Introduction to DevOps on AWS

David Chapman

December 2014

Contents

Abstract

As innovation accelerates and customer needs rapidly evolve, businesses must become increasingly agile. Time to market is key, and to facilitate overall business goals, IT departments need to be agile. Over the years software development lifecycles moved from waterfall to agile models of development. These improvements are moving downstream toward IT operations with the evolution of DevOps.

In order to meet the demands of an agile business, IT operations need to deploy applications in a consistent, repeatable, and reliable manner. This can only be fully achieved with the adoption of automation.

Amazon Web Services (AWS) supports numerous DevOps principles and practices that IT departments can capitalize on to improve business agility.

This paper focuses on DevOps principles and practices supported on the AWS platform. A brief introduction to the origins of DevOps sets the scene and explains how and why DevOps has evolved.

Introduction

DevOps is a new term that primarily focuses on improved collaboration, communication, and integration between software developers and IT operations. It's an umbrella term that some describe as a philosophy, cultural change, and paradigm shift.

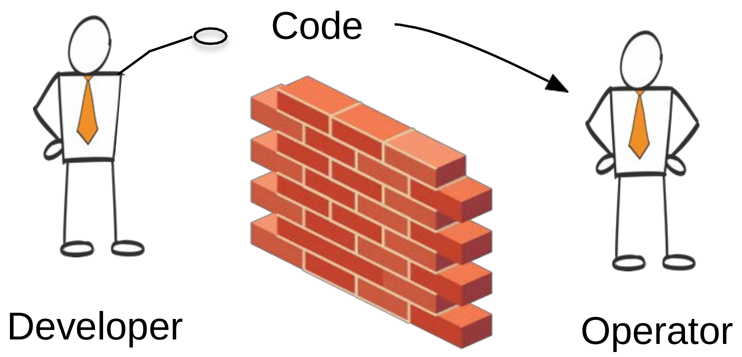


Figure 1: Developer throwing code "over the wall"

Historically many organizations have been vertically structured with poor integration among development, infrastructure, security and support teams. Frequently the groups report into different organizational structures with different corporate goals and philosophies.

Deploying software has predominately been the role of the IT operations group. Fundamentally developers like to build software and change things quickly, whereas IT operations focus on stability and reliability. This mismatch of goals can lead to conflict, and ultimately the business may suffer.

Today, these old divisions are breaking down, with the IT and developer roles merging and following a series of systematic principles:

- Infrastructure as code
- Continuous deployment
- Automation
- Monitoring
- Security

An examination of each of these principles reveals a close connection to the offerings available from Amazon Web Services.

Agile Evolution to DevOps

To fully appreciate DevOps principles, it is helpful to understand the context in which they evolved. The story begins with agile software development, which became popular over a decade ago and was seen as better approach to building software. Prior to agile, the dominant waterfall development methodology was based on a sequence starting with a requirements phase where 100% of the system under development was defined up front. The approach has shown itself to be inflexible and monolithic.

The agile model brought the concept of new and improved collaboration between business users and developers. Software development began to focus on iterations of working software that would evolve over time, delivering value along the way. Agile is a disciplined engineering process, and numerous tools now support it. For developers, such tools include IDEs, unit test frameworks, and code optimizers. As developers become more productive, the business becomes more agile and can respond to their customer requests more quickly and efficiently.

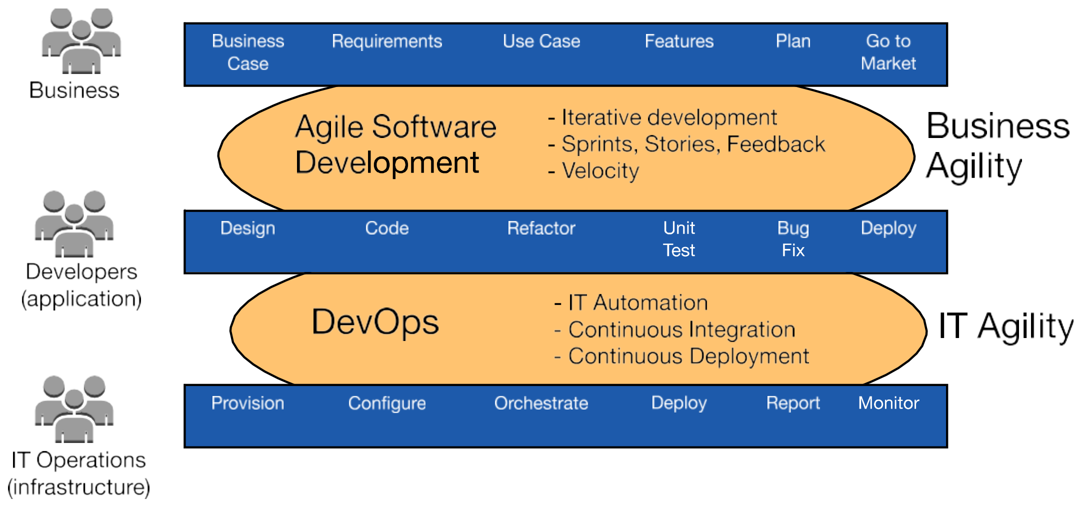


Figure 2: The coevolution of agile software development and DevOps

Over the last few years, the agile software development evolution has started to move downstream towards infrastructure under the label *DevOps*. Whereas agile software development primarily focuses on the collaboration between the business and its developers, DevOps focuses on the collaboration between developers, IT operations and security teams. IT operations include system administrators, database administrators, network engineers, infrastructure architects, and support personnel. Whereas agile software development provides business agility, DevOps provides IT agility, enabling the deployment of applications that are more reliable, predicable, and efficient.

