Blue/Green Deployments on AWS

August 2016

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

Abstract

Blue/green deployment is a technique for releasing applications by shifting traffic between two identical environments running different versions of the application. Blue/green deployments can mitigate common risks associated with deploying software, such as downtime and rollback capability. This paper provides an overview of the blue/green deployment methodology and describes techniques customers can implement using Amazon Web Services (AWS) services and tools. This paper also addresses considerations around the data tier, which is an important component of most applications.

Introduction

In a traditional approach to application deployment, you typically fix a failed deployment by redeploying an older, stable version of the application. Redeployment in traditional data centers is typically done on the same set of resources due to the cost and effort of provisioning additional resources. Although this approach works, it has many shortcomings. Rollback isn't easy because it's implemented by redeployment of an older version from scratch. This process takes time, making the application potentially unavailable for long periods. Even in situations where the application is only impaired, a rollback is required, which overwrites the faulty version. As a result, you have no opportunity to debug the faulty application in place.

Applying the principles of agility, scalability, utility consumption, and automation capabilities of the AWS platform can shift the paradigm of application deployment. This enables a better deployment technique called *blue/green deployment*.

Blue/Green Deployment Methodology

Blue/green deployments provide near zero-downtime release and rollback capabilities. The fundamental idea behind blue/green deployment is to shift traffic between two identical environments that are running different versions of your application. The blue environment represents the current application version serving production traffic. In parallel, the green environment is staged running a different version of your application. After the green environment is ready and tested, production traffic is redirected from blue to green. If any

problems are identified, you can roll back by reverting traffic back to the blue environment.

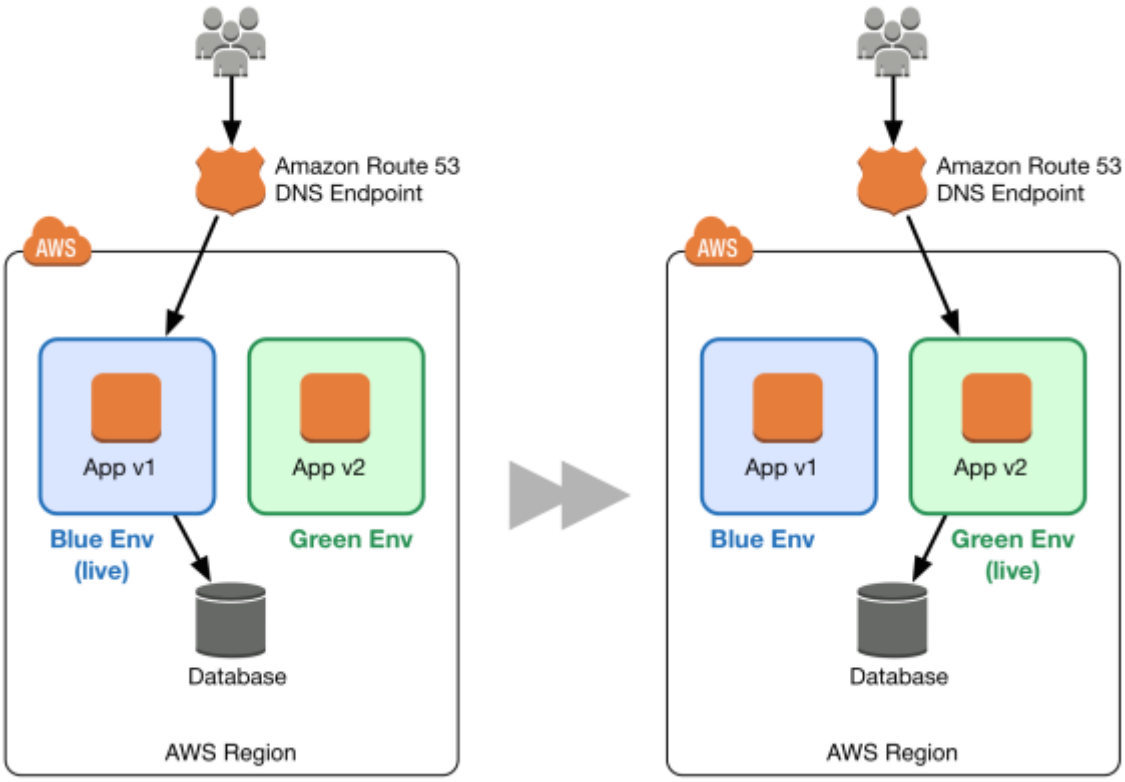

Figure 1: Basic blue/green example

Although blue/green deployment isn't a new concept, you don't commonly see it used in traditional, on-premises hosted environments due to the cost and effort required to provision additional resources. The advent of cloud computing dramatically changes how easy and cost-effective it is to adopt the blue/green approach to deploying software.

Benefits of Blue/Green

Traditionally, with in-place upgrades, it was difficult to validate your new application version in a production deployment while also continuing to run your old version of the application. Blue/green deployments provide a level of isolation between your blue and green application environments. It ensures spinning up a parallel green environment does not affect resources underpinning your blue environment. This isolation reduces your deployment risk.

After you deploy the green environment, you have the opportunity to validate it. You might do that with test traffic before sending production traffic to the green environment, or by using a very small fraction of production traffic, to better reflect real user traffic. This is called *canary analysis* or *canary testing*. If you discover the green environment is not operating as expected, there is no impact on the blue environment. You can route traffic back to it, minimizing impaired operation or downtime, and limiting the blast radius of impact.

This ability to simply roll traffic back to the still-operating blue environment is a key benefit of blue/green deployments. You can roll back to the blue environment at any time during the deployment process. Impaired operation or downtime is minimized because impact is limited to the window of time between green environment issue detection and shift of traffic back to the blue environment. Furthermore, impact is limited to the portion of traffic going to the green environment, not all traffic. If the blast radius of deployment errors is reduced, so is the overall deployment risk.

Blue/green deployments also fit well with continuous integration and continuous deployment (CI/CD) workflows, in many cases limiting their complexity. Your deployment automation would have to consider fewer dependencies on an existing environment, state, or configuration. Your new green environment gets launched onto an entirely new set of resources.

In AWS, blue/green deployments also provide cost optimization benefits. You're not tied to the same underlying resources. So if the performance envelope of the application changes from one version to another, you simply launch the new environment with optimized resources, whether that means fewer resources or just different compute resources. You also don't have to run an overprovisioned architecture for an extended period of time. During the deployment, you can scale out the green environment as more traffic gets sent to it and scale the blue environment back in as it receives less traffic. Once the deployment succeeds, you decommission the blue environment and stop paying for the resources it was using.

Define the Environment Boundary

When planning for blue/green deployments, you have to think about your environment boundary—where have things changed and what needs to be

deployed to make those changes live. The scope of your environment is influenced by a number of factors, as described in the following table.

Factors	Criteria
Application architecture	Dependencies, loosely/tightly coupled
Organizational	Speed and number of iterations
Risk and complexity	Blast radius and impact of failed deployment
People	Expertise of teams
Process	Testing/QA, rollback capability
Cost	Operating budgets, additional resources

Table 1: Factors that affect environment boundary

For example, organizations operating applications that are based on the microservices architecture pattern could have smaller environment boundaries because of the loose coupling and well-defined interfaces between the individual services. Organizations running legacy, monolithic apps can still leverage blue/green deployments, but the environment scope can be wider and the testing more extensive. Regardless of the environment boundary, you should leverage automation wherever you can to streamline the process, reduce human error, and control your costs.

AWS Tools and Services Enabling Blue/Green Deployments

AWS provides a number of tools and services to help you automate and streamline your deployments and infrastructure through the AWS API, which you can leverage using the web console, CLI tools, SDKs, and IDEs.[1] Because many services are available in the AWS ecosystem, the following is not a complete list. Instead, this list provides an overview of only the services we discuss in this paper. You may find software solutions outside of AWS to help automate and monitor your infrastructure and deployment, but this paper focuses on AWS services.

