Infrastructure as Code

July 2017

This paper has been archived.

For the latest technical content about the AWS Cloud, see the AWS Whitepapers & Guides page:

https://aws.amazon.com/whitepapers

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

Infrastructure as Code has emerged as a best practice for automating the provisioning of infrastructure services. This paper describes the benefits of Infrastructure as Code, and how to leverage the capabilities of Amazon Web Services in this realm to support DevOps initiatives.

DevOps is the combination of cultural philosophies, practices, and tools that increases your organization's ability to deliver applications and services at high velocity. This enables your organization to be more responsive to the needs of your customers. The practice of Infrastructure as Code can be a catalyst that makes attaining such a velocity possible.

Introduction to Infrastructure as Code

Infrastructure management is a process associated with software engineering. Organizations have traditionally "racked and stacked" hardware, and then installed and configured operating systems and applications to support their technology needs. Cloud computing takes advantage of virtualization to enable the on-demand provisioning of compute, network, and storage resources that constitute technology infrastructures.

Infrastructure managers have often performed such provisioning manually. The manual processes have certain disadvantages, including:

- Higher cost because they require human capital that could otherwise go toward more important business needs.
- Inconsistency due to human error, leading to deviations from configuration standards.
- Lack of agility by limiting the speed at which your organization can release new versions of services in response to customer needs and market drivers.
- Difficulty in attaining and maintaining compliance to corporate or industry standards due to the absence of repeatable processes.

Infrastructure as Code addresses these deficiencies by bringing automation to the provisioning process. Rather than relying on manually performed steps, both administrators and developers can instantiate infrastructure using configuration files. Infrastructure as Code treats these configuration files as software code. These files can be used to produce a set of artifacts, namely the compute, storage, network, and application services that comprise an operating environment. Infrastructure as Code eliminates configuration drift through automation, thereby increasing the speed and agility of infrastructure deployments.

The Infrastructure Resource Lifecycle

In the previous section, we presented Infrastructure as Code as a way of provisioning resources in a repeatable and consistent manner. The underlying concepts are also relevant to the broader roles of infrastructure technology operations. Consider the following diagram.

Figure 1: Infrastructure resource lifecycle

Figure 1 illustrates a common view of the lifecycle of infrastructure resources in an organization. The stages of the lifecycle are as follows:

1. **Resource provisioning.** Administrators provision the resources according to the specifications they want.
2. **Configuration management.** The resources become components of a configuration management system that supports activities such as tuning and patching.
3. **Monitoring and performance.** Monitoring and performance tools validate the operational status of the resources by examining items such as metrics, synthetic transactions, and log files.
4. **Compliance and governance.** Compliance and governance frameworks drive additional validation to ensure alignment with corporate and industry standards, as well as regulatory requirements.

5. **Resource optimization.** Administrators review performance data and identify changes needed to optimize the environment around criteria such as performance and cost management.

Each stage involves procedures that can leverage code. This extends the benefits of Infrastructure as Code from its traditional role in provisioning to the entire resource lifecycle. Every lifecycle then benefits from the consistency and repeatability that Infrastructure as Code offers. This expanded view of Infrastructure as Code results in a higher degree of maturity in the Information Technology (IT) organization as a whole.

In the following sections, we explore each stage of the lifecycle – provisioning, configuration management, monitoring and performance, governance and compliance, and optimization. We will consider the various tasks associated with each stage and discuss how to accomplish those tasks using the capabilities of Amazon Web Services (AWS).

Resource Provisioning

The information resource lifecycle begins with resource provisioning. Administrators can use the principle of Infrastructure as Code to streamline the provisioning process. Consider the following situations:

- A release manager needs to build a replica of a cloud-based production environment for disaster recovery purposes. The administrator designs a template that models the production environment and provisions identical infrastructure in the disaster recovery location.
- A university professor wants to provision resources for classes each semester. The students in the class need an environment that contains the appropriate tools for their studies. The professor creates a template with the appropriate infrastructure components, and then instantiates the template resources for each student as needed.
- A service that has to meet certain industry protection standards requires infrastructure with a set of security controls each time the service is installed. The security administrator integrates the security controls into the configuration template so that the security controls are instantiated with the infrastructure.

- The manager of a software project team needs to provide development environments for programmers that include the necessary tools and the ability to interface with a continuous integration platform. The manager creates a template of the resources and publishes the template in a resource catalog. This enables the team members to provision their own environments as needed.

These situations have one thing in common: the need for a repeatable process for instantiating resources consistently. Infrastructure as Code provides the framework for such a process. To address this need, AWS offers AWS CloudFormation.[1]

AWS CloudFormation

AWS CloudFormation gives developers and systems administrators an easy way to create, manage, provision, and update a collection of related AWS resources in an orderly and predictable way. AWS CloudFormation uses templates written in JSON or YAML format to describe the collection of AWS resources (known as a stack), their associated dependencies, and any required runtime parameters. You can use a template repeatedly to create identical copies of the same stack consistently across AWS Regions. After deploying the resources, you can modify and update them in a controlled and predictable way. In effect, you are applying version control to your AWS infrastructure the same way you do with your application code.