DevOps practices vary with the task: With application development, DevOps focuses on code building, code coverage, unit testing, packaging, and deployment. With infrastructure, on the other hand, DevOps focuses on provisioning, configuration, orchestration, and deployment. But in each area the underlying principles of version management, deployment, roll back, roll forward, and testing remain the same.

Infrastructure as Code

A fundamental principle of DevOps is to treat infrastructure the same way developers treat code. Application code has a defined format and syntax. If the code is not written according to the rules of the programming language, applications cannot be created. Code is stored in a version-management system that logs a history of code development, changes, and bug fixes. When code is compiled (built) into applications, we expect a consistent application to be created. That is to say, the build is repeatable and reliable.

Practicing "infrastructure as code" means applying the same rigor of application code development to infrastructure provisioning. All configurations should be defined in a declarative way and stored in a version management system, just like application code. Infrastructure provisioning, orchestration, and deployment should support the use of the "infrastructure code."

Until recently the rigor applied to application code development has not necessarily been applied to infrastructure. Frequently infrastructure is provisioned using manual processes. Scripts developed during the provisioning may not be stored in version control systems and the creation of environments is not always repeatable, reliable, or consistent.

In contrast, AWS provides a DevOps-focused way of creating and maintaining infrastructure. Similar to the way software developers write application code, AWS provides services that enable the creation, deployment and maintenance of infrastructure in a programmatic, descriptive, and declarative way. These services provide rigor, clarity, and reliability. The AWS services discussed in this paper are core to a DevOps strategy and form the underpinnings of numerous higher level AWS DevOps principles and practices.

AWS CloudFormation

A good example of how the DevOps principles are used in practice is AWS CloudFormation.[1] By using AWS CloudFormation templates, you can define and model AWS resources that can be created and updated. These templates are written in a format called JavaScript Object Notation (JSON). The templates require a specific syntax and structure that depends on the types of resources being created and managed. By using templates, you can provision infrastructure in a repeatable and reliable way.

You can create custom AWS CloudFormation templates or use sample templates that are available publicly. Once templates are deployed or updated into the AWS environment, the collection of resources under management is called a "stack." You can manage stacks through the AWS Management Console, AWS Command Line Interface, or AWS CloudFormation APIs. Common actions include create-stack, describe-stacks, list-stacks, and update-stack.

When you create or update a stack in the console, events are displayed showing the status of the configuration. If an error occurs, the stack is rolled back to its previous state. Amazon Simple Notification Service (Amazon SNS) helps you manage these events. For example, you can use Amazon SNS to track stack creation and deletion progress via email and integrate with other processes programmatically.

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

With templates, you can work with a broad set on AWS offerings, including Amazon Simple Storage Service (Amazon S3), Auto Scaling, Amazon CloudFront, Amazon DynamoDB, Amazon Elastic Compute Cloud (EC2), Amazon ElastiCache, AWS Elastic Beanstalk, Elastic Load Balancing, AWS Identity and Access Management, AWS OpsWorks and Amazon Virtual Private Cloud.

