance to a complex multitier, multiregional application. Using templates means you can impose version control on your infrastructure and easily replicate your infrastructure stack quickly and with repeatability. AWS CloudFormation is recommended if you want a tool for granular control over the provisioning and management of your own infrastructure. AWS CodeDeploy is a recommended adjunct to AWS CloudFormation for managing the application deployments and updates.[14]

AWS OpsWorks

AWS OpsWorks is an application-management service that makes it easy for both developers and operations personnel to deploy and operate applications of all shapes and sizes.[15] AWS OpsWorks works best if you want to deploy your code, have some abstraction from the underlying infrastructure, and have an application more complex than a three-tier architecture. AWS OpsWorks is also recommended if you want to manage your infrastructure with a configuration management system such as Chef.

AWS CodeCommit

AWS CodeCommit is a highly available, highly scalable managed source-control service that hosts private Git repositories.[16] With AWS CodeCommit, you can store anything from code to binaries and work seamlessly with your existing Git-based tools. CodeCommit integrates with AWS CodePipeline and AWS CodeDeploy to streamline your development and release process.

AWS CodePipeline

AWS CodePipeline is a continuous delivery and release automation service for rapidly releasing new features to users.[17] With AWS CodePipeline, you can design your development workflow for checking in code, building the code, deploying your application into staging, testing it, and releasing it to production. AWS CodePipeline can be easily integrated or extended by leveraging third-party tools into any step of your release process or you can use AWS CodePipeline as an end-

to-end solution. For best results, use AWS CodeCommit along with AWS CodePipeline to streamline your development and release cycles.

AWS CodeDeploy

AWS CodeDeploy is a service that coordinates application deployments across Amazon EC2 instances.[18] AWS CodeDeploy works with your existing application files and deployment scripts, and it can easily reuse existing configuration management scripts. The service scales with your infrastructure so you can deploy to as few as one EC2 instance or thousands. AWS CodeDeploy is a good choice if you want to deploy code to infrastructure managed by yourself or other people in your organization. Use AWS CodeDeploy to deploy code to infrastructure that is provisioned and managed with AWS CloudFormation. Even if you don't use AWS CloudFormation but you use Amazon EC2 with third-party integration, AWS CodeDeploy can help manage your application deployment.

Amazon EC2 Container Service

Amazon EC2 Container Service[19] is a highly scalable, high performance container management service that makes it easy to run, stop, and manage Docker containers on a cluster of Amazon EC2 instances. With Amazon EC2 Container Service you can manage container-enabled applications with simple API calls, get the state of your cluster from a centralized service, and gain access to many familiar Amazon EC2 features like security groups,[20] Amazon EBS volumes, and AWS Identity and Access Management (IAM) roles.[21] Amazon EC2 Container Service is a good option if you are using Docker for a consistent build and deployment experience, if you want to improve the utilization of your EC2 instances, or as the basis for sophisticated distributed systems.

Common Features

AWS offers several key features that are unique to each deployment service. However, there are some characteristics that are common to these services. Each feature can influence service adoption in its own way. The following table discusses some of the common features in the deployment services:

Deployment Feature[1]	Category	Description	AWS Elastic Beanstalk	AWS Cloud Formation	AWS OpsWorks	AWS CodeDeploy
Provision	Provision infrastructure	EC2 instances, Amazon EBS volumes, VPC, etc.	√ (Details) [2]	√ (Details)	√ (Details)	N/A
Deploy	Deploy applications	Deploy applications from chosen repository	√ (Details) [2]	√ (Details)	√ (Details)	√ (Details)
Configure	Configuration management	Install software, configure software and AWS resources	√ (Details)	√ (Details)	√ (Details)	
Scale	Scalability	Automatically scale to handle the load	√ (Details)	√ (Details)	√ (Details)	N/A
Monitoring	Monitoring	Monitor events, resources, application health	√ (Details)	√ (Details)	√ (Details)	√ (Details)
Logging	Troubleshooting, security	System, application logs	√ (Details)	√ (Details)	√ (Details)	√ (Details)
Instance profiles	Security	Securely access AWS services such as Amazon S3, DynamoDB	√ (Details)	√ (Details)	√ (Details)	√ (Details)
Custom variables	Configuration management	Pass variables to application environments	√ (Details)	√ (Details)	√ (Details)	N/A
Other AWS service integration	Service integration	Integration with other AWS services	√ (Details)	√ (Details)	√ (Details)	√ (Details)
Tags	Security, troubleshooting, configuration management	Automate configuring tags on EC2, Amazon RDS	√ (Details)	√ (Details)	√ (Details)	

1. Lists only the relevant deployment service with the common feature set.

2. Elastic Beanstalk provisions the resources to support either web application that handles HTTP(S) requests or a web application that handles background-processing tasks.