Template Anatomy

Figure 2 shows a basic AWS CloudFormation YAML-formatted template fragment. Templates contain parameters, resource declaration, and outputs. Templates can reference the outputs of other templates, which enables modularization.

```
---
AWSTemplateFormatVersion: "version date"

Description:
  String

Parameters:
  set of parameters

Mappings:
  set of mappings

Conditions:
  set of conditions

Transform:
  set of transforms

Resources:
  set of resources

Outputs:
  set of outputs
```

Figure 2: Structure of an AWS CloudFormation YAML template

Figure 3 is an example of an AWS CloudFormation template. The template requests the name of an Amazon Elastic Compute Cloud (EC2) key pair from the user in the parameters section.[2] The resources section of the template then creates an EC2 instance using that key pair, with an EC2 security group that enables HTTP (TCP port 80) access.

```
Parameters:
  KeyName:
    Description: The EC2 key pair to allow SSH access to the
instance
    Type: AWS::EC2::KeyPair::KeyName
Resources:
  Ec2Instance:
    Type: AWS::EC2::Instance
    Properties:
      SecurityGroups: !Ref InstanceSecurityGroup
      KeyName: !Ref KeyName
      ImageId: ami-70065467
  InstanceSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Enable HTTP access via port 80
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '80'
        ToPort: '80'
        CidrIp: 0.0.0.0/0
```

Figure 3: Example of an AWS CloudFormation YAML template

Change Sets

You can update AWS CloudFormation templates with application source code to add, modify, or delete stack resources. The change sets feature enables you to preview proposed changes to a stack without performing the associated updates.[3] You can control the ability to create and view change sets using AWS Identity and Access Management (IAM).[4] You can allow some developers to create and preview change sets, while reserving the ability to update stacks or execute change sets to a select few. For example, you could allow a developer to see the impact of a template change before promoting that change to the testing stage.

There are three primary phases associated with the use of change sets.

1. Create the change set.

To create a change set for a stack, submit the changes to the template or parameters to AWS CloudFormation. AWS CloudFormation generates a change set by comparing the current stack with your changes.

2. View the change set.

   You can use the AWS CloudFormation console, AWS CLI, or AWS CloudFormation API to view change sets. The AWS CloudFormation console provides a summary of the changes and a detailed list of changes in JSON format. The AWS CLI and AWS CloudFormation API return a detailed list of changes in JSON format.

3. Execute the change set.

   You can select and execute the change set in the AWS CloudFormation console, use the `aws cloudformation execute-change-set` command in the AWS CLI, or the `ExecuteChangeSet` API.

The change sets capability allows you to go beyond version control in AWS CloudFormation by enabling you to keep track of what will actually change from one version to the next. Developers and administrators can gain more insight into the impact of changes before promoting them and minimize the risk of introducing errors.

Reusable Templates

Many programming languages offer ways to modularize code with constructs such as functions and subroutines. Similarly, AWS CloudFormation offers multiple ways to manage and organize your stacks. Although you can maintain all your resources within a single stack, large single-stack templates can become difficult to manage. There is also a greater possibility of encountering a number of AWS CloudFormation limits.[5]

When designing the architecture of your AWS CloudFormation stacks, you can group the stacks logically by function. Instead of creating a single template that includes all the resources you need, such as virtual private clouds (VPCs), subnets, and security groups, you can use nested stacks or cross-stack references.[6], [7]

The nested stack feature allows you to create a new AWS CloudFormation stack resource within an AWS CloudFormation template and establish a parent-child relationship between the two stacks. Each time you create an AWS

CloudFormation stack from the parent template, AWS CloudFormation also creates a new child stack. This approach allows you to share infrastructure code across projects while maintaining completely separate stacks for each project.

Cross-stack references enable an AWS CloudFormation stack to export values that other AWS CloudFormation stacks can then import. Cross-stack references promote a service-oriented model with loose coupling that allows you to share a single set of resources across multiple projects.

Template Linting

As with application code, AWS CloudFormation templates should go through some form of static analysis, also known as linting. The goal of linting is to determine whether the code is syntactically correct, identify potential errors, and evaluate adherence to specific style guidelines. In AWS CloudFormation, linting validates that a template is correctly written in either JSON or YAML.

AWS CloudFormation provides the `ValidateTemplate` API that checks for proper JSON or YAML syntax. [8] If the check fails, AWS CloudFormation returns a template validation error. For example, you can run the following command to validate a template stored in Amazon Simple Storage Service (Amazon S3): [9]

```
aws cloudformation validate-template --template-url \
    s3://examplebucket/example_template.template
```

You can also use third-party validation tools. For example, cfn-nag performs additional evaluations on templates to look for potential security concerns. Another tool, cfn-check, performs deeper checks on resource specifications to identify potential errors before they emerge during stack creation. [10], [11]

Best Practices

The AWS CloudFormation User Guide provides a list of best practices for designing and implementing AWS CloudFormation templates.[12] We provide links to these practices below.

Planning and organizing

- Organize Your Stacks By Lifecycle and Ownership[13]
- Use IAM to Control Access[14]

- Reuse Templates to Replicate Stacks in Multiple Environments[15]
- Use Nested Stacks to Reuse Common Template Patterns[16]
- Use Cross-Stack References to Export Shared Resources[17]

Creating templates

- Do Not Embed Credentials in Your Templates[18]
- Use AWS-Specific Parameter Types[19]
- Use Parameter Constraints[20]
- Use AWS::CloudFormation::Init to Deploy Software Applications on Amazon EC2 Instances[21]
- Use the Latest Helper Scripts[22]
- Validate Templates Before Using Them[23]
- Use Parameter Store to Centrally Manage Parameters in Your Templates[24]

Managing stacks

- Manage All Stack Resources Through AWS CloudFormation[25]
- Create Change Sets Before Updating Your Stacks[26]
- Use Stack Policies[27]
- Use AWS CloudTrail to Log AWS CloudFormation Calls[28]
- Use Code Reviews and Revision Controls to Manage Your Templates[29]
- Update Your Amazon EC2 Linux Instances Regularly[30]

Summary

The information resource lifecycle starts with the provisioning of resources. AWS CloudFormation provides a template-based way of creating infrastructure and managing the dependencies between resources during the creation process. With AWS CloudFormation, you can maintain your infrastructure just like application source code.