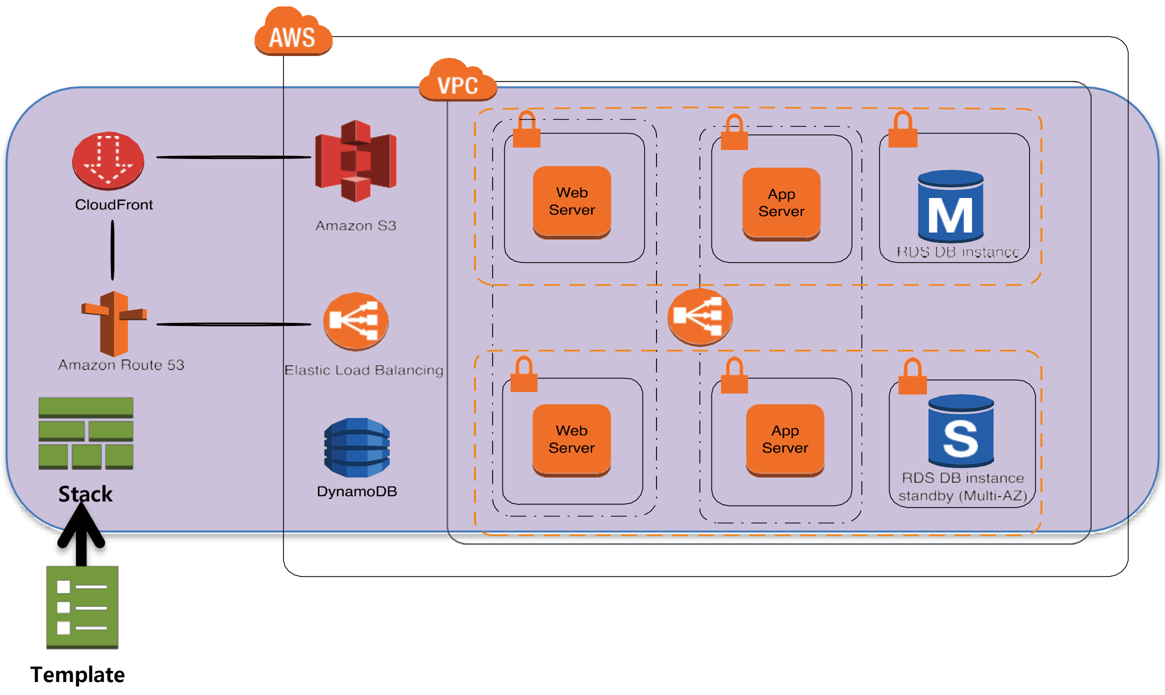


Figure 3: AWS CloudFormation example 1 — creating an entire environment (stack) from one template

You can use a single template to create and update an entire environment or separate templates to manage layers within an environment. This allows templates to be modularized and also provides a layer of governance that is important to many organizations.

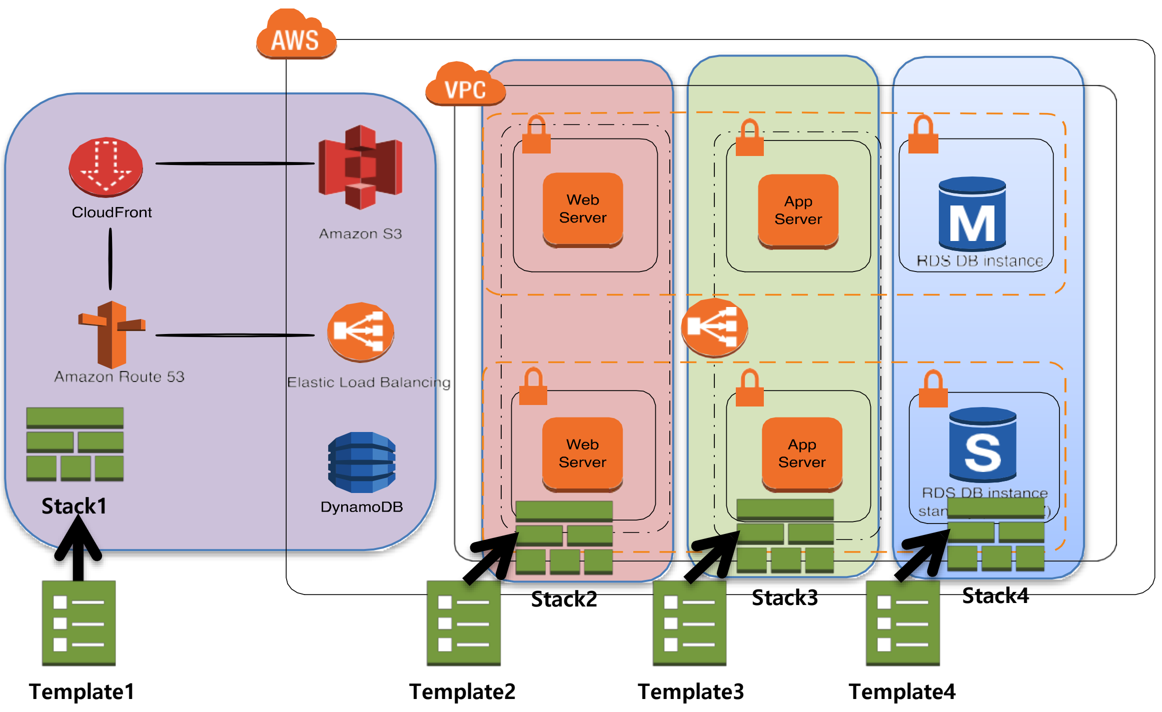
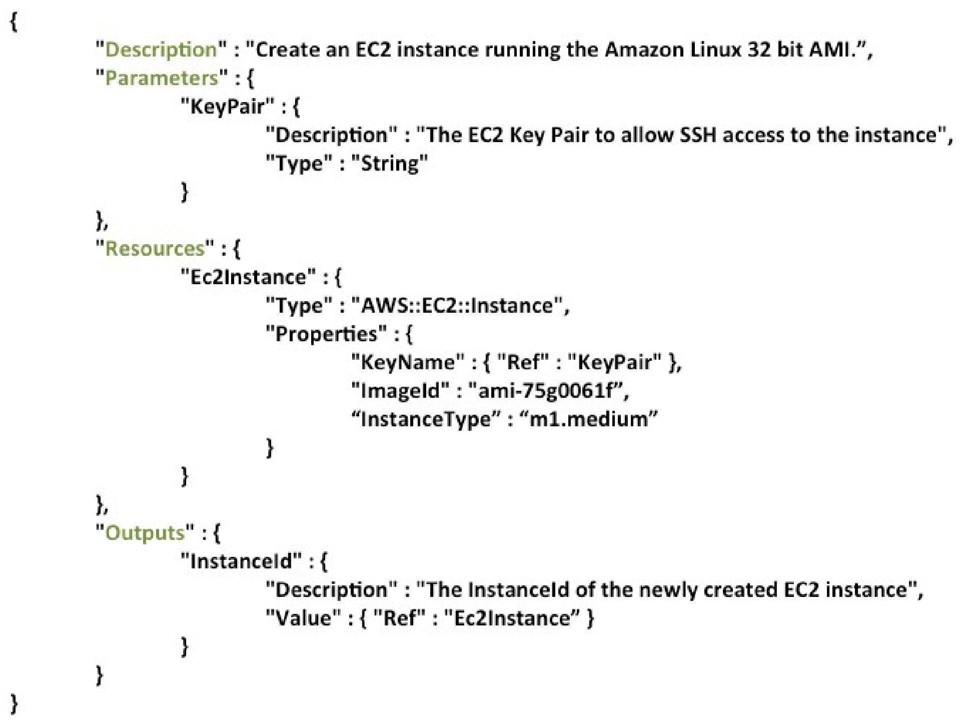