Amazon Route 53

Amazon Route 53 is a highly available and scalable authoritative DNS service that routes user requests for Internet-based resources to the appropriate destination.[2] Amazon Route 53 runs on a global network of DNS servers providing customers with added features, such as routing based on health checks, geography, and latency. DNS is a classic approach to blue/green deployments, allowing administrators to direct traffic by simply updating DNS records in the hosted zone. Also, time to live (TTL) can be adjusted for resource records; this is important for an effective DNS pattern because a shorter TTL allows record changes to propagate faster to clients.

Elastic Load Balancing

Another common approach to routing traffic for a blue/green deployment is through the use of load balancing technologies. Elastic Load Balancing distributes incoming application traffic across designated Amazon Elastic Compute Cloud (Amazon EC2) instances.[3] Elastic Load Balancing scales in response to incoming requests, performs health checking against Amazon EC2 resources, and naturally integrates with other AWS tools, such as Auto Scaling. This makes it a great option for customers who want to increase application fault tolerance.

Auto Scaling

Auto Scaling helps maintain application availability and lets customers scale EC2 capacity up or down automatically according to defined conditions.[4] The templates used to launch EC2 instances in an Auto Scaling group are called *launch configurations*. You can attach different versions of launch configuration to an Auto Scaling group to enable blue/green deployment. You can also configure Auto Scaling for use with an Elastic Load Balancing load balancer. In this configuration, Elastic Load Balancing balances the traffic across the EC2 instances running in an Auto Scaling group. You define termination policies in Auto Scaling groups to determine which EC2 instances to remove during a scaling action. As explained in the Auto Scaling Developer Guide, Auto Scaling also allows instances to be placed in Standby state, instead of termination, which helps with quick rollback when required.[5] Both Auto Scaling's termination policies and Standby state enable blue/green deployment.

AWS Elastic Beanstalk

AWS Elastic Beanstalk is a fast and simple way to get an application up and running on AWS.[6] It's perfect for developers who want to deploy code without worrying about managing the underlying infrastructure. Elastic Beanstalk supports Auto Scaling and Elastic Load Balancing, both of which enable blue/green deployment. Elastic Beanstalk makes it easy to run multiple versions of your application and provides capabilities to swap the environment URLs, facilitating blue/green deployment.

AWS OpsWorks

AWS OpsWorks is a configuration management service based on Chef that allows customers to deploy and manage application stacks on AWS.[7] Customers can specify resource and application configuration, and deploy and monitor running resources. OpsWorks simplifies cloning entire stacks when you're preparing blue/green environments.

AWS CloudFormation

AWS CloudFormation provides customers with the ability to describe the AWS resources they need through JSON formatted templates.[8] This service provides very powerful automation capabilities for provisioning blue/green environments and facilitating updates to switch traffic, whether through Route 53 DNS, Elastic Load Balancing, etc. The service can be used as part of a larger infrastructure as code strategy, where infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration, in a manner similar to how application code is treated.

Amazon CloudWatch

Amazon CloudWatch is a monitoring service for AWS Cloud resources and the applications you run on AWS.[9] CloudWatch can collect and track metrics, collect and monitor log files, and set alarms. It provides system-wide visibility into resource utilization, application performance, and operational health, which are key to early detection of application health in blue/green deployments.

Techniques

The following techniques are examples of how you can implement blue/green on AWS. While we highlight specific services in each technique, you may have other services or tools to implement the same pattern. Choose the appropriate pattern based on the existing architecture, the nature of the application, and goals for software deployment in your organization. Experiment as much as possible to gain experience for your environment and understand how the different deployment risk factors interact with your specific workload.

Update DNS Routing with Amazon Route 53

DNS routing through record updates is a common approach to blue/green deployments. DNS is the mechanism for switching traffic from the blue environment to the green and vice versa, if rollback is necessary. This approach works with a wide variety of environment configurations, as long as you can express the endpoint into the environment as a DNS name or IP address.

In AWS, this technique applies to environments that are:

- Single instances, with a public or Elastic IP address
- Groups of instances behind an Elastic Load Balancing load balancer, or third-party load balancer
- Instances in an Auto Scaling group with an Elastic Load Balancing load balancer as the front end
- Services running on an Amazon EC2 Container Service (Amazon ECS) cluster fronted by an Elastic Load Balancing load balancer
- Elastic Beanstalk environment web tiers
- Other configurations that expose an IP or DNS endpoint

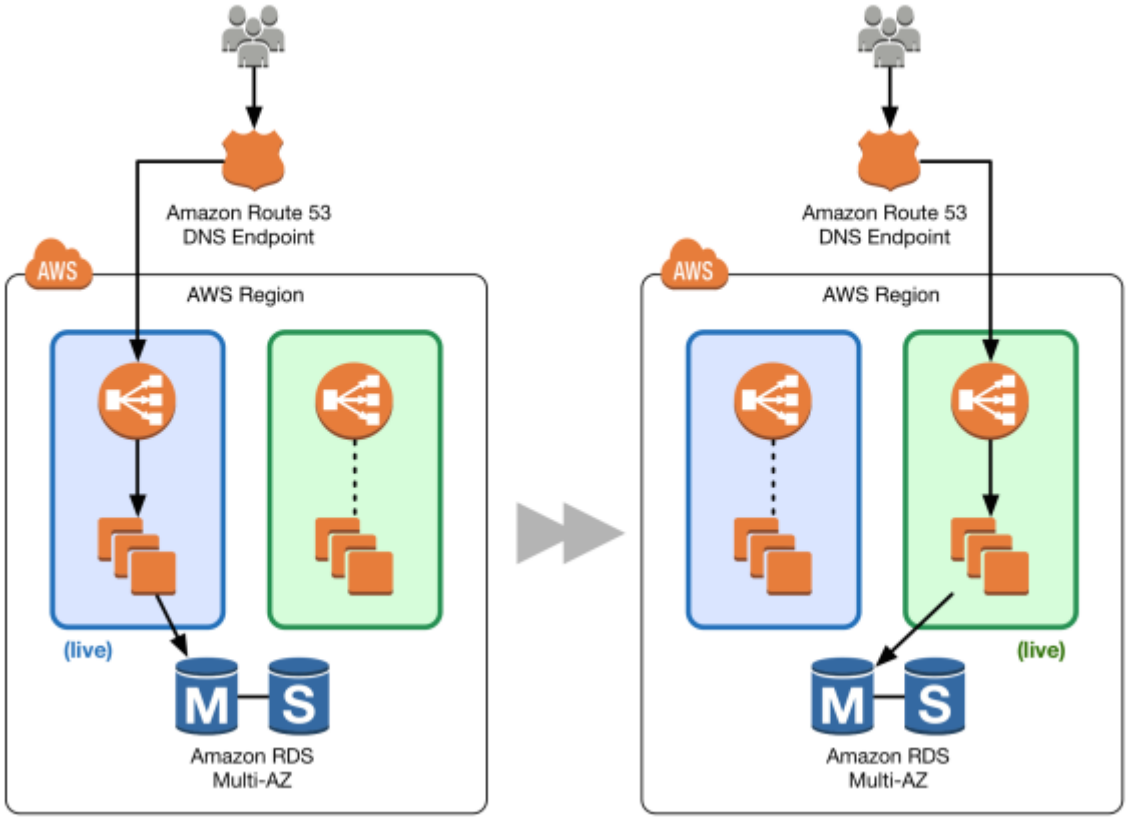

Figure 2: Classic DNS pattern

Figure 2 shows how Amazon Route 53 manages the DNS hosted zone. By updating the alias record,[10] you can route traffic from the blue environment to the green environment.