Configuration Management

Once you provision your infrastructure resources and that infrastructure is up and running, you must address the ongoing configuration management needs of the environment. Consider the following situations:

- A release manager wants to deploy a version of an application across a group of servers and perform a rollback if there are problems.
- A system administrator receives a request to install a new operating system package in developer environments, but leave the other environments untouched.
- An application administrator needs to periodically update a configuration file across all servers housing an application.

One way to address these situations is to return to the provisioning stage, provision fresh resources with the required changes, and dispose of the old resources. This approach, also known as infrastructure immutability, ensures that the provisioned resources are built anew according to the code baseline each time a change is made. This eliminates configuration drift.

There are times, however, when you might want to take a different approach. In environments that have high levels of durability, it might be preferable to have ways to make incremental changes to the current resources instead of reprovisioning them. To address this need, AWS offers Amazon EC2 Systems Manager and AWS OpsWorks for Chef Automate. [31], [32]

Amazon EC2 Systems Manager

Amazon EC2 Systems Manager is a collection of capabilities that simplifies common maintenance, management, deployment, and execution of operational tasks on EC2 instances and servers or virtual machines (VMs) in on-premises environments. Systems Manager helps you easily understand and control the current state of your EC2 instance and OS configurations. You can track and remotely manage system configuration, OS patch levels, application configurations, and other details about deployments as they occur over time. These capabilities help with automating complex and repetitive tasks, defining system configurations, preventing drift, and maintaining software compliance of both Amazon EC2 and on-premises configurations.

Table 1 lists the tasks that Systems Manager simplifies.

Tasks	Details
Run Command[33]	Manage the configuration of managed instances at scale by distributing commands across a fleet.
Inventory[34]	Automate the collection of the software inventory from managed instances.
State Manager[35]	Keep managed instances in a defined and consistent state.
Maintenance Window[36]	Define a maintenance window for running administrative tasks.
Patch Manager[37]	Deploy software patches automatically across groups of instances.
Automation[38]	Perform common maintenance and deployment tasks, such as updating Amazon Machine Images (AMIs).
Parameter Store[39]	Store, control, access, and retrieve configuration data, whether plain-text data such as database strings or secrets such as passwords, encrypted through AWS Key Management System (KMS).

Table 1: Amazon EC2 Systems Manager tasks

Document Structure

A Systems Manager document defines the actions that Systems Manager performs on your managed instances. Systems Manager includes more than a dozen preconfigured documents to support the capabilities listed in Table 1. You can also create custom version-controlled documents to augment the capabilities of Systems Manager. You can set a default version and share it across AWS accounts. Steps in the document specify the execution order. All documents are written in JSON and include both parameters and actions. As with AWS OpsWorks for Chef Automate, documents for Systems Manager become part of the infrastructure code base, bringing Infrastructure as Code to configuration management.

The following is an example of a custom document for a Windows-based host. The document uses the ipconfig command to gather the network configuration of the node and then installs MySQL.

```
{
   "schemaVersion": "2.0",
   "description": "Sample version 2.0 document v2",
   "parameters": {},
```

```
   "mainSteps": [
      {
         "action": "aws:runPowerShellScript",
         "name": "runShellScript",
         "inputs": {
            "runCommand": ["ipconfig"]
         }
      },
      {
         "action": "aws:applications",
         "name": "installapp",
         "inputs": {
            "action": "Install",
            "source":
"http://dev.mysql.com/get/Downloads/MySQLInstaller/mysql-
installer-community-5.6.22.0.msi"
         }
      }
   ]
}
```

Figure 4: Example of a Systems Manager document

Best Practices

The best practices for each of the Systems Manager capabilities appear below.

Run Command

- Improve your security posture by leveraging Run Command to access your EC2 instances, instead of SSH/RDP.[40]
- Audit all API calls made by or on behalf of Run Command using AWS CloudTrail.
- Use the rate control feature in Run Command to perform a staged command execution.[41]
- Use fine-grained access permissions for Run Command (and all Systems Manager capabilities) by using AWS Identity and Access Management (IAM) policies.[42]

Inventory

- Use Inventory in combination with AWS Config to audit your application configuration overtime.

State Manager

- Update the SSM agent periodically (at least once a month) using the preconfigured AWS-UpdateSSMAgent document.[43]
- Bootstrap EC2 instances on launch using EC2Config for Windows.[44]
- (Specific to Windows) Upload the PowerShell or Desired State Configuration (DSC) module to Amazon S3, and use AWS-InstallPowerShellModule.
- Use tags to create application groups. Then target instances using the `Targets` parameter, instead of specifying individual instance IDs.
- Automatically remediate findings generated by Amazon Inspector by using Systems Manager.[45]
- Use a centralized configuration repository for all of your Systems Manager documents, and share documents across your organization.[46]

Maintenance Windows

- Define a schedule for performing disruptive actions on your instances such as OS patching, driver updates, or software installs.

Patch Manager

- Use Patch Manager to roll out patches at scale and to increase fleet compliance visibility across your EC2 instances.

Automation

- Create self-serviceable runbooks for infrastructure as Automation documents.
- Use Automation to simplify creating AMIs from the AWS Marketplace or custom AMIs, using public documents, or authoring your own workflows.

- Use the documents AWS-UpdateLinuxAmi or AWS-UpdateWindowsAmi or create a custom Automation document to build and maintain images.

Parameter Store

- Use Parameter Store to manage global configuration settings in a centralized manner.[47]
- Use Parameter Store for secrets managements, encrypted through AWS KMS.[48]
- Use Parameter Store with Amazon EC2 Container Service (ECS) task definitions to store secrets.[49]

AWS OpsWorks for Chef Automate

AWS OpsWorks for Chef Automate brings the capabilities of Chef, a configuration management platform, to AWS. OpsWorks for Chef Automate further builds on Chef's capabilities by providing additional features that support DevOps capabilities at scale. Chef is based on the concept of *recipes*, configuration scripts written in the Ruby language that perform tasks such as installing services. Chef recipes, like AWS CloudFormation templates, are a form of source code that can be version controlled, thereby extending the principle of Infrastructure as Code to the configuration management stage of the resource lifecycle.