Figure 4: AWS CloudFormation example 2 — creating an entire environment (stacks) from multiple templates in a layered approach

AWS CloudFormation makes it easy to organize and deploy a collection of AWS resources and lets you describe any dependencies or pass in special parameters when the stack is configured.

To realize AWS CloudFormation's potential for "information as code," you should store templates in a source-code management systems version control before you deploy or update them in AWS. Amazon S3 provides a good location for storing and versioning templates. You can integrate AWS CloudFormation with the development and management tools of your choice.

There is no charge for defining your "infrastructure as code" in the AWS CloudFormation service. You are billed only the normal rates for the AWS resources that AWS CloudFormation creates and your application uses.



```
{
        "Description" : "Create an EC2 instance running the Amazon Linux 32 bit AMI.",
        "Parameters" : {
                "KeyPair" : {
                        "Description" : "The EC2 Key Pair to allow SSH access to the instance",
                        "Type" : "String"
                }
        },
        "Resources" : {
                "Ec2Instance" : {
                        "Type" : "AWS::EC2::Instance",
                        "Properties" : {
                                "KeyName" : { "Ref" : "KeyPair" },
                                "ImageId" : "ami-75g0061f",
                                "InstanceType" : "m1.medium"
                        }
                }
        },
        "Outputs" : {
                "InstanceId" : {
                        "Description" : "The InstanceId of the newly created EC2 instance",
                        "Value" : { "Ref" : "Ec2Instance" }
                }
        }
}
```

Figure 5: Example AWS CloudFormation template for launching an EC2 instance

In the above example, an AWS CloudFormation template has been defined in JSON notation to create an Amazon EC2 instance. This case provisions an m1.medium type of EC2 instance. The word `Ref` refers to parameters that are accessed when the stack is created. A parameter called `KeyPair` may be provided when the stack is created. This is the name of the key pair that is used when accessing the instance with SSH.

AWS AMI

An Amazon Machine Image (AMI) is another example of “infrastructure as code.” This core component of AWS computing is a kind of digital template that can launch (provision) Amazon EC2 instances, the fundamental AWS compute environment in the cloud. The image contains a software configuration such as a web server, application server, and database.

You can choose from three types of AMIs:

- AWS published
- Third party
- Custom created

AWS publishes AMIs that contain common software configurations based upon popular operating systems like Linux and Microsoft Windows. AMIs may also be obtained from third-party vendors, some of which are available on the AWS Marketplace.[2] You or your organization can also create and publish your own custom AMIs. Custom organizational AMIs usually include corporately distributed software such as hardened operating systems, antivirus software, and office productivity suites.

AMIs can also include application software that is bundled along with the AMI, or they can contain scripts and software that allow the instance to install application software at launch time, called "booting." There are pros and cons of preloading application-level software into an AMI. On the plus side, the instances can be launched very rapidly because they don't need to install any additional software at boot time. However, you may need to create a new AMI every time your application-level software changes.

Continuous Deployment

Continuous deployment is another core concept in a DevOps strategy. Its primary goal is to enable the automated deployment of production-ready application code.

Sometimes continuous deployment is referred to as continuous delivery. The only difference is that continuous deployment usually refers to production deployments.

By using continuous delivery practices and tools, software can be deployed rapidly, repeatedly, and reliably. If a deployment fails, it can be automatically rolled back to previous version.

AWS CodeDeploy

A prime example of this principle in AWS is the code deployment service AWS CodeDeploy.[3] Its core features provide the ability to deploy applications across an Amazon E2C fleet with minimum downtime, centralizing control and integrating with your existing software release or continuous delivery process.