You can shift traffic all at once or you can do a weighted distribution. With Amazon Route 53, you can define a percentage of traffic to go to the green environment and gradually update the weights until the green environment carries the full production traffic. A weighted distribution provides the ability to perform canary analysis where a small percentage of production traffic is introduced to a new environment. You can test the new code and monitor for errors, limiting the blast radius if any issues are encountered. It also allows the green environment to scale out to support the full production load if you're using Elastic Load Balancing, for example. Elastic Load Balancing automatically scales its request-handling capacity to meet the inbound application traffic; the process of scaling isn't instant, so we recommend that you test, observe, and understand

your traffic patterns. Load balancers can also be pre-warmed (configured for optimum capacity) through a support request.[11]

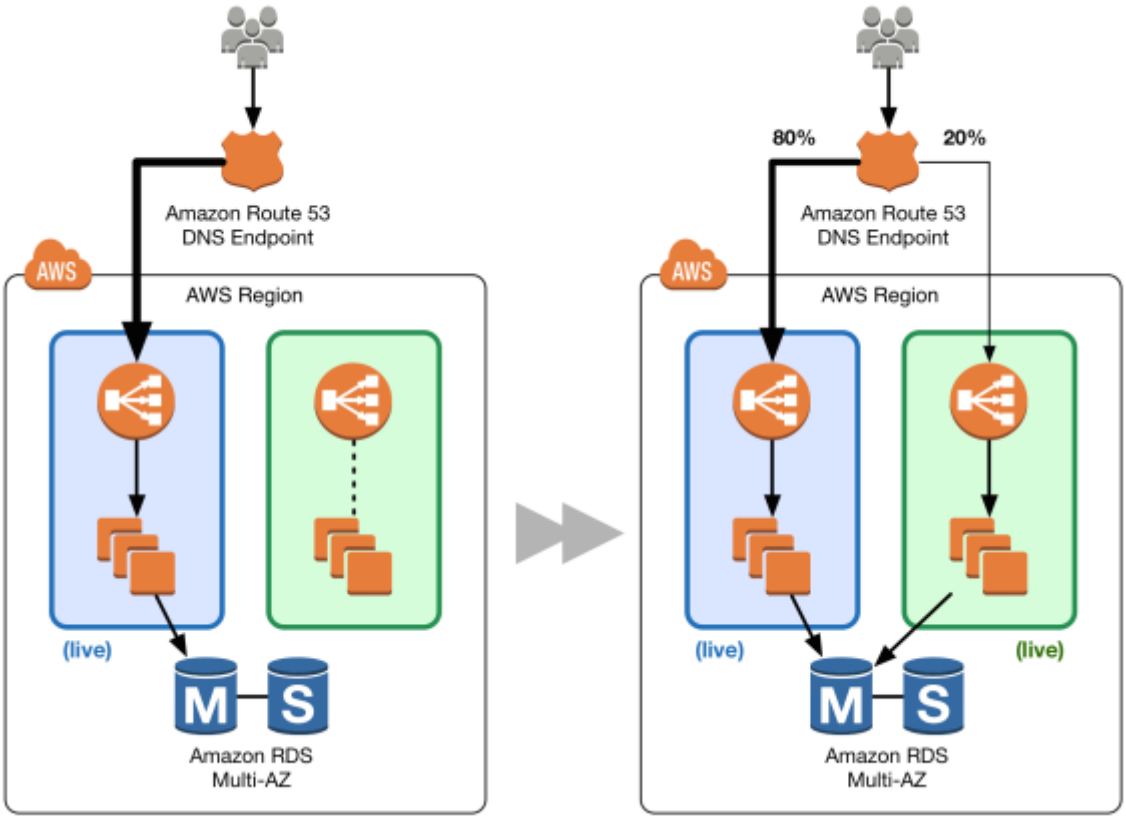

Figure 3: Classic DNS-weighted distribution

If issues arise during the deployment, you achieve rollback by updating the DNS record to shift traffic back to the blue environment. Although DNS routing is simple to implement for blue/green, the question is how quickly can you complete a rollback. DNS TTL determines how long clients cache query results. However, with older clients and potentially misbehaving clients in the wild, certain sessions may still be tied to the previous environment.

Although rollback can be challenging, this pattern certainly has the benefit of enabling a granular transition at your own pace to allow for more substantial testing and for scaling activities. To help manage costs, consider using Auto Scaling for the EC2 instances to scale out the resources based on actual demand. This works well with the gradual shift using Amazon Route 53 weighted distribution. For a full cutover, be sure to tune your Auto Scaling policy to scale

as expected and remember that the new Elastic Load Balancing endpoint may need time to scale up as well.

Swap the Auto Scaling Group Behind Elastic Load Balancer

If DNS complexities are prohibitive, consider using load balancing for traffic management to your blue and green environments. This technique uses Auto Scaling to manage the EC2 resources for your blue and green environments, scaling up or down based on actual demand. You can also control the Auto Scaling group size by updating your maximum desired instance counts for your particular group.

Auto Scaling also integrates with Elastic Load Balancing, so any new instances are automatically added to the load balancing pool if they pass the health checks governed by the load balancer. Elastic Load Balancing tests the health of your registered EC2 instances with a simple ping or a more sophisticated connection attempt or request. Health checks occur at configurable intervals and have defined thresholds to determine whether an instance is identified as healthy or unhealthy. For example, you could have an Elastic Load Balancing health check policy that pings port 80 every 20 seconds and, after passing a threshold of 10 successful pings, reports the instance as being InService. If enough ping requests time out, then the instance is reported to be OutofService. Used in concert with Auto Scaling, an instance that is OutofService could be replaced if the Auto Scaling policy dictates. Conversely, for scale-down activities, the load balancer removes the EC2 instance from the pool and drains current connections before they terminate.

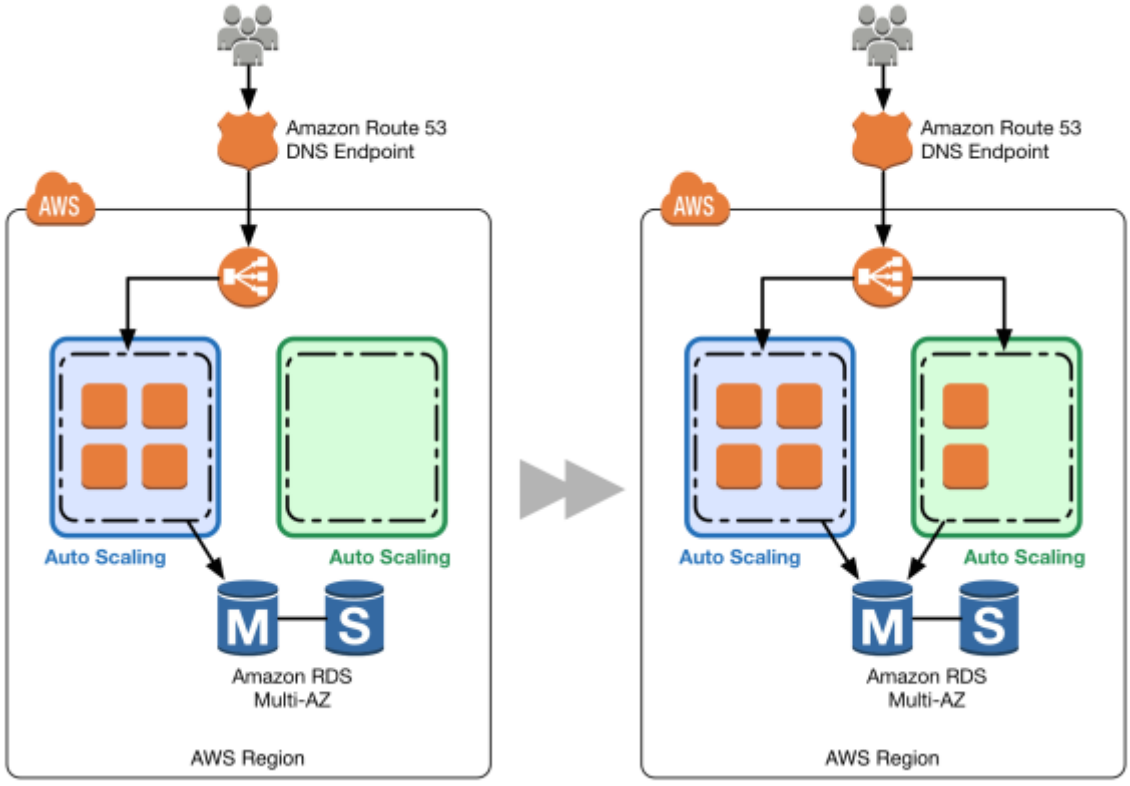

Figure 4: Swap Auto Scaling group pattern

Figure 4 shows the environment boundary reduced to the Auto Scaling group. A blue group carries the production load while a green group is staged and deployed with the new code. When it's time to deploy, you simply attach the green group to the existing load balancer to introduce traffic to the new environment. For HTTP/HTTPS listeners, the load balancer favors the green Auto Scaling group because it uses a least outstanding requests routing algorithm, as explained in the Elastic Load Balancing Developer Guide.[1][2] You can control how much traffic is introduced by adjusting the size of your green group up or down.