OpsWorks for Chef Automate expands the capabilities of Chef to enable your organization to implement DevOps at scale. OpsWorks for Chef Automate provides three key capabilities that you can configure to support DevOps practices: workflow, compliance, and visibility.

Workflow

You can use a workflow in OpsWorks for Chef Automate to coordinate development, test, and deployment. The workflow includes quality gates that enable users with the appropriate privileges to promote code between phases of the release management process. This capability can be very useful in supporting collaboration between teams. Each team can implement its own gates to ensure compatibility between the projects of each team.

Compliance

OpsWorks for Chef Automate provides features that can assist you with organizational compliance as part of configuration management. Chef Automate can provide reports that highlight matters associated with compliance and risk. You can also leverage profiles from well-known groups such as the Center for Internet Security (CIS).

Visibility

OpsWorks for Chef Automate provides visibility into the state of workflows and compliance within projects. A Chef user can create and view dashboards that provide information about related events and query the events through a user interface.

Recipe Anatomy

A Chef recipe consists of a set of resource definitions. The definitions describe the desired state of the resources and how Chef can bring them to that state. Chef supports over 60 resource types. A list of common resource types appears below.

Resource Name	Purpose
Bash	Execute a script using the bash interpreter
Directory	Manage directories
Execute	Execute a single command
File	Manage files
Git	Manage source resources in Git repositories
Group	Manage groups
Package	Manage packages
Route	Manage a Linux route table entry
Service	Manage a service
User	Manage users

Table 2: Common Chef resources

The following is an example of a Chef recipe. This example defines a resource based on the installation of the Apache web server. The resource definition includes a check for the underlying operating system. It uses the `case` operator to examine the value of `node[:platform]` to check for the underlying

operating system. The `action: install` directive brings the resource to the desired state (that is, it installs the package).

```
package 'apache2' do
  case node[:platform]
  when 'centos','redhat','fedora','amazon'
    package_name 'httpd'
  when 'debian','ubuntu'
    package_name 'apache2'
  end
  action :install
end
```

Figure 5: Example of a Chef recipe

Recipe Linting and Testing

A variety of tools is available from both Chef and the Chef user community that support linting (syntax checking) and unit and integration testing. We highlight some of the most common platforms in the following sections.

Linting with Rubocop and Foodcritic

Linting can be done on infrastructure code such as Chef recipes using tools such as Rubocop and Foodcritic. [50], [51], [52] Rubocop performs static analysis on Chef recipes based on the Ruby style guide. (Ruby is the language used to create Chef recipes.) This tool is part of the Chef Development Kit and can be integrated into the software development workflow. Foodcritic checks Chef recipes for common syntax errors based on a set of built-in rules, which can be extended by community contributions.

Unit Testing with ChefSpec

ChefSpec can provide unit testing on Chef cookbooks. [53] These tests can determine whether Chef is being asked to do the appropriate tasks to accomplish the desired goals. ChefSpec requires a configuration test specification that is then evaluated against a recipe.

For example, ChefSpec would not actually check whether Chef installed the Apache package, but instead checks whether a Chef recipe asked to install Apache. The goal of the test is to validate whether the recipe reflects the intentions of the programmer.

Integration Testing with Test Kitchen

Test Kitchen is a testing platform that creates test environments and then uses *bussers*, which are test frameworks, to validate the creation of the resources specified in the Chef recipes. [54]

By leveraging the previous testing tools in conjunction with OpsWorks for Chef Automate workflow capabilities, developers can automate the testing of their infrastructures during the development lifecycle. These tests are a form of code themselves and are another key part of the Infrastructure as Code approach to deployments.

Best Practices

The strategies, techniques, and suggestions presented here will help you get the maximum benefit and optimal outcomes from AWS OpsWorks for Chef Automate:

- Consider storing your Chef recipes in an Amazon S3 archive. Amazon S3 is highly reliable and durable. Explicitly version each archive file by using a naming convention. Or use Amazon S3 versioning, which provides an audit trail and an easy way to revert to an earlier version.
- Establish a backup schedule that meets your organizational governance requirements.
- Use IAM to limit access to the OpsWorks for Chef Automate API calls.

Summary

Amazon EC2 Systems Manager lets you deploy, customize, enforce, and audit an expected state configuration to your EC2 instances and servers or VMs in your on-premises environment. AWS OpsWorks for Chef Automate enables you to use Chef recipes to support the configuration of an environment. You can use OpsWorks for Chef Automate independently or on top of an environment provisioned by AWS CloudFormation. The run documents and policies associated with Systems Manager and the recipes associated with OpsWorks for Chef Automate can become part of the infrastructure code base and be controlled just like application source code.

Monitoring and Performance

Having reviewed the role of Infrastructure as Code in the provisioning of infrastructure resources and configuration management, we now look at infrastructure health. Consider how the following events could affect the operation of a website during periods of peak demand:

- Users of a web application are experiencing timeouts because of latency of the load balancer, making it difficult to browse the product catalogs.
- An application server experiences performance degradation due to insufficient CPU capacity and can no longer process new orders.
- A database that tracks session state doesn't have enough throughput. This causes delays as users transition through the various stages of an application.