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

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

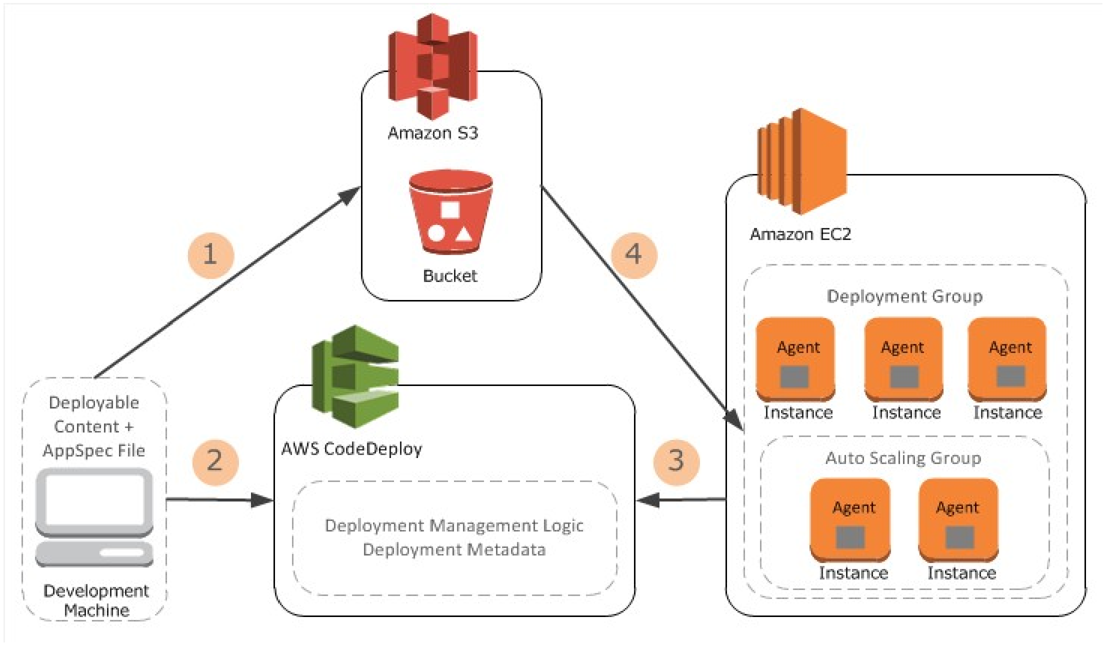


Figure 6: AWS CodeDeploy process

Here's how it works:

1. Application content is packaged and deployed to Amazon S3 along with an Application Specific (AppSpec) file that defines a series of deployment steps that AWS CodeDeploy needs to execute. The package is called a CodeDeploy "revision."
2. You create an application in AWS CodeDeploy and define the instances to which the application should be deployed (DeploymentGroup). The application also defines the Amazon S3 bucket where the deployment package resides.
3. An AWS CodeDeploy agent is deployed on each participating Amazon EC2. The agent polls AWS CodeDeploy to determine what and when to pull a revision from the specified Amazon S3 bucket.
4. The AWS CodeDeploy agent pulls the packaged application code and deploys it on the instance. The AppSec file containing deployment instructions is also downloaded.

In this way, AWS CodeDeploy exemplifies the continuous automated deployment that is central to DevOps.

AWS CodePipeline

Like AWS CodeDeploy, AWS CodePipeline (available in 2015) is a continuous delivery and release automation service that aids smooth deployments.[4] 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. You can integrate third-party tools into any step of your release process, or you can use AWS CodePipeline as an end-to-end solution. With AWS CodePipeline, you can rapidly deliver features and updates with high quality through the automation of your build, test, and release process.

AWS CodePipeline has several benefits that align with the DevOps principle of continuous deployment:

- Rapid delivery
- Improved quality
- Configurable workflow
- Easy to integrate

AWS CodeCommit

Also coming in 2015 is AWS CodeCommit, a secure, highly scalable, managed source control service that hosts private Git repositories.[5] CodeCommit eliminates the need for you to operate your own source control system or worry about scaling its infrastructure. You can use CodeCommit to store anything from code to binaries, and it supports the standard functionality of Git, allowing it to work seamlessly with your existing Git-based tools. Your team can also use CodeCommit's online code tools to browse, edit, and collaborate on projects.

AWS CodeCommit has several benefits:

- Fully managed
- Able to store anything
- Highly available
- Offers faster development lifecycles
- Works with your existing tools
- Secure

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

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

AWS Elastic Beanstalk and AWS OpsWorks

Both AWS Elastic Beanstalk[6] and AWS OpsWorks[7] support continuous deployment of application code changes and infrastructure modifications. In AWS Elastic Beanstalk, code changes deployments are stored as “application versions,” and infrastructure changes are deployed “saved configurations.” AWS OpsWorks has its own process for deploying applications and can define additional run-time launch commands and Chef recipes.

An example of an application version would be a new Java application that you upload as a `.zip` or `.war` file. An example of a saved configuration would be an AWS Elastic Beanstalk configuration that uses Elastic Load Balancing and Auto Scaling rather than a single instance. When you finish making changes, you can save your new configuration.

AWS Elastic Beanstalk supports the DevOps practice called “rolling deployments.” When enabled, your configuration deployments work hand in hand with Auto Scaling to ensure there are always a defined number of instances available as configuration changes are made. This gives you control as Amazon EC2 instances are updated. For example, if the EC2 instance type is being changed, you can determine whether AWS Elastic Beanstalk updates all instances concurrently or keeps some instances running to serve requests as other instances are being updated.

Similarly, AWS OpsWorks gives you the option of defining which instances in which layers should be updated when deployments are made.

Additional features of AWS Elastic Beanstalk and AWS OpsWorks are described in the Automation section.

Blue–Green Deployment

Blue–green deployment is a DevOps deployment practice that uses domain name services (DNS) to make application deployments. The strategy involves starting with an existing (blue) environment while testing a new (green) one. When the new environment has passed all the necessary tests and is ready to go live, you simply redirect traffic from the old environment to the new one via DNS.