Provision

As mentioned earlier, you can work with the building blocks such as Amazon EC2, Amazon EBS, Amazon S3, Amazon Virtual Private Cloud (VPC)

individually,[22] or you can use the automation provided by deployment services for provisioning infrastructure components. The advantage of using these services is the rich feature set they bring for deploying and configuring your application, monitoring, scalability, integration with other AWS services and more. A detailed discussion of these features will make this clear.

Deploy

The deployment services can also make it easier to deploy your application on the underlying infrastructure. You can create an application, specify the source to your desired deployment service, and let the tool handle the complexity of provisioning the AWS resources needed to run your application. Despite providing similar functionality in terms of deployment, each service has its own unique method for deploying and managing your application.

Configure

In addition to deploying your application, you can use the deployment services to customize and manage the application configuration. The underlying task could be replacing custom configuration files (such as `httpd.conf`) for your custom web application or updating packages that are required by your application (such as yum and apt-get repositories). You can customize the software on your Amazon EC2 instance as well as the infrastructure resources in your stack configuration.

Scale

Scaling your application fleet during periods of increased demand not only provides a better experience for your end users but also keeps the cost low. You can configure Auto Scaling to dynamically add or remove Amazon EC2 instances based on metrics triggers that you set in Amazon CloudWatch (CPU, memory, disk I/O, network I/O).[23] This type of Auto Scaling configuration is integrated seamlessly into Elastic Beanstalk and AWS CloudFormation. Similarly, you can use AWS OpsWorks to automatically manage scaling based on time or load.

Monitoring

Monitoring gives you visibility into the resources you launch in the cloud. Whether you want to monitor the resource utilization of your overall stack or get

an overview of your application health, the deployment services help provide this info from single pane of glass. You can also navigate to the CloudWatch console to get a system-wide view into all of your resources and operational health. You can use similar techniques to create alarms for metrics that you want to monitor. Alarms can send an alert whenever a certain threshold is met or take an action to mitigate an issue. For example, you can set an alarm that sends an email alert when an EC2 instance fails on status checks or trigger a scaling event when the CPU utilization meets certain threshold.

Each deployment services provide the progress of your deployment. You can track the resources that are being created or removed via AWS Management Console,[24] CLI,[25] or APIs.[26]

Logging

Logging is an important element of your application deployment cycle. Logging can provide important debugging information or provide key characteristics of your application behavior. The deployment services make it simpler to access these logs through a combination of the AWS Management Console, CLI, and API methods so that you don't have to log into Amazon EC2 instances to view them.

In addition to built-in features, the deployment services provide seamless integration with CloudWatch Logs to expand your ability to monitor the system, application, and custom log files.[27] You can use CloudWatch Logs to monitor logs from EC2 instances in real time, monitor CloudTrail events, or archive log data in Amazon S3 for future analysis.[28]

Instance Profiles

Instance profiles[29] is a great way of embedding necessary IAM roles required to carry out an operation to access an AWS resource. These IAM roles can securely make API requests from your instances to AWS services without requiring you to manage security credentials. The deployment services integrate seamlessly with instance profiles to simplify credentials management and relieve you from hardcoding API keys in your application configuration.

For example, if your application needs to access an Amazon S3 bucket with read-only permission, you can create an instance profile and assign read-only Amazon

S3 access in the associated IAM role. The deployment service will take the complexity of passing these roles to EC2 instance so that your application can securely access AWS resource with the privileges that you define.

Custom Variables

When you develop an application, you want to customize configuration values such as database connection strings, security credentials, or other information that you don't want to hardcode into your application. Defining variables can help loosely couple your application configuration and gives you the flexibility to scale different tiers of your application independently. Embedding variables outside of your application code also helps improve portability of your application. Additionally, you can differentiate environments into development, test, and production based on customized variables. The deployment services help facilitate customizing variables so that once they are set, the variables become available to your application environments.

Other AWS Service Integration

AWS deployment services provide easier integration with other AWS services. Whether you need to load balance across multiple Availability Zones[30] by using Elastic Load Balancing or by using Amazon RDS as a back end, the deployment services like AWS Elastic Beanstalk, AWS CloudFormation, and AWS OpsWorks make it simpler to use these services as part of your deployment.