These situations describe operational problems arising from infrastructure resources that don't meet their performance expectations. It's important to capture key metrics to assess the health of the environment and take corrective action when problems arise. Metrics provide visibility. With metrics, your organization can respond automatically to events. Without metrics, your organization is blind to what is happening in its infrastructure, thereby requiring human intervention to address all issues. With scalable and loosely coupled systems written in multiple languages and frameworks, it can be difficult to capture the relevant metrics and logs and respond accordingly. To address this need, AWS offers the Amazon CloudWatch services.[55]

Amazon CloudWatch

Amazon CloudWatch is a set of services that ingests, interprets, and responds to runtime metrics, logs, and events. CloudWatch automatically collects metrics from many AWS services, such as Amazon EC2, Elastic Load Balancing (ELB), and Amazon DynamoDB.[56], [57], [58] Responses can include built-in actions such as sending notifications or custom actions handled by AWS Lambda, a serverless event-driven compute platform.[59] The code for Lambda functions becomes part of the infrastructure code base, thereby extending Infrastructure as Code to the operational level. CloudWatch consists of three services: the main CloudWatch service, Amazon CloudWatch Logs, and Amazon CloudWatch Events. We now consider each of these in more detail.

Amazon CloudWatch

The main Amazon CloudWatch service collects and tracks metrics for many AWS services such as Amazon EC2, ELB, DynamoDB, and Amazon Relational Database Service (RDS). You can also create custom metrics for services you develop, such as applications. CloudWatch issues alarms when metrics reach a given threshold over a period of time.

Here are some examples of metrics and potential responses that could apply to the situations mentioned at the start of this section:

- If the latency of ELB exceeds five seconds over two minutes, send an email notification to the systems administrators.
- When the average EC2 instance CPU usage exceeds 60 percent for three minutes, launch another EC2 instance.
- Increase the capacity units of a DynamoDB table when excessive throttling occurs.

You can implement responses to metrics-based alarms using built-in notifications, or by writing custom Lambda functions in Python, Node.js, Java, or C#. Figure 6 shows an example of how a CloudWatch alarm uses Amazon Simple Notification Service (Amazon SNS) to trigger a DynamoDB capacity update.

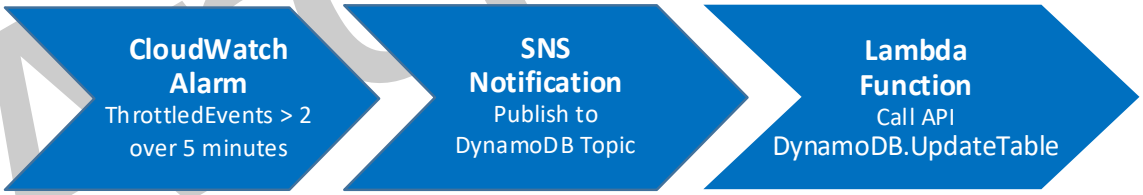

Figure 6: Example of a CloudWatch alarm flow

Amazon CloudWatch Logs

Amazon CloudWatch Logs monitors and stores logs from Amazon EC2, AWS CloudTrail, and other sources. EC2 instances can ship logging information using the CloudWatch Logs Agent and logging tools such as Logstash, Graylog, and Fluentd. [60] Logs stored in Amazon S3 can be sent to CloudWatch Logs by configuring an Amazon S3 event to trigger a Lambda function.

Ingested log data can be the basis for new CloudWatch metrics that can, in turn, trigger CloudWatch alarms. You can use this capability to monitor any resource that generates logs without writing any code whatsoever. If you need a more advanced response procedure, you can create a Lambda function to take the appropriate actions. For example, a Lambda function can use the SES.SendEmail or SNS.Publish APIs to publish information to a Slack channel when `NullPointerException` errors appear in production logs. [61], [62]

Log processing and correlation allow for deeper analysis of application behaviors and can expose internal details that are hard to figure out from metrics. CloudWatch Logs provides both the storage and analysis of logs, and processing to enable data-driven responses to operational issues.

Amazon CloudWatch Events

Amazon CloudWatch Events produces a stream of events from changes to AWS environments, applies a rules engine, and delivers matching events to specified targets. Examples of events that can be streamed include EC2 instance state changes, Auto Scaling actions, API calls published by CloudTrail, AWS console sign-ins, AWS Trusted Advisor optimization notifications, custom application-level events, and time-scheduled actions. Targets can include built-in actions such as SNS notifications or custom responses using Lambda functions.

The ability of an infrastructure to respond to selected events offers benefits in both operations and security. From the operations perspective, events can automate maintenance activities without having to manage a separate scheduling system. With regard to information security, events can provide notifications of console logins, authentication failures, and risky API calls recorded by CloudTrail. In both realms, incorporating event responses into the infrastructure code promotes a greater degree of self-healing and a higher level of operational maturity.

Best Practices

Here are some recommendations for best practices related to monitoring:

- Ensure that all AWS resources are emitting metrics.
- Create CloudWatch alarms for metrics that provide the appropriate responses as metric-related events arise.

- Send logs from AWS resources, including Amazon S3 and Amazon EC2, to CloudWatch Logs for analysis using log stream triggers and Lambda functions.
- Schedule ongoing maintenance tasks with CloudWatch and Lambda.
- Use CloudWatch custom events to respond to application-level issues.

Summary

Monitoring is essential to understand systems behavior and to automate data-driven reactions. CloudWatch collects observations from runtime environments, in the form of metrics and logs, and makes those actionable through alarms, streams, and events. Lambda functions written in Python, Node.js, Java, or C# can respond to events, thereby extending the role of Infrastructure as Code to the operational realm and improving the resiliency of operating environments.

Governance and Compliance

Having considered how you can use Infrastructure as Code to monitor the health of your organization's environments, we now turn our focus to technology governance and compliance. Many organizations require visibility into their infrastructures to address industry or regulatory requirements. The dynamic provisioning capabilities of the cloud pose special challenges because visibility and governance must be maintained as resources are added, removed, or updated. Consider the following situations:

- A user is added to a privileged administration group, and the IT organization is unable to explain when this occurred.
- The network access rules restricting remote management to a limited set of IP addresses are modified to allow access from additional locations.
- The RAM and CPU configurations for several servers has unexpectedly doubled, resulting in a much larger bill than in previous months.