AWS offers all the tools that you need for implementing a blue–green development strategy. You configure your ideal new infrastructure environment by using a service like AWS CloudFormation or AWS Elastic Beanstalk. With AWS CloudFormation templates, you can easily create a new environment identical to the existing production environment.

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

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

If you use the AWS DNS service Amazon Route 53, you can direct the traffic flow by means of weighted resource record sets. By using these record sets, you can define multiple services or load balancers with the DNS resolution.

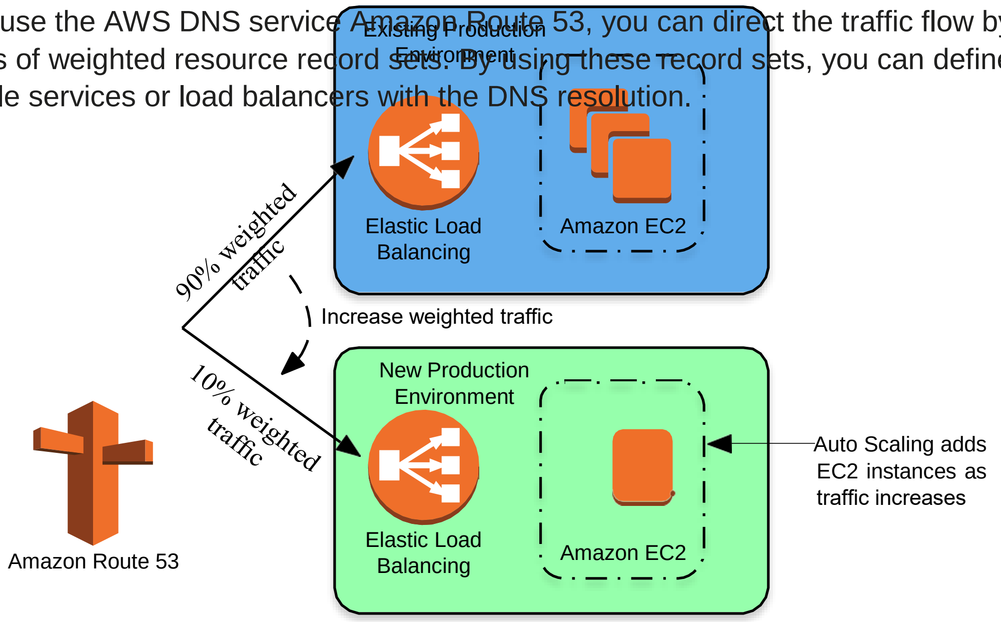

Existing Production Environment
Elastic Load Balancing
Amazon EC2
90% weighted traffic
Increase weighted traffic
10% weighted traffic
New Production Environment
Elastic Load Balancing
Amazon EC2
Auto Scaling adds EC2 instances as traffic increases
Amazon Route 53

Figure 7: Blue-Green deployment using Amazon Route 53 weighted resource record sets

The DNS service resolution (converting a domain name to an IP address) is weighted, meaning you can define how much traffic is directed to your newly deployed production environment. By using this feature, you can test the environment and, when you are confident that the deployment is good, increase the weighting. When the old production environment is receiving 0% traffic, you can either keep it for backup purposes or decommission it. As the amount of the traffic in the new environment increases, you can use Auto Scaling to scale up additional Amazon EC2 instances.

This ability to create and dispose of identical environments easily in the AWS cloud makes DevOps practices like blue–green deployment feasible.

You can also use blue–green deployment for back-end services like database deployment and failover.

Automation

Another core philosophy and practice of DevOps is automation. Automation focuses on the setup, configuration, deployment, and support of infrastructure and the applications that run on it. By using automation, you can set up environments more rapidly in a standardized and repeatable manner. The removal of manual processes is a key to a successful DevOps strategy. Historically, server configuration and application deployment have been predominantly a manual process. Environments become nonstandard, and reproducing an environment when issues arise is difficult.

The use of automation is critical to realizing the full benefits of the cloud. Internally AWS relies heavily on automation to provide the core features of elasticity and scalability. Manual processes are error prone, unreliable, and inadequate to support an agile business. Frequently an organization may tie up highly skilled resources to provide manual configuration. Time could be better spent supporting other, more critical and higher value activities within the business.

Modern operating environments commonly rely on full automation to eliminate manual intervention or access to production environments. This includes all software releasing, machine configuration, operating system patching, troubleshooting, or bug fixing. Many levels of automation practices can be used together to provide a higher level end-to-end automated process.

Automation has many benefits:

- Rapid changes
- Improved productivity
- Repeatable configurations
- Reproducible environments
- Leveraged elasticity
- Leveraged auto scaling
- Automated testing

Automation is a cornerstone with AWS services and is internally supported in all services, features, and offerings.

AWS Elastic Beanstalk

For an example of automation in AWS, one need look no further than AWS Elastic Beanstalk. AWS Elastic Beanstalk is an application container that makes it easy and productive for developers to deploy applications into commonly used technology stacks. Its simple-to-use interface helps developers deploy multitiered applications quickly and easily. AWS Elastic Beanstalk supports automation and numerous other DevOps best practices including automated application deployment, monitoring, infrastructure configuration, and version management. Application and infrastructure changes can be easily rolled back as well as forward.