If you need to use other AWS services, you can leverage tool-specific integration methods to interact with the resource. For example, if you are using Elastic Beanstalk for deployment and want to use DynamoDB for your back end, you can customize your environment resources by including a configuration file within your application source bundle.[31] With AWS OpsWorks, you can create custom recipes to configure the application so that it can access other AWS services.[32] Similarly, several template snippets with a number of example scenarios are available for you to use within your AWS CloudFormation templates.[33]

Tags

Another advantage of using a deployment service is to reap the benefits of automating tag usage. A tag consists of a user-defined key and value. You can define tags based on application, project, cost centers, business division, and

more so that you can easily identify a resource. When you use tags during your deployment steps, the tools automatically propagate the tags to underlying resources such as Amazon EC2 instances, Auto Scaling groups, or Amazon RDS.

Appropriate use of tagging can provide a better way to manage your costs with cost allocation reports.[34] Cost allocation reports aggregate costs based on tags. This way, you can determine how much you are spending for each application or a particular project.

Strategies for Updating Your Stacks

Depending on your choice of deployment service, the strategy for updating your application code could vary a fair amount. AWS deployment services bring agility and improve the speed of your application deployment cycle, but using a proper tool and the right strategy is key for building a robust environment.

The following section looks at how the deployment service can help while performing application updates. The approaches mentioned below will start with prebaking machine images and then move to performing in-place and disposable upgrades.

Prebaking AMIs

An Amazon Machine Image (AMI) is an image consisting of a base operating system or an application server in the cloud.[35] In order to launch an EC2 instance, you need to choose which AMI you will use for installing your application. A common practice to install an application is during instance boot. This process is called bootstrapping an instance. AWS CloudFormation provides multiple options for bootstrapping an application. To review the options in detail, see Bootstrapping Applications via AWS CloudFormation .[36]

Note that the bootstrapping process can be slower if you have a complex application or multiple applications to install. Managing a fleet of applications with several build tools and dependencies can be a challenging task during rollouts. Furthermore, your deployment service should be designed to do faster rollouts to take advantage of Auto Scaling.

Prebaking is a process of embedding a significant portion of your application artifacts within your base AMI. During the deployment process you can customize application installations by using EC2 instance artifacts such as instance tags, instance metadata, and Auto Scaling groups.

For example, let's say you are managing a Ruby application that needs Nginx for the front end; Elasticsearch, Logstash, and Kibana for log processing; and MongoDB for document management. You can have logical grouping of your base AMIs that can take 80% of application binaries loaded on these AMI sets. You can choose to install most applications during the bootstrapping process and alter the installation based on configuration sets grouped by instance tags, Auto Scaling groups, or other instance artifacts. You can set a tag on your Nginx instances (such as Nginx-v-1.6.2). Your update process can query for the instance tag, validate whether it's the most current version of Nginx, and then proceed with the installation. When it's time to update the prebaked AMI, you can simply swap your existing AMI with the most recent version in the underlying deployment service and update the tag.

Deployment services like AWS CloudFormation and AWS OpsWorks are better suited for the prebaked AMI approach. You can also find multiple third-party tools for prebaking AMIs. Some well-known ones are packer.io[37] and aminator (built by Netflix).[38] You can also choose third-party tools for your configuration management such as Chef, Puppet, Salt, Ansible, and Capistrano.

In-place vs Disposable Method

The deployment services offer two methods to help you update your application stack, namely in-place and disposable. An in-place upgrade involves performing application updates on live Amazon EC2 instances. A disposable upgrade, on the other hand, involves rolling out a new set of EC2 instances by terminating older instances.

An in-place upgrade is typically useful in a rapid deployment with a consistent rollout schedule. It is designed for sessionless applications. You can still use the in-place upgrade method for stateful applications by implementing a rolling deployment schedule and by following the guidelines mentioned in the section on blue-green deployments.

In contrast, disposable upgrades offer a simpler way to know if your application has unknown dependencies. The underlying EC2 instance usage is considered temporary or ephemeral in nature for the period of deployment until the current release is active. During the new release, a new set of EC2 instances are rolled out by terminating older instances. This type of upgrade technique is more common in an immutable infrastructure.

Two services are especially useful for an in-place upgrade: You can use AWS CodeDeploy to manage the updates while managing application deployment using the building blocks (Amazon EC2, Amazon EBS, Amazon S3, Amazon RDS) individually or third-party managed build systems like Github, Jenkins, Travis CI, or Circle CI. Alternatively, you can use AWS OpsWorks to manage both your application deployment as well as updates.