Although you have visibility into the current state of your AWS resource configurations using the AWS CLI and API calls, addressing these situations requires the ability to look at how those resources have changed over time. To address this need, AWS offers the AWS Config service.[63]

AWS Config

AWS Config enables you to assess, audit, and evaluate the configurations of your AWS resources. AWS Config automatically builds an inventory of your resources and tracks changes made to them. Figure 7 shows an example of an AWS Config inventory of EC2 instances.

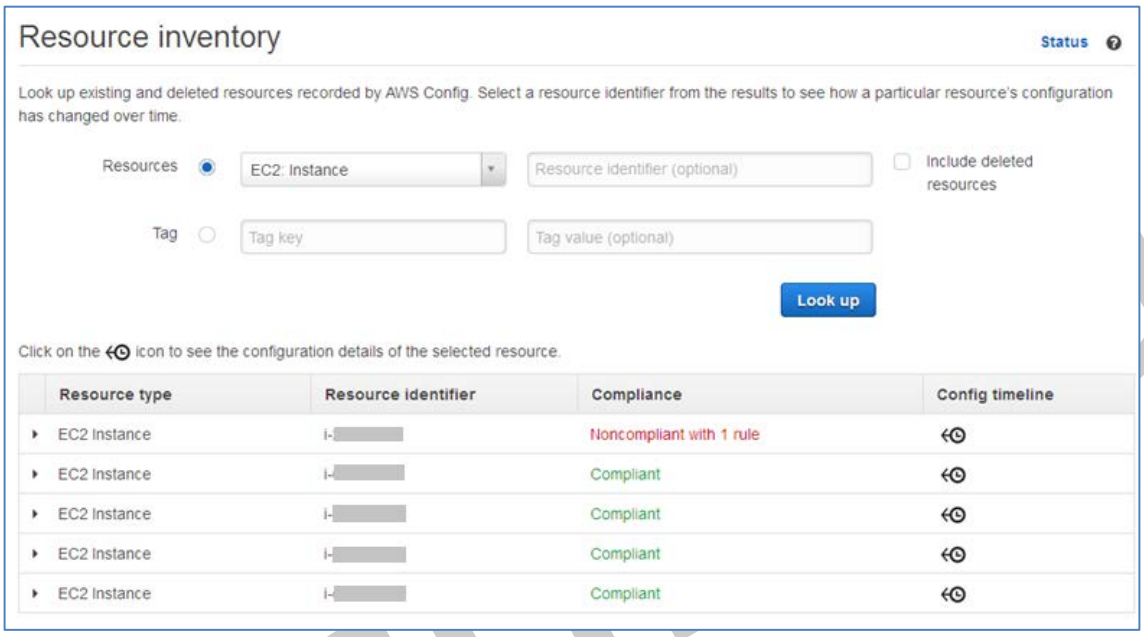

Figure 7: Example of an AWS Config resource inventory

AWS Config also provides a clear view of the resource change timeline, including changes in both the resource configurations and the associations of those resources to other AWS resources. Figure 8 shows the information maintained by AWS Config for a VPC resource.

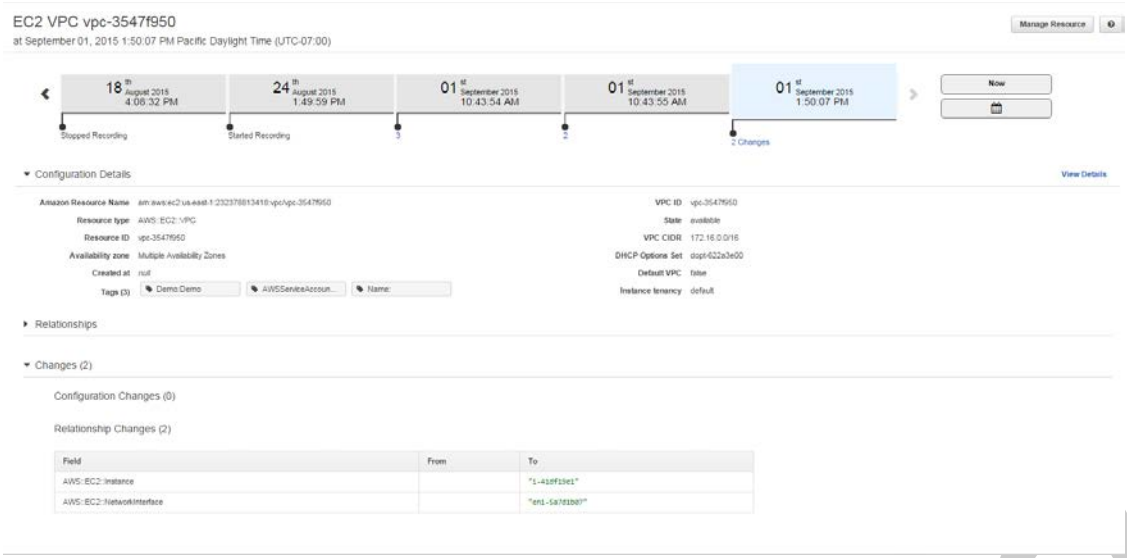

Figure 8: Example of AWS Config resource timeline

When many different resources are changing frequently and automatically, automating compliance can become as important as automating the delivery pipeline. To respond to changes in the environment, you can use AWS Config rules.

AWS Config Rules

With AWS Config rules, every change triggers an evaluation by the rules associated with the resources. AWS provides a collection of managed rules for common requirements such as IAM users having good passwords, groups and policies, or for determining if EC2 instances are on the correct VPCs and Security Groups. AWS Config rules can quickly identify noncompliant resources and help with reporting and remediation. For validations beyond those provided by the managed rules, AWS Config rules also support the creation of custom rules using Lambda functions.[64] These rules become part of the infrastructure code base, thus bringing the concept of Infrastructure as Code to the governance and compliance stages of the information resource lifecycle.