Creating environments provides a good example of AWS Elastic Beanstalk automation. You simply specify the details for your environment, and AWS Beanstalk does all the configuration and provisioning work on its own. For example, here are just some of the options you can specify for in the create application wizard:

- Whether you want a web server tier (which contains a web server and an application server) or a worker tier (which utilizes the Amazon Simple Queue Service).
- What platform to use as a container for your application. Choices include IIS, Node.js, PHP, Python, Ruby, Tomcat, or Docker.
- Whether to launch a single instance or create a load balancing, autoscaling environment.
- What URL to automatically assign to your environment.
- Whether the environment includes an Amazon Relational Database instance.
- Whether to create your environment inside an Amazon Virtual Private Cloud.
- What URL (if any) to use for automatic health checks of your application.
- What tags (if any) to apply to identify your environment.

AWS Elastic Beanstalk also uses automation to deploy applications. Depending on the platform, all you need to do to deploy applications is to upload packages in the form of `.war` or `.zip` files directly from your computer or from Amazon S3.

As the environment is being created, AWS Elastic Beanstalk automatically logs events on the management console providing feedback on the progress and status of the launch. Once complete, you can access your application by using the defined URL.

AWS Elastic Beanstalk can be customized should you want to take control over certain aspects of the application and technology stack.

AWS OpsWorks

AWS OpsWorks take the principles of DevOps even further than AWS Elastic Beanstalk. It can be considered an application management service rather than simply an application container. AWS OpsWorks provides even more levels of automation with additional features like integration with configuration management software (Chef) and application lifecycle management. You can use application lifecycle management to define when resources are set up, configured, deployed, un-deployed, or terminated.

For added flexibility AWS OpsWorks has you define your application in configurable stacks. You can also select predefined application stacks. Application stacks contain all the provisioning for AWS resources that your application requires, including application servers, web servers, databases, and load balancers.

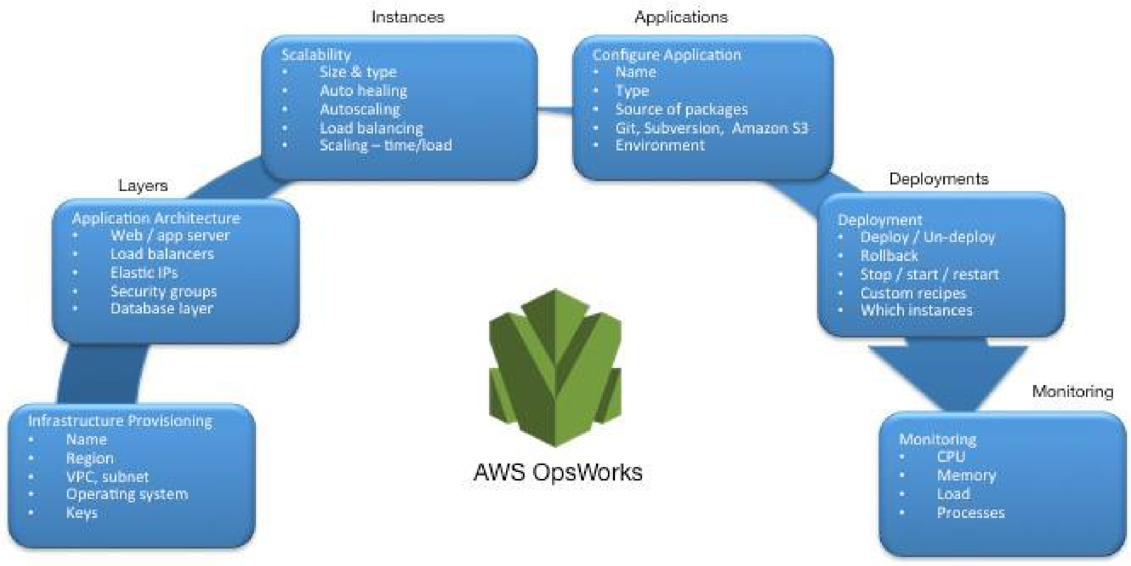


Figure 8: AWS OpsWorks showing DevOps features and architecture

Application stacks are organized into architectural layers so that stacks can be maintained independently. Example layers could include web tier, application tier, and database tier. Out of the box, AWS OpsWorks also simplifies setting up Auto Scaling groups and Elastic Load Balancing load balancers, further illustrating the DevOps principle of automation. Just like AWS Elastic Beanstalk, AWS OpsWorks supports application versioning, continuous deployment, and infrastructure configuration management.

AWS OpsWorks also supports the DevOps practices of monitoring and logging (covered in the next section). Monitoring support is provided by Amazon CloudWatch. All lifecycle events are logged, and a separate Chef log documents any Chef recipes that are run, along with any exceptions.[8]

Monitoring

Communication and collaboration is fundamental in a DevOps strategy. To facilitate this, feedback is critical. In AWS feedback is provided by two core services: Amazon CloudWatch and AWS CloudTrail. Together they provide a robust monitoring, alerting, and auditing infrastructure so developers and operations teams can work together closely and transparently.

[8] A Chef recipe is a Ruby application that defines everything that needs to be done to configure a system, such as installing packages and patches.