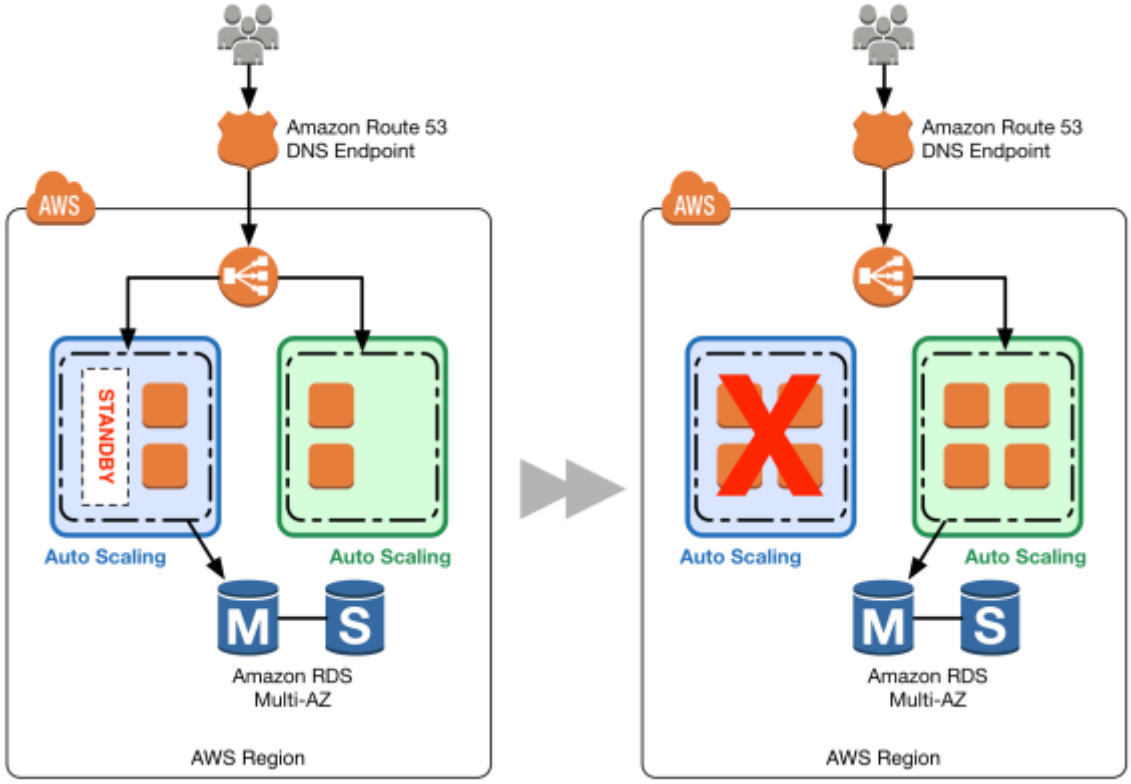

Figure 5: Blue Auto Scaling group nodes in standby and decommission

As you scale up the green Auto Scaling group, you can take blue Auto Scaling group instances out of service by either terminating them or putting them in Standby state, which is discussed in the Auto Scaling Developer Guide.[13] Standby is a good option because if you need to roll back to the blue environment, you only have to put your blue server instances back in service and they're ready to go.[14] As soon as the green group is scaled up without issues, you can decommission the blue group by adjusting the group size to zero. If you need to roll back, detach the load balancer from the green group or reduce the group size of the green group to zero.

This pattern's traffic management capabilities aren't as granular as the classic DNS, but you could still exercise control through the configuration of the Auto Scaling groups. For example, you could have a larger fleet of smaller instances with finer scaling policies, which would also help control costs of scaling. Because the complexities of DNS are removed, the traffic shift itself is more expedient. In

addition, with an already warm load balancer, you can be confident that you'll have the capacity to support production load.

Update Auto Scaling Group Launch Configurations

Auto Scaling groups have their own launch configurations. A launch configuration contains information like the Amazon Machine Image (AMI) ID, instance type, key pair, one or more security groups, and a block device mapping. You can associate only one launch configuration with an Auto Scaling group at a time, and it can't be modified after you create it. To change the launch configuration associated with an Auto Scaling group, replace the existing launch configuration with a new one. After a new launch configuration is in place, any new instances that are launched use the new launch configuration parameters, but existing instances are not affected. When Auto Scaling removes instances (referred to as *scaling in*) from the group, the default termination policy is to remove instances with the oldest launch configuration. However, you should know that if the Availability Zones were unbalanced to begin with, then Auto Scaling could remove an instance with a new launch configuration to balance the zones. In such situations, you should have processes in place to compensate for this effect.

To implement this technique, you start with an Auto Scaling group and Elastic Load Balancing load balancer. The current launch configuration has the blue environment.

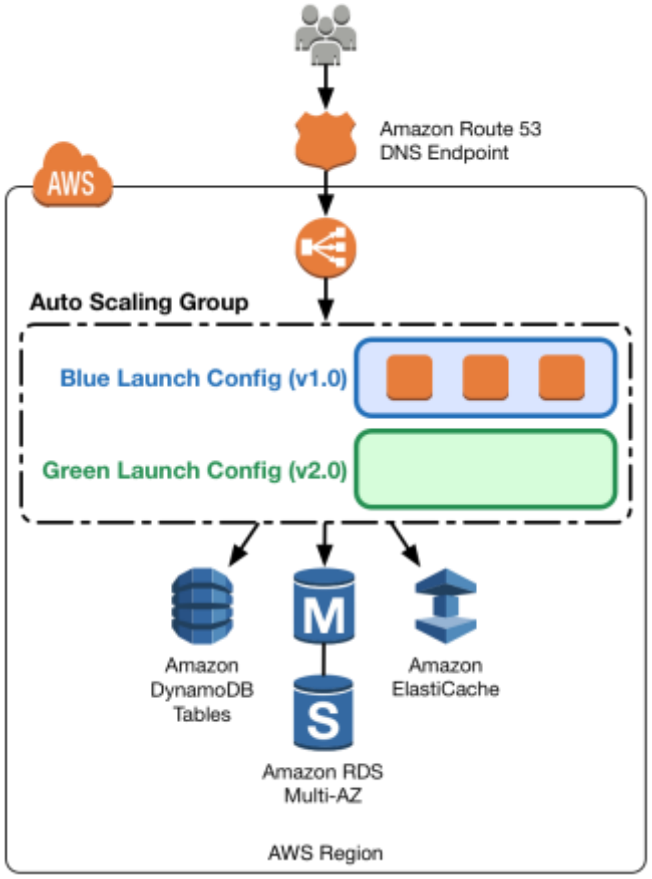

Figure 6: Launch configuration update pattern

To deploy the new version of the application in the green environment, update the Auto Scaling group with the new launch configuration, and then scale the Auto Scaling group to twice its original size.