For disposable upgrades, you can set up a cloned environment with the deployment services (AWS Elastic Beanstalk, AWS CloudFormation, and AWS OpsWorks) or use them in combination with an Auto Scaling configuration to manage the updates.

In-place Upgrade Method

AWS CodeDeploy is a tool focused on software deployment. You can deploy applications from Amazon S3 and GitHub repositories using this tool. Once you prepare deployment content and the underlying Amazon EC2 instances, you can deploy an application and its revisions on a consistent basis. You can push the updates to a set of instances called deployment groups that are made of tagged EC2 instances[39] and/or Auto Scaling groups.[40] In addition, AWS CodeDeploy works with various configuration management tools, continuous integration and deployment systems, and source control systems. You can find complete list of product integration options in the AWS CodeDeploy documentation.[41]

Another service to use for managing the entire lifecycle of an application is AWS OpsWorks. You can use built-in layers or deploy custom layers and recipes to launch your application stack. In addition, tons of customization options are available for configuration and pushing application updates. For more information, read the whitepaper on Managing Multi-Tiered Web Application with OpsWorks for reviewing strategies to update OpsWorks stacks.[42]

Disposable Upgrade Method

You can perform disposable upgrades in a couple of ways. You can use an Auto Scaling policy to define how you want to add (scale out) or remove (scale in) instances.[43] By coupling this with your update strategy, you can control rolling out of an application update as part of the scaling event.

For example, you can update Auto Scaling to use the new AMI and configure a termination policy to use *OldestInstance* during a scale in event. Or you could use *OldestLaunchConfiguration* to phase out all instances that use the previous configuration. If you are using an Elastic Load Balancing (ELB), you can attach an additional Auto Scaling configuration behind the ELB and use a similar approach to phase in newer instances while removing older instances.

Similarly, you can configure rolling deployments in conjunction with deployment services such as AWS Elastic Beanstalk[44] and AWS CloudFormation.[45] You can use update policies to describe how instances in an Auto Scaling group are replaced or modified as part of your update strategy. You can control the number of instances to get updated concurrently or in batches. You can choose to apply the updates to certain instances while isolating in-service instances. You can also specify the time to wait between batched updates. In addition, you can cancel or roll back an update if you discover a bug in your application code. These features can help increase the availability of your application during updates. See the next section on blue-green deployments to address some concerns related to managing updates for sessionful applications using Auto Scaling.

Blue-Green Method

Blue-green is a method in which you have two identical stacks of your application running in their own environments. You use various strategies to migrate the traffic from your current application stack (blue) to a new version of the application (green). This is a popular technique for deploying applications with zero downtime. The deployment services like AWS Elastic Beanstalk, AWS CloudFormation, or AWS OpsWorks are particularly useful as they provide a simple way to clone your running application stack. You can set up a new version of your application (green) by simply cloning current version of the application (blue).

For a sessionless web application, the update process is pretty straightforward. Simply upload the new version of your application and let your deployment service (AWS Elastic Beanstalk, AWS CloudFormation, or AWS OpsWorks) deploy a new version (green). To cut over to the new version, you simply replace the ELB URLs in your DNS records. Elastic Beanstalk has a **Swap Environment URLs** feature to facilitate a simpler cutover process. If you use Amazon Route 53 to manage your DNS records, you need to swap ELB endpoints for AWS CloudFormation or AWS OpsWorks deployment services.[46]