Amazon CloudWatch

Amazon CloudWatch monitors in real time all AWS resources and the applications you run on them.[9] Resources and applications can produce metrics that Amazon CloudWatch collates and tracks. You can configure alarms to send notifications when events occur. You can configure notifications in numerous formats, including email, Amazon SNS, and Amazon Simple Queue Service. The notifications can be delivered to individuals, teams, or other AWS resources.

As well as providing feedback, Amazon CloudWatch also supports the DevOps concept of automation. AWS services such as Auto Scaling rely on CloudWatch for notifications that trigger appropriate automated action such as scaling up and scaling down Amazon EC2 instances and load increases or decreases.

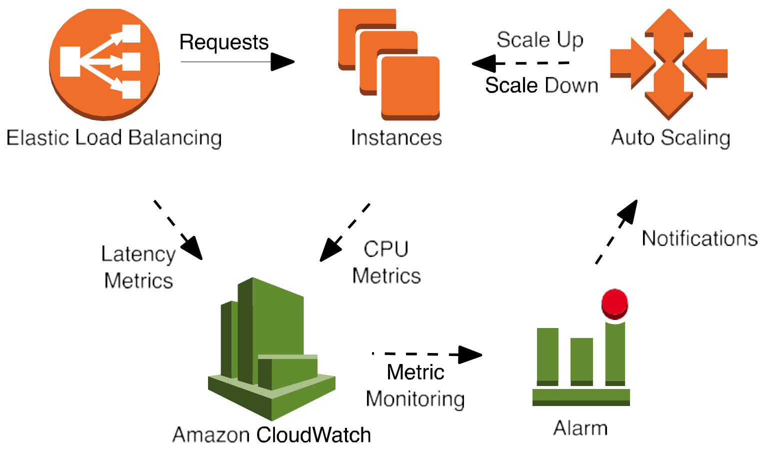


Figure 9: Example DevOps automation using Amazon CloudWatch and Auto Scaling

In the above example Amazon CloudWatch monitors latency metrics from Elastic Load Balancing and average CPU metrics from the running Amazon EC2 instances. Latency metrics measure how long it takes for replies to be returned after requests are made to the Amazon EC2 instances. You can create scaling policies to act upon alarms that are triggered when defined thresholds are broken. Such a policy can result in an increase or decrease in the number of Amazon EC2 instances, depending upon the situation. You can define additional notifications to deliver messages through Amazon SNS. This can be useful to notify interested parties such as support teams that events have occurred.

The Auto Scaling example illustrates how AWS services work together to provide automated, transparent services that are key to embracing a DevOps strategy. System administrators and support teams can focus on other value-added business needs and can rest assured that the AWS infrastructure is taking care of the applications scaling

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

requirements. Note that this scenario assumes that the application in question is cloud optimized and designed in a horizontally scalable way to leverage the benefits of Auto Scaling.

AWS CloudTrail

In order to embrace the DevOps principles of collaboration, communication, and transparency, it's important to understand who is making modifications to your infrastructure. In AWS this transparency is provided by AWS CloudTrail service.[10] All AWS interactions are handled through AWS API calls that are monitored and logged by AWS CloudTrail. All generated log files are stored in an Amazon S3 bucket that you define. Log files are encrypted using Amazon S3 server-side encryption (SSE). All API calls are logged whether they come directly from a user or on behalf of a user by an AWS service. Numerous groups can benefit from CloudTrail logs, including operations teams for support, security teams for governance, and finance teams for billing.

Security

In a DevOps enabled environment, focus on security is still of paramount importance. Infrastructure and company assets need to be protected, and when issues arise they need to be rapidly and effectively addressed.

Identity and Access Management (IAM)

The AWS Identity and Access Management service (IAM) is one component of the AWS security infrastructure. With IAM, you can centrally manage users and security credentials such as passwords, access keys, and permissions policies that control which AWS services and resources users can access. You can also use IAM to create roles that are used widely within a DevOps strategy. With an IAM role you can define a set of permissions to access the resources that a user or service needs. But instead of attaching the permissions to a specific user or group, you attach them to a named role. Resources can be associated with roles and services can then be programmatically defined to assume a role.

Security requirements and controls must be adhered to during any automation processes and great care should be made when working with passwords and keys. Security best practices should be followed at all times. For details about the importance of security to AWS, visit the AWS Security Center.[11]

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

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

Conclusion

In order to make the journey to the cloud smooth, efficient and effective, technology companies should embrace DevOps principles and practices. These principles are embedded in the AWS platform. Indeed, they form the cornerstone of numerous AWS services, especially those in the deployment and monitoring offerings.

Begin by defining your infrastructure as code using the service AWS CloudFormation. Next, define the way in which your applications are going to use continuous deployment with the help of services like AWS CodeDeploy, AWS CodePipeline, and AWS CodeCommit. At the application level, use containers like AWS Elastic Beanstalk and AWS OpsWorks to simplify the configuration of common architectures. Using these services also makes it easy to include other important services like Auto Scaling and Elastic Load Balancing. Finally, use the DevOps strategy of monitoring (AWS CloudWatch) and solid security practices (AWS IAM).

With AWS as your partner, your DevOps principles will bring agility to your business and IT organization and accelerate your journey to the cloud.