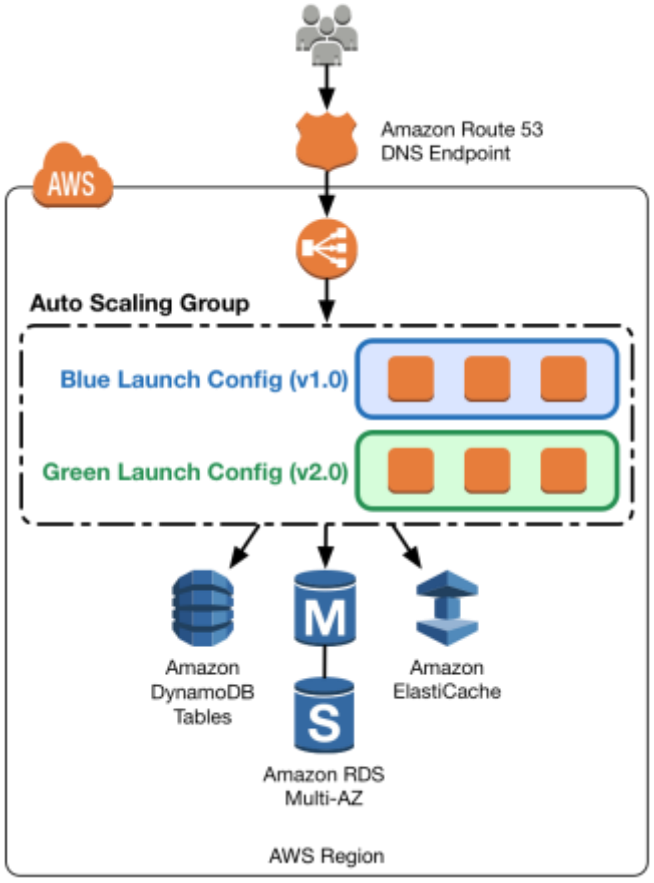

Figure 7: Scale up green launch configuration

Then, shrink the Auto Scaling group back to the original size. By default, instances with the old launch configuration are removed first. You can also leverage a group's Standby state to temporarily remove instances from an Auto Scaling group, as explained in the Auto Scaling Developer Guide.[15] Having the instance in Standby state helps in quick rollbacks, if required. As soon as you're confident about the newly deployed version of the application, you can permanently remove instances in Standby state.

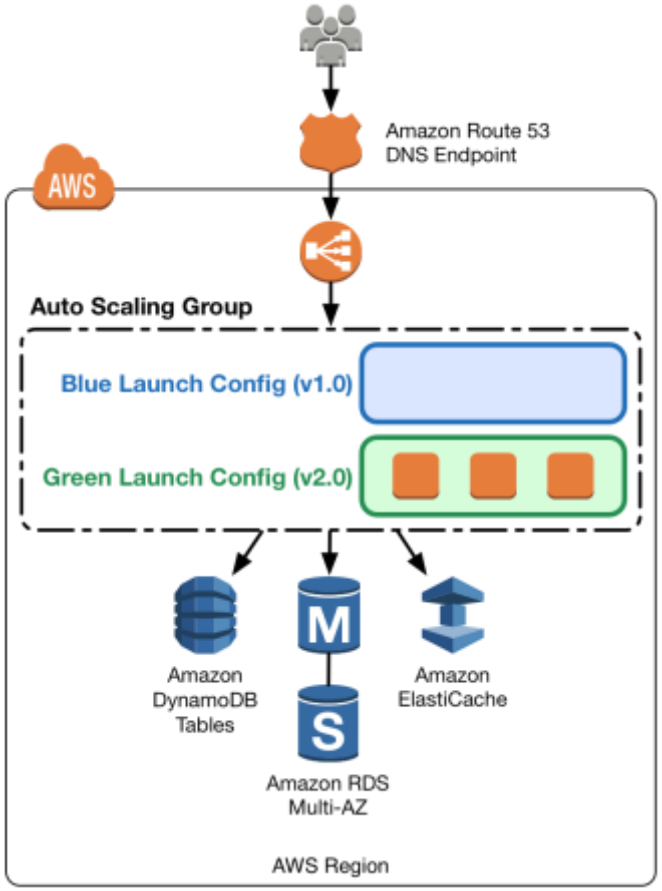

Figure 8: Scale down blue launch configuration

To perform a rollback, update the Auto Scaling group with the old launch configuration. Then, do the preceding steps in reverse. Or if the instances are in Standby state, bring them back online.

Swap the Environment of an Elastic Beanstalk Application

Elastic Beanstalk enables quick and easy deployment and management of applications without having to worry about the infrastructure that runs those applications. To deploy an application using Elastic Beanstalk, upload an application version in the form of an application bundle (for example, java `.war` file or `.zip` file), and then provide some information about your application. Based on application information, Elastic Beanstalk deploys the application in

the blue environment and provides a URL to access the environment (typically for web server environments).

Elastic Beanstalk provides several deployment policies that you can configure to use, ranging from policies that perform an in-place update on existing instances, to immutable deployment using a set of new instances. Because Elastic Beanstalk performs an in-place update when you update your application versions, your application may become unavailable to users for a short period of time.

However, you can avoid this downtime by deploying the new version to a separate environment. The existing environment's configuration is copied and used to launch the green environment with the new version of the application. The new—green—environment will have its own URL. When it's time to promote the green environment to serve production traffic, you can use Elastic Beanstalk's Swap Environment URLs feature, as explained in the AWS Elastic Beanstalk Developer Guide.[16]

To implement this technique, you would use Elastic Beanstalk to spin up the blue environment.

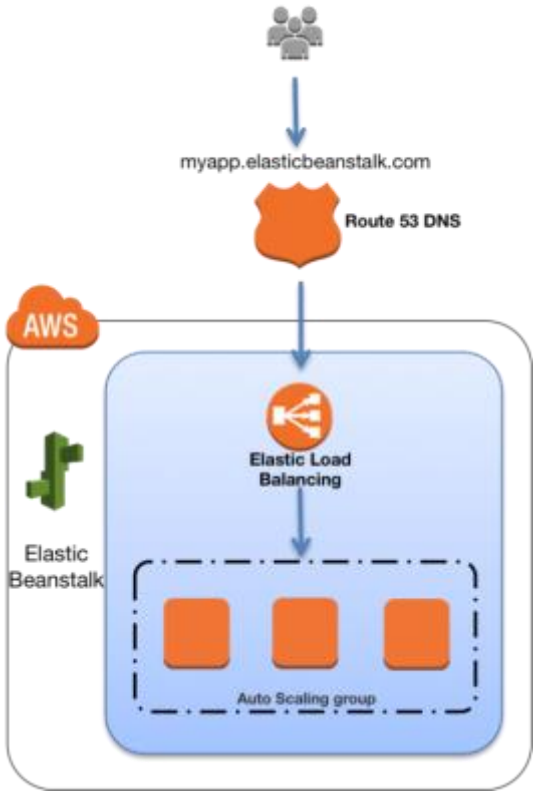

Figure 9: Elastic Beanstalk environment

Elastic Beanstalk provides an environment URL when the application is up and running. Then, the green environment is spun up with its own environment URL. At this time, two environments are up and running, but only the blue environment is serving production traffic.