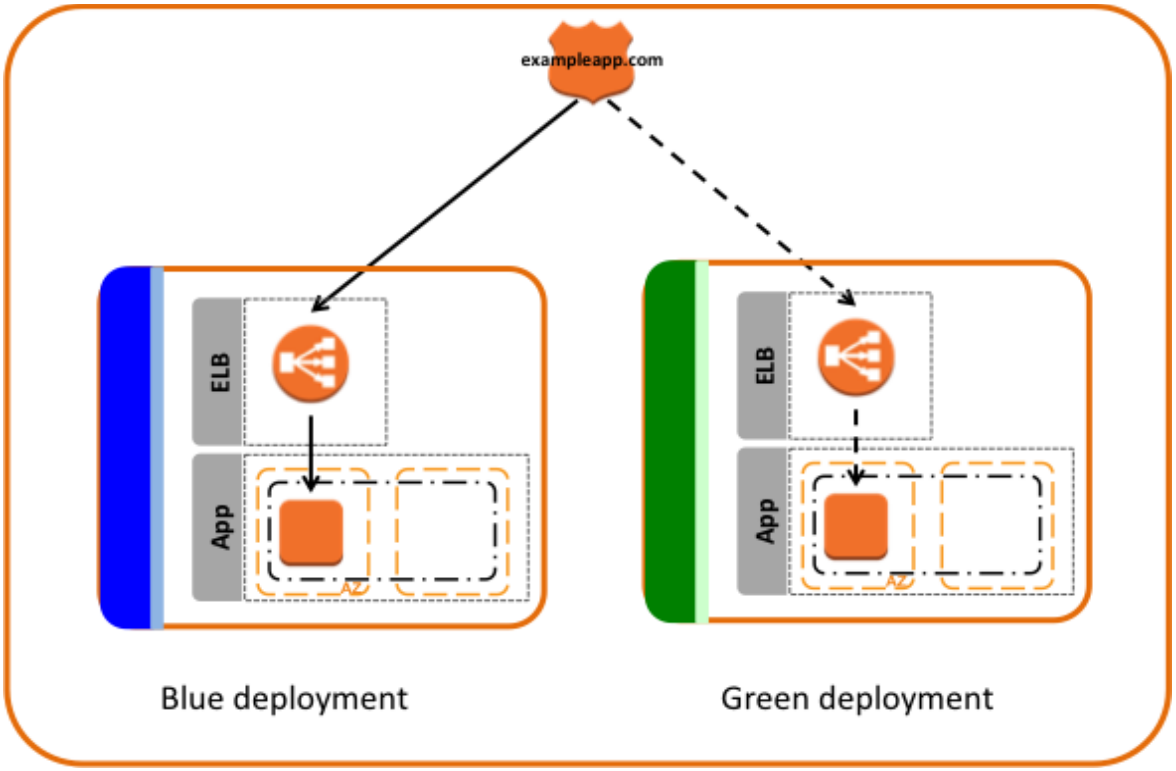

Figure 2: Blue-Green Deployment

For applications with session states, the cutover process can be complex. When you perform an update, you don't want your end users to experience downtime or lose data. You should consider storing the sessions outside of your deployment service because with certain deployment service creating a new stack will recreate the session database. In particular, consider storing the sessions separately from your deployment service if you are using Amazon RDS database or Amazon ElastiCache.[47]

Read additional recommendations for achieving zero downtime with Elastic Beanstalk during your application upgrade.[48] Similarly, review the recommendation for updating AWS CloudFormation stacks while preventing updates to stack resources.[49] In addition, consider monitoring your instances in the blue deployment and ELB's connection draining before terminating instances.[50]

If you use Amazon Route53 to host your DNS records, you can consider using the Weighted Round Robin (WRR) feature for migrating from blue to green deployments. The feature helps to drive the traffic gradually rather than instantly.[51] If your application has a bug, this method helps ensure the blast radius is minimal as it only affects small number of users. This method also simplifies rollbacks if they become necessary. In addition, you only use the required number of instances while you scale up in the green and scale down in the blue deployment. For example, you can set WRR to allow 10% of the traffic to go to green deployment while keeping 90% of traffic on blue. You gradually increase the percentage of green instances until you achieve a full cutover. Keeping the DNS cache to a shorter TTL on the client side also ensures the client will connect to green deployment with rapid release cycle thus minimizing bad DNS caching behavior.

Hybrid Deployment Model Approach

You can also use the deployment services in a hybrid fashion for managing your application fleet. For example, you can combine the simplicity of managing AWS infrastructure provided by Elastic Beanstalk and the automation of custom network segmentation provided by AWS CloudFormation. Leveraging a hybrid deployment model also simplifies your architecture as it decouples your deployment method so that you can choose different strategies for updating your application stack.

A few example scenarios are provided below. These are not exhaustive; they are meant to give you an idea of hybrid deployment approaches that you can plan for.

Scenario 1: Use AWS CloudFormation to deploy an Elastic Beanstalk application along with an AWS service integration such as DynamoDB, Amazon RDS, and Amazon S3.