Rule Structure

When a custom rule is invoked through AWS Config rules, the associated Lambda function receives the configuration events, processes them, and returns results. The following function determines if Amazon Virtual Private Cloud (Amazon VPC) flow logs are enabled on a given Amazon VPC.

```
import boto3
import json

def evaluate_compliance(config_item, vpc_id):
    if (config_item['resourceType'] != 'AWS::EC2::VPC'):
        return 'NOT_APPLICABLE'
    elif is_flow_logs_enabled(vpc_id):
        return 'COMPLIANT'
    else:
        return 'NON_COMPLIANT'

def is_flow_logs_enabled(vpc_id):
    ec2 = boto3.client('ec2')
    response = ec2.describe_flow_logs(
        Filter=[{'Name': 'resource-id','Values': [vpc_id]},],
    )
    if len(response[u'FlowLogs']) != 0: return True

def lambda_handler(event, context):
    invoking_event = json.loads(event['invokingEvent'])
    compliance_value = 'NOT_APPLICABLE'
    vpc_id = invoking_event['configurationItem']['resourceId']
    compliance_value = evaluate_compliance(
        invoking_event['configurationItem'], vpc_id)

    config = boto3.client('config')
    response = config.put_evaluations(
       Evaluations=[
           {
               'ComplianceResourceType':
invoking_event['configurationItem']['resourceType'],
               'ComplianceResourceId': vpc_id,
               'ComplianceType': compliance_value,
               'OrderingTimestamp':
invoking_event['configurationItem']['configurationItemCaptureTim
e']
           },
       ],
       ResultToken=event['resultToken'])
```

Figure 9: Example of a Lambda function to support AWS Config rules

In this example, when a configuration event on an Amazon VPC occurs, the event passes to the function lambda_handler. This code extracts the ID of the Amazon VPC and uses the describe_flow_logs API call to determine whether the flow logs are enabled. The Lambda function returns a value of COMPLIANT if the flow logs are enabled and NON_COMPLIANT otherwise.

Best Practices

Here are some recommendations for implementing AWS Config in your environments:

- Enable AWS Config for all regions to record the configuration item history, to facilitate auditing and compliance tracking.
- Implement a process to respond to changes detected by AWS Config. This could include email notifications and the use of AWS Config rules to respond to changes programmatically.

Summary

AWS Config extends the concept of infrastructure code into the realm of governance and compliance. AWS Config can continuously record the configuration of resources while AWS Config rules allow for event-driven responses to changes in the configuration of tracked resources. You can use this capability to assist your organization with the monitoring of compliance controls.

Resource Optimization

We now focus on the final stage in the information resource lifecycle, resource optimization. In this stage, administrators review performance data and identify changes needed to optimize the environment around criteria such as security, performance, and cost management. It's important for all application stakeholders to regularly evaluate the infrastructure to determine if it is being used optimally.

Consider the following questions:

- Are there provisioned resources that are underutilized?

- Are there ways to reduce the charges associated with the operating environment?
- Are there any suggestions for improving the performance of the provisioned resources?
- Are there any service limits that apply to the resources used in the environment and, if so, is the current usage of resources close to exceeding these limits?

To answer these questions, we need a way to interrogate the operating environment, retrieve data related to optimization, and use that data to make meaningful decisions. To address this need, AWS offers AWS Trusted Advisor.[65]

AWS Trusted Advisor

AWS Trusted Advisor helps you observe best practices by scanning your AWS resources and comparing their usage against AWS best practices in four categories: cost optimization, performance, security, and fault tolerance. As part of ongoing improvement to your infrastructure and applications, taking advantage of Trusted Advisor can help keep your resources provisioned optimally. Figure 10 shows an example of the Trusted Advisor dashboard.

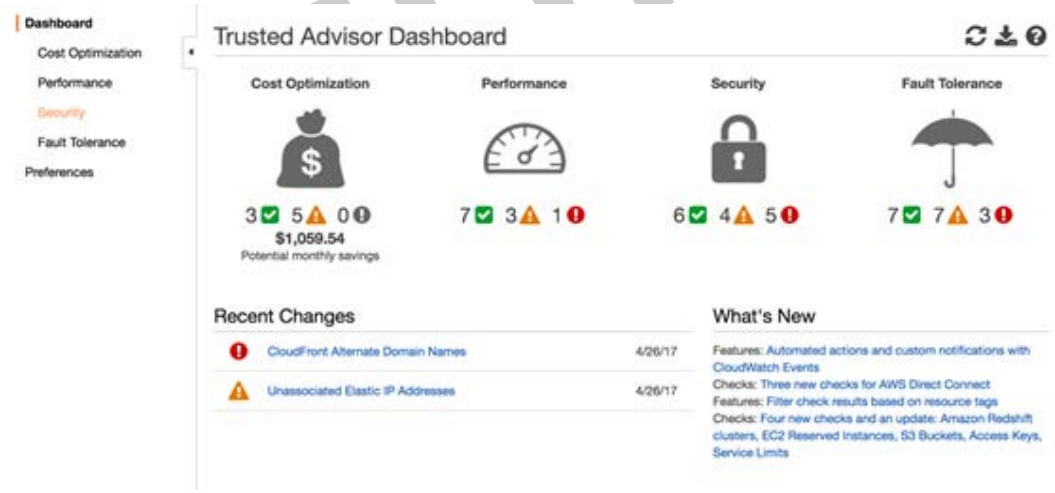

Figure 10: Example of the AWS Trusted Advisor dashboard

Checks

Trusted Advisor provides a variety of checks to determine if the infrastructure is following best practices. The checks include detailed descriptions of

recommended best practices, alert criteria, guidelines for action, and a list of useful resources on the topic. Trusted Advisor provides the results of the checks and can also provide ongoing weekly notifications for status updates and cost savings.