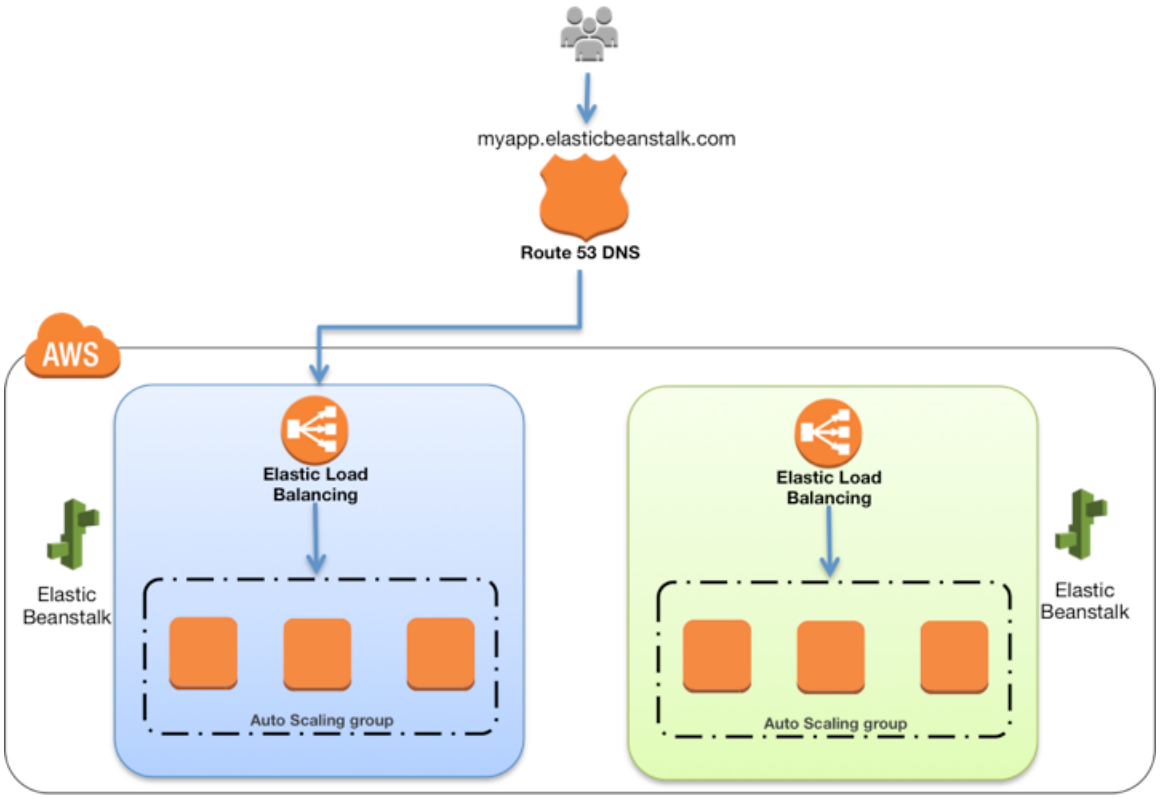

Figure 10: Prepare green Elastic Beanstalk environment

To promote the green environment to serve production traffic, you go to the environment's dashboard in the Elastic Beanstalk console and choose **Swap Environment URL** from the **Actions** menu. Elastic Beanstalk performs a DNS switch, which typically takes a few minutes. Refer to the technique Update DNS Routing with Amazon Route 53 for the factors to consider when performing a DNS switch. When the DNS changes have propagated, you can terminate the blue environment. To perform a rollback, invoke **Swap Environment URL** again.

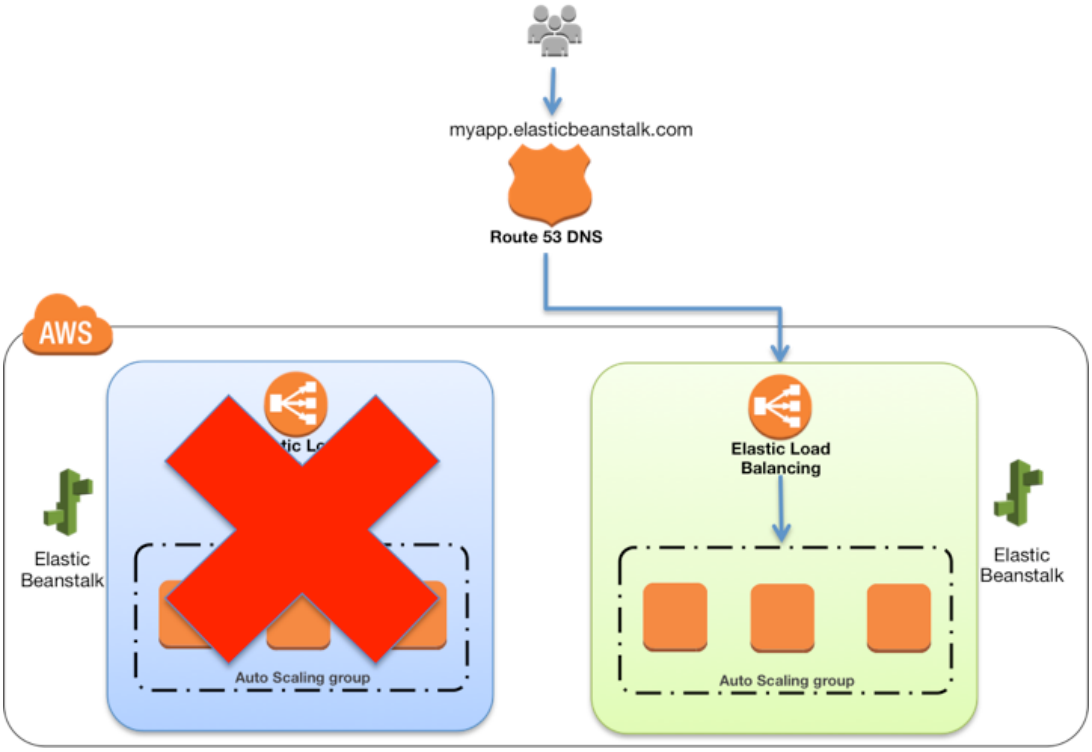

Figure 11: Decommission blue Elastic Beanstalk environment

Clone a Stack in AWS OpsWorks and Update DNS

AWS OpsWorks has the concept of stacks, which are logical groupings of AWS resources (EC2 instances, Amazon RDS, Elastic Load Balancing, and so on) that have a common purpose and should be logically managed together. Stacks are made of one or more layers. A layer represents a set of EC2 instances that serve a particular purpose, such as serving applications or hosting a database server. When a data store is part of the stack, you should be aware of certain data management challenges. We discuss those in depth in the next section.

To implement this technique in AWS OpsWorks, bring up the blue environment /stack with the current version of the application.[17]

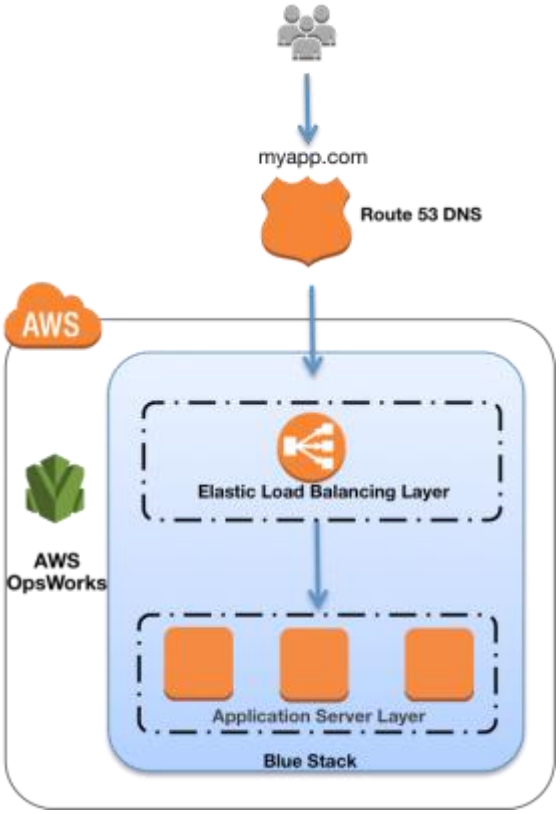

Figure 12: AWS OpsWorks stack

Next, create the green environment/stack with the newer version of application. At this point, the green environment is not receiving any traffic. If Elastic Load Balancing needs to be pre-warmed[18], you can do that at this time.