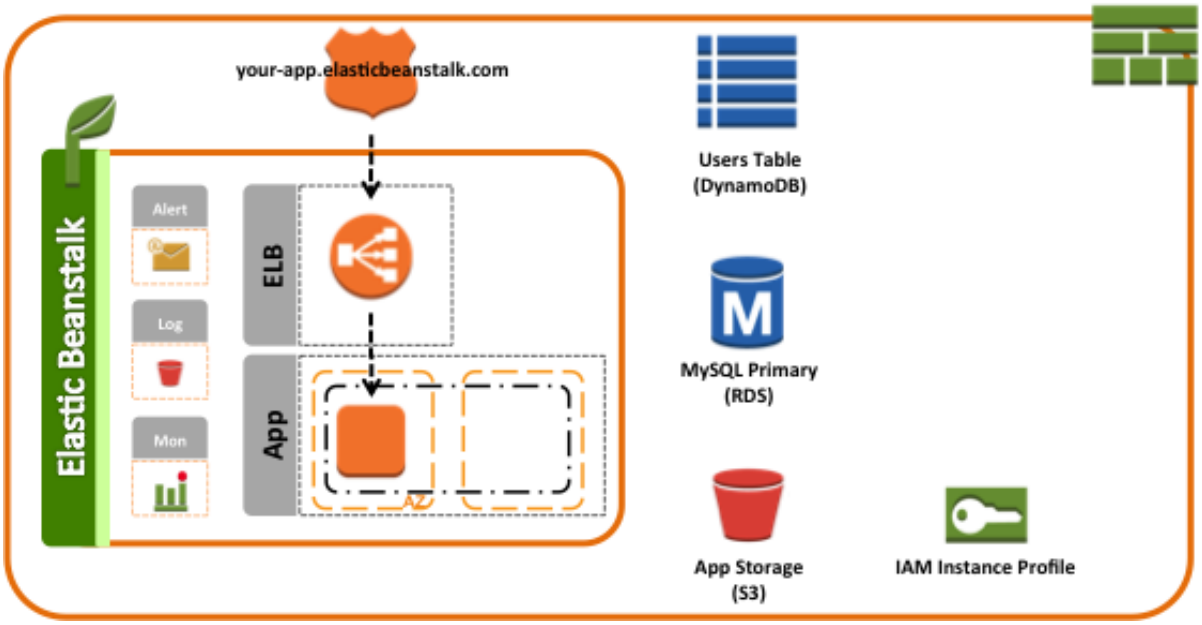

Figure 3: Reference Architecture for Scenario 1

Scenario 2: Use AWS CloudFormation to deploy similar application stacks in AWS OpsWorks and manage the entire infrastructure using AWS CloudFormation.

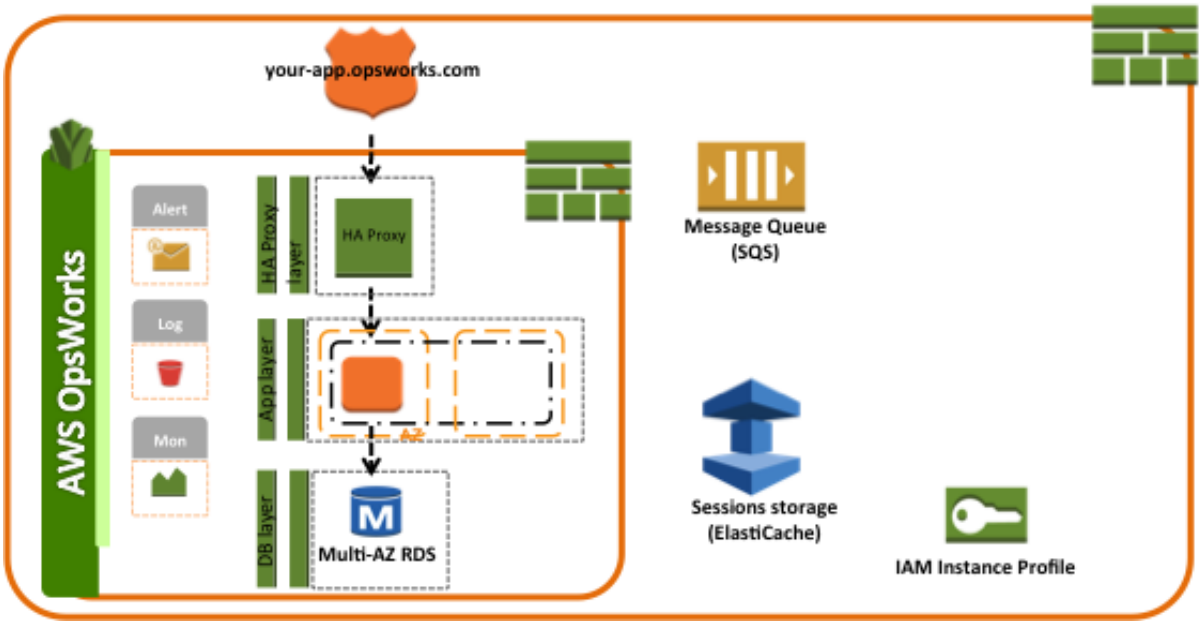

Figure 4: Reference Architecture for Scenario 2

Scenario 3: Use AWS CloudFormation to deploy multiple application stacks that you manage with Elastic Beanstalk and AWS OpsWorks.