All customers have access to a core set of Trusted Advisor checks. Customers with AWS Business or Enterprise support can access all Trusted Advisor checks and the Trusted Advisor APIs. Using the APIs, you can obtain information from Trusted Advisor and take corrective actions. For example, a program could leverage Trusted Advisor to examine current account service limits. If current resource usages approach the limits, you can automatically create a support case to increase the limits.

Additionally, Trusted Advisor now integrates with Amazon CloudWatch Events. You can design a Lambda function to notify a Slack channel when the status of Trusted Advisor checks changes. These examples illustrate how the concept of Infrastructure as Code can be extended to the resource optimization level of the information resource lifecycle.

Best Practices

The best practices for AWS Trusted Advisor appear below.

- Subscribe to Trusted Advisor notifications through email or an alternative delivery system.
- Use distribution lists and ensure that the appropriate recipients are included on all such notifications.
- If you have AWS Business or Enterprise support, use the AWS Support API in conjunction with Trusted Advisor notifications to create cases with AWS Support to perform remediation.

Summary

You must continuously monitor your infrastructure to optimize the infrastructure resources with regard to performance, security, and cost. AWS Trusted Advisor provides the ability to use APIs to interrogate your AWS infrastructure for recommendations, thus extending Infrastructure as Code to the optimization phase of the information resource lifecycle.

Next Steps

You can begin the adoption of Infrastructure as Code in your organization by viewing your infrastructure specifications in the same way you view your product code. AWS offers a wide range of tools that give you more control and flexibility over how you provision, manage, and operationalize your cloud infrastructure.

Here are some key actions you can take as you implement Infrastructure as Code in your organization:

- Start by using a managed source control service, such as AWS CodeCommit, for your infrastructure code.
- Incorporate a quality control process via unit tests and static code analysis before deployments.
- Remove the human element and automate infrastructure provisioning, including infrastructure permission policies.
- Create idempotent infrastructure code that you can easily redeploy.
- Roll out every new update to your infrastructure via code by updating your idempotent stacks. Avoid making one-off changes manually.
- Embrace end-to-end automation.
- Include infrastructure automation work as part of regular product sprints.
- Make your changes auditable, and make logging mandatory.
- Define common standards across your organization and continuously optimize.

By embracing these principles, your infrastructure can dynamically evolve and accelerate with your rapidly changing business needs.

Conclusion

Infrastructure as Code enables you to encode the definition of infrastructure resources into configuration files and control versions, just like application

software. We can now update our lifecycle diagram and show how AWS supports each stage through code.

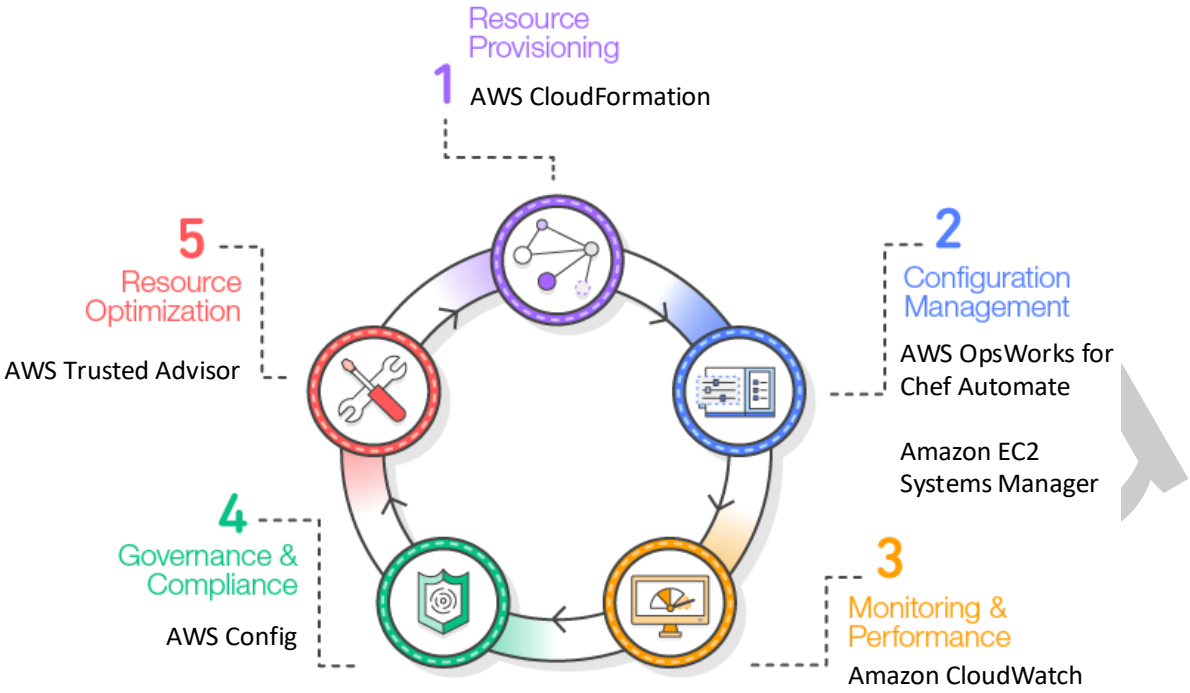

Figure 11: Information resource lifecycle with AWS

AWS CloudFormation, AWS OpsWorks for Chef Automate, Amazon EC2 Systems Manager, Amazon CloudWatch, AWS Config, and AWS Trusted Advisor enable you to integrate the concept of Infrastructure as Code into all phases of the project lifecycle. By