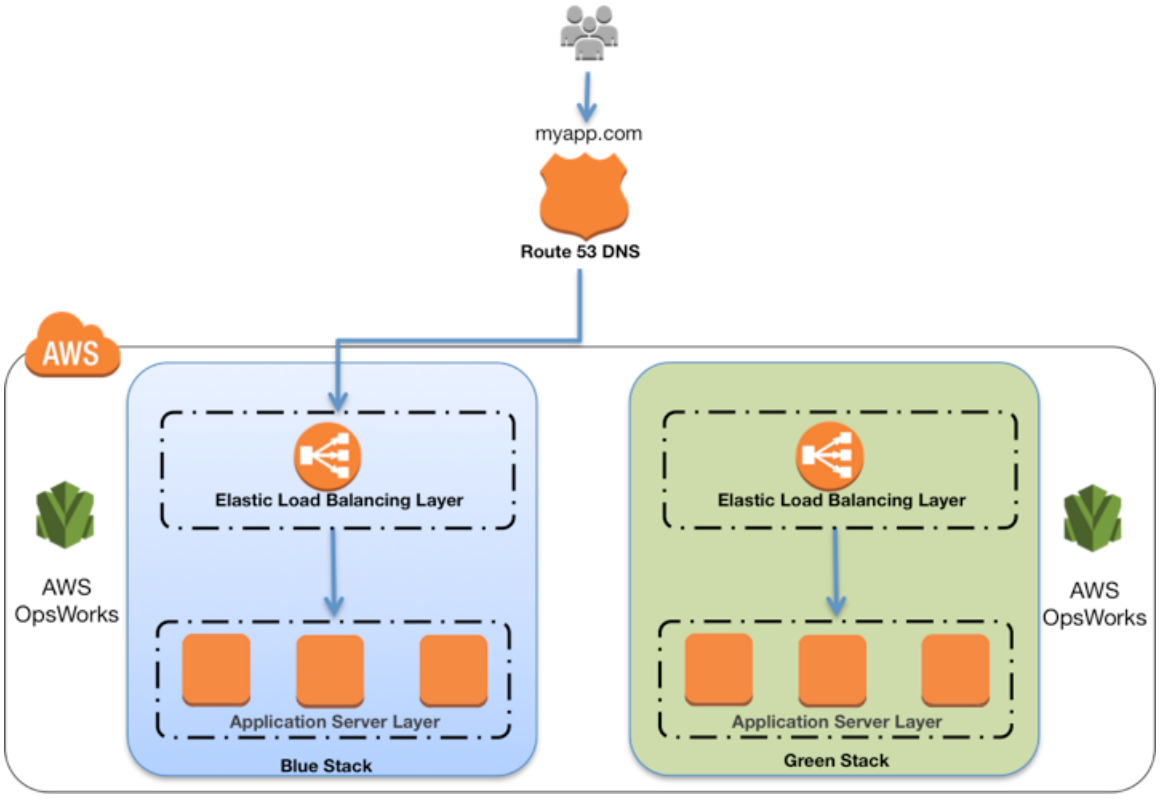

Figure 13: Clone stack to create green environment

When it's time to promote the green environment/stack into production, update DNS records to point to the green environment/stack's load balancer. You can also do this DNS flip gradually by using the Amazon Route 53 weighted routing policy. This technique involves updating DNS, so be aware of DNS issues discussed in the technique in [Update DNS Routing with Amazon Route 53].

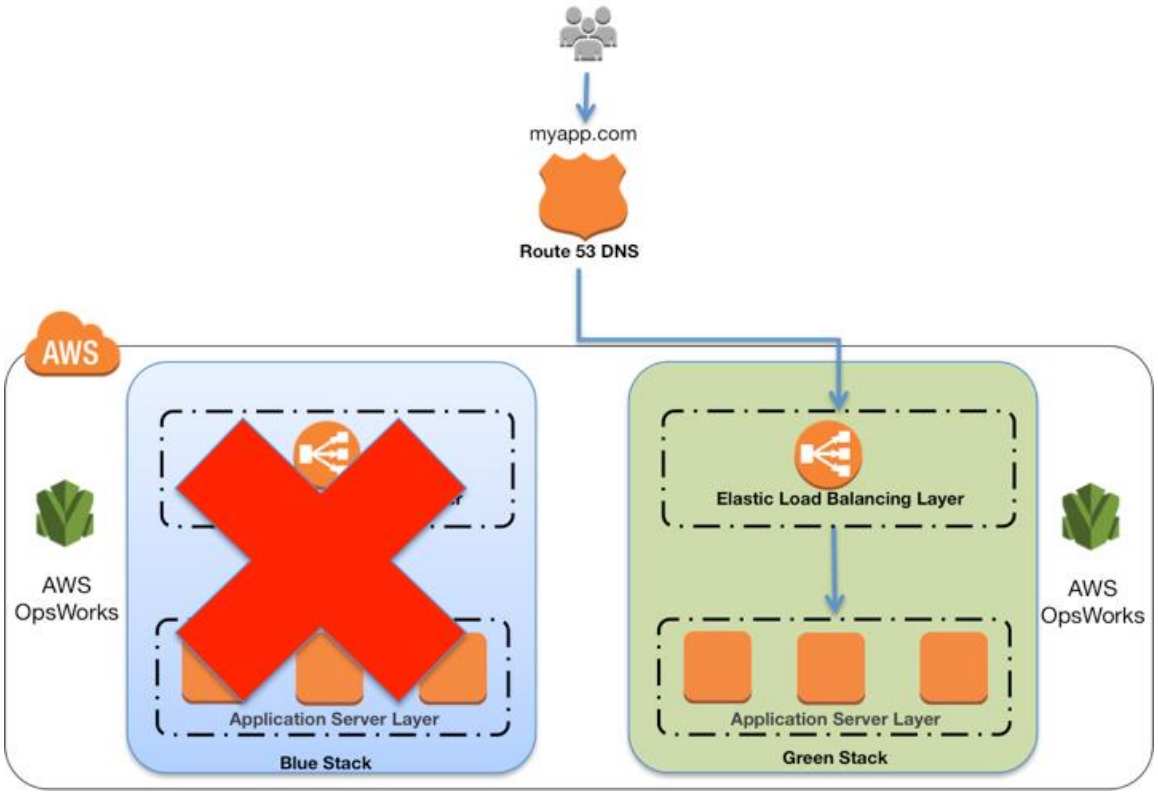

Figure 14: Decommission blue stack

Best Practices for Managing Data Synchronization and Schema Changes

Managing data synchronization across two distinct environments can be complex, depending on the number of data stores in use, the intricacy of the data model, and the data consistency requirements.

Both the blue and green environments need up-to-date data:

- The green environment needs up-to-date data access because it's becoming the new production environment.
- The blue environment needs up-to-date data in the event of a rollback, when production is then either shifted back or kept on the blue environment.

Broadly, you accomplish this by having both the green and blue environments share the same data stores. Unstructured data stores, such as Amazon Simple Storage Service (Amazon S3) object storage, NoSQL databases, and shared file systems are often easier to share between the two environments. Structured data stores, such as relational database management systems (RDBMS), where the data schema can diverge between the environments, typically require additional considerations.

Decoupling Schema Changes from Code Changes

A general recommendation is to decouple schema changes from the code changes. This way, the relational database is outside of the environment boundary defined for the blue/green deployment and shared between the blue and green environments. The two approaches for performing the schema changes are often used in tandem:

- The schema is **changed first**, before the blue/green code deployment. Database updates must be backward compatible, so the old version of the application can still interact with the data.
- The schema is **changed last**, after the blue/green code deployment. Code changes in the new version of the application must be backward compatible with the old schema.

Schema modifications in the former approach are often additive. You add fields to tables, new entities, and relationships. If needed, you can use triggers or asynchronous processes to populate these new constructs with data based on data changes performed by the old application version.

You need to follow coding best practices when developing applications to ensure your application can tolerate the presence of additional fields in existing tables, even if they are not used. When table row values are read and mapped into source code structures (objects, array hashes, etc.), your code should ignore fields it can't map instead of causing application runtime errors.

Schema modifications in the latter approach are often deletive. You remove unneeded fields, entities, and relationships, or merge and consolidate them. By this time, the old application version is no longer operational.