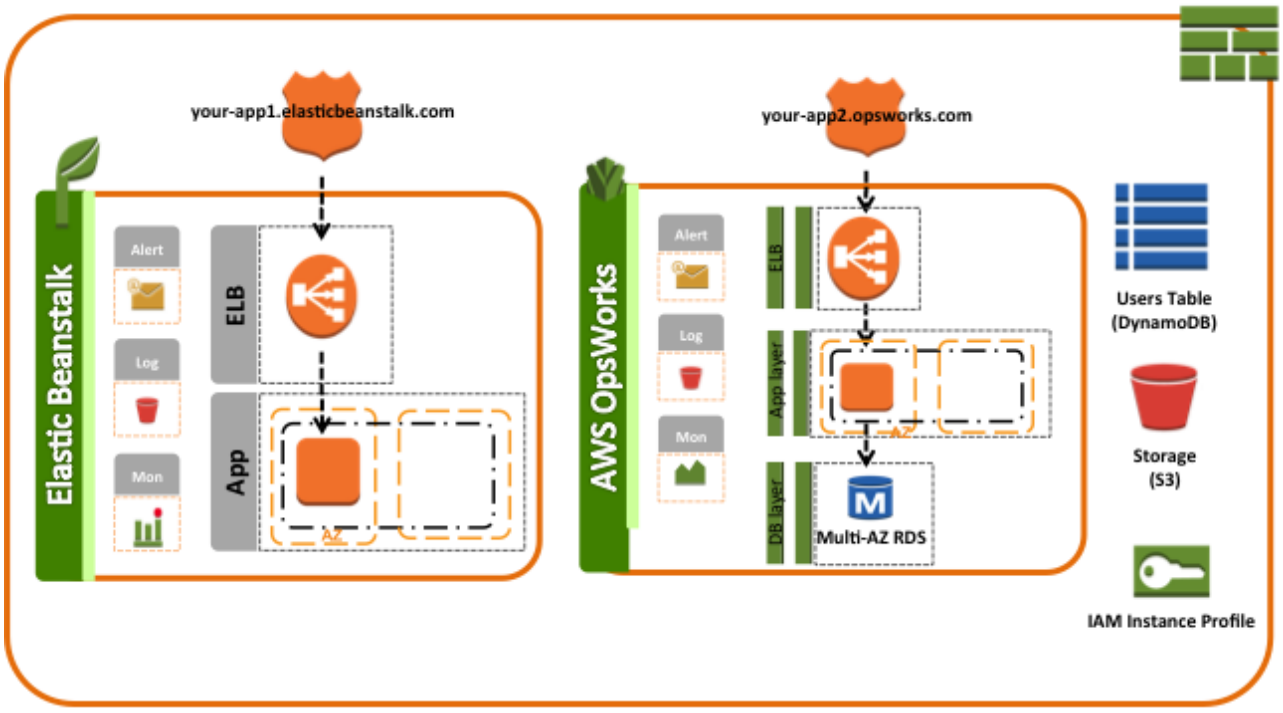

Figure 5: Reference Architecture for Scenario 3

Scenario 4: Use AWS CodeDeploy to deploy and manage multiple applications while provisioning the infrastructure using Amazon EC2 and AWS CloudFormation.