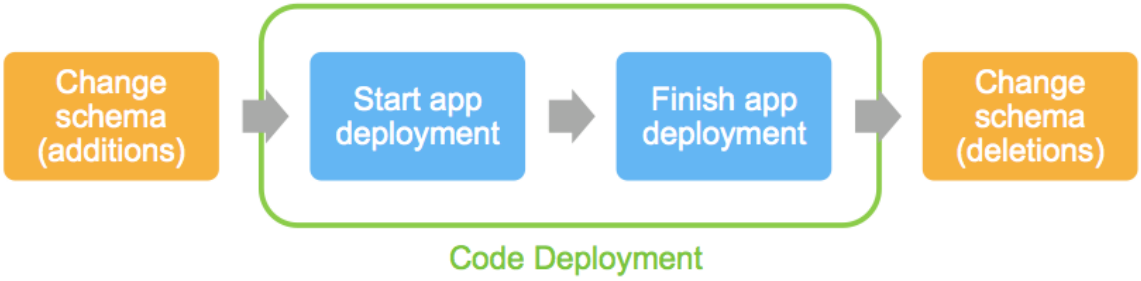

Figure 15: Decoupled schema and code changes

There's an increased risk involved when managing schema changes in this way: failures in the schema modification process can impact your production environment. Your additive changes can bring down the old application because of an undocumented issue where best practices weren't followed or where the new application version still has a dependency on a deleted field somewhere in the code.

To mitigate risk appropriately, this pattern places a heavy emphasis on your pre-deployment software lifecycle steps. Be sure to have a strong testing phase and framework and a strong QA phase. Performing the deployment in a testing environment can help identify these sorts of issues early, before the push to production.

When Blue/Green Deployments Are Not Recommended

As blue/green deployments become more popular, developers and companies are constantly applying the methodology to new and innovative use cases. However, there are some common use case patterns where applying this methodology, even if possible, isn't recommended.

These are cases where implementing blue/green deployment introduces too much risk, whether due to workarounds or additional "moving parts" in the deployment process. These complexities can introduce additional points of failure, or opportunities for the process to break down, that may negate any risk mitigation benefits blue/green deployments bring in the first place.

The following scenarios highlight patterns that may not be well suited for blue/green deployments.

Are your schema changes too complex to decouple from the code changes? Is sharing of data stores not feasible?

In some scenarios, sharing a data store isn't desired or feasible. Schema changes are too complex to decouple. Data locality introduces too much performance degradation to the application, as when the blue and green environments are in geographically disparate regions. All these situations require a solution where the data store is inside the deployment environment boundary and tightly coupled to the blue and green applications, respectively.

This requires data changes to be synchronized—propagated from the blue environment to the green one, and vice versa. The systems and processes to accomplish this are generally complex and limited by the data consistency requirements of your application. This means that during the deployment itself, you have to also manage the reliability, scalability and performance of that synchronization workload, adding risk to the deployment.

Does your application need to be "deployment aware"?

You have to use feature flags to control the behavior of the application during the blue/green deployment. This is often a consideration in conjunction with the inability to effectively decouple schema and code changes. Your application code would execute additional or alternate subroutines during the deployment, to keep data in sync, or perform other deployment-related duties. These routines are enabled and turned off, as the case may be, during the deployment by using configuration flags.

This practice also introduces additional risk and complexity and typically isn't recommended with blue/green deployments. The goal of blue/green deployments is to achieve immutable infrastructure, where you don't make changes to your application after it's deployed, but redeploy altogether. That way, you ensure the same code is operating in a production setting and in the deployment setting, reducing overall risk factors.

Does your commercial off-the-shelf (COTS) application come with a predefined update/upgrade process that isn't blue/green deployment friendly?

Many commercial software vendors provide their own update and upgrade process for their applications that they have tested and validated for distribution. While vendors are increasingly adopting the principles of immutable infrastructure and automated deployment, not all software products have those capabilities to date.

Working around the vendor's recommended update and deployment practices to try to implement or simulate a blue/green deployment process may also introduce unnecessary risk that can potentially negate the benefits of this methodology.

Conclusion

Application deployment has associated risks. But the advent of cloud computing, deployment and automation frameworks, and new deployment techniques, such as blue/green, help mitigate risks, such as human error, process, downtime, and rollback capability. The AWS utility billing model and breadth of automation tools make it much easier for customers to move fast and cost-effectively implement blue/green deployments at scale.

Contributors

The following individuals and organizations contributed to this document:

- George John, Solutions Architect, Amazon Web Services
- Andy Mui, Solutions Architect, Amazon Web Services
- Vlad Vlasceanu, Solutions Architect, Amazon Web Services

Appendix

Comparison of Blue Green Deployment Techniques

The following table offers an overview and comparison of the different blue/green deployment techniques discussed in this paper. The risk potential is evaluated from desirable lower risk (O) to less desirable higher risk (O O O).

Technique	Risk Category	Risk Potential	Reasoning
Update DNS Routing with Amazon Route 53	Application Issues	O	Facilitates canary analysis
	Application Performance	O	Gradual switch, traffic split management
	People/Process Errors	O O	Depends on automation framework, overall simple process
	Infrastructure Failures	O O	Depends on automation framework
	Rollback	O O O	DNS TTL complexities (reaction time, flip/flop)
	Cost	O	Optimized via Auto Scaling
Swap the Auto Scaling group behind Elastic Load Balancer	Application Issues	O	Facilitates canary analysis
	Application Performance	O O	Less granular traffic split management, already warm load balancer
	People/Process Errors	O O	Depends on automation framework
	Infrastructure Failures	O	Auto Scaling
	Rollback	O	No DNS complexities
	Cost	O	Optimized via Auto Scaling
Update Auto Scaling Group	Application Issues	O O O	Detection of errors/issues in a heterogeneous fleet is complex

Technique	Risk Category	Risk Potential	Reasoning
launch configurations	Application Performance	● ● ●	Less granular traffic split, initial traffic load
	People/Process Errors	● ●	Depends on automation framework
	Infrastructure Failures	●	Auto Scaling
	Rollback	●	No DNS complexities
	Cost	● ●	Optimized via Auto Scaling, but initial scale-out overprovisions
Swap the environment of an Elastic Beanstalk application	Application Issues	● ●	Ability to do canary analysis ahead of cutover, but not with production traffic
	Application Performance	● ● ●	Full cutover
	People/Process Errors	●	Simple process, automated
	Infrastructure Failures	●	Auto Scaling, CloudWatch monitoring, Elastic Beanstalk health reporting
	Rollback	● ● ●	DNS TTL complexities
	Cost	● ●	Optimized via Auto Scaling, but initial scale-out may overprovision
Clone a stack in OpsWorks and update DNS	Application Issues	●	Facilitates canary analysis
	Application Performance	●	Gradual switch, traffic split management
	People/Process Errors	●	Highly automated
	Infrastructure Failures	●	Auto-healing capability
	Rollback	● ● ●	DNS TTL complexities
	Cost	● ● ●	Dual stack of resources

Document Revisions

Date	Revision
June 2015	Initial Publication

Notes

[1] https://aws.amazon.com/tools/

[2] https://aws.amazon.com/route53/

[3] https://aws.amazon.com/elasticloadbalancing/

[4] https://aws.amazon.com/autoscaling/

[5] http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/AutoScalingEnteringAndExitingStandby.html

[6] https://aws.amazon.com/elasticbeanstalk/

[7] https://aws.amazon.com/opsworks/

[8] https://aws.amazon.com/cloudformation/

[9] https://aws.amazon.com/cloudwatch/

[10] Alias records are specific to Amazon Route 53, offering extended capabilities to standard DNS. They act as pointers to other AWS resources such as Elastic Load Balancing endpoints or Amazon CloudFront distributions. You can read more about them at http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/resource-record-sets-choosing-alias-non-alias.html.

[11] For best practices for evaluating Elastic Load Balancing, see http://aws.amazon.com/articles/1636185810492479.

[12] http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/how-elb-works.html

[13] http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/AutoScalingEnteringAndExitingStandby.html

[14] For additional information about Auto Scaling state lifecycle, see http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/AutoScalingGroupLifecycle.html.

[15] http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/AutoScalingEnteringAndExitingStandby.html

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

[17] Refer to the AWS OpsWorks web page at https://aws.amazon.com/opsworks/ for more details.

[18] For more information on pre-warming with Elastic Load Balancing, see http://aws.amazon.com/articles/1636185810492479#pre-warming.