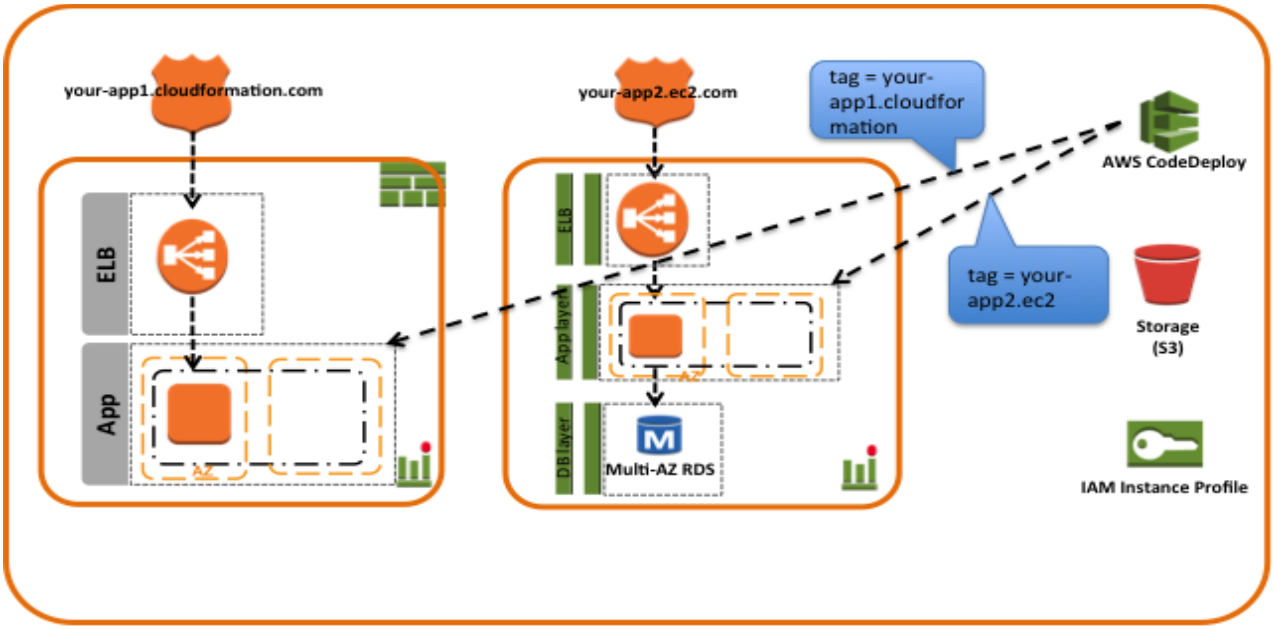

Figure 6: Reference Architecture for Scenario 4

Conclusion

Amazon Web Services provides number of tools to simplify and automate the provisioning of infrastructure and deployment of applications. Each deployment service has a unique approach for managing application deployments and offers various strategies for updating your application. For best results, focus on your workload and choose a deployment service that is tailored to your specific needs. As you plan, consider using hybrid deployment model approach, which uses a combination of deployment services for managing multiple applications throughout their lifecycle.

Notes

[1] http://aws.amazon.com/ec2/

[2] http://aws.amazon.com/ebs/

[3] http://aws.amazon.com/s3/

[4] http://aws.amazon.com/glacier/

[5] http://aws.amazon.com/rds/

[6] http://aws.amazon.com/redshift/

[7] http://aws.amazon.com/dynamodb/

[8] http://aws.amazon.com/kinesis/

[9] http://aws.amazon.com/elasticbeanstalk/

[10] https://www.docker.com/

[11] http://aws.amazon.com/autoscaling/

[12] http://aws.amazon.com/elasticloadbalancing/

[13] http://aws.amazon.com/cloudformation/

[14] http://aws.amazon.com/codedeploy/

[15] http://aws.amazon.com/opsworks/

[16] http://aws.amazon.com/codecommit/

[17] http://aws.amazon.com/codepipeline/

[18] http://aws.amazon.com/codedeploy/

[19] http://aws.amazon.com/ecs/

[20] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html

[21] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html

[22] http://aws.amazon.com/vpc/

[23] http://aws.amazon.com/cloudwatch/

[24] https://console.aws.amazon.com/

[25] http://aws.amazon.com/cli/

[26] http://aws.amazon.com/tools/

[27] http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/WhatIsCloudWatchLogs.html

[28] http://aws.amazon.com/cloudtrail/

[29] http://docs.aws.amazon.com/IAM/latest/UserGuide/instance-profiles.html

[30] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html

[31] http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customize-containers.html

[32] http://docs.aws.amazon.com/opsworks/latest/userguide/other-services.html

[33] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/CHAP_TemplateQuickRef.html

[34] http://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/allocation.html

[35] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html

[36] https://s3.amazonaws.com/cloudformation-examples/BoostrappingApplicationsWithAWSCloudFormation.pdf

[37] https://www.packer.io/

[38] https://github.com/Netflix/aminator

[39] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html#Using_Tags_Console

[40] http://docs.aws.amazon.com/codedeploy/latest/userguide/auto-scaling-integ.html

[41] http://aws.amazon.com/codedeploy/product-integrations/

[42] http://do.awsstatic.com/whitepapers/managing-multi-tiered-web-applications-with-opsworks.pdf

[43] http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/AutoScalingBehavior.InstanceTermination.html

[44] http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.rollingupdates.html

[45] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-attribute-updatepolicy.html

[46] http://aws.amazon.com/route53/

[47] http://aws.amazon.com/elasticache/

[48] http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.CNAMESwap.html

[49] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/protect-stack-resources.html

[50] http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/config-conn-drain.html

[51] http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-policy.html#routing-policy-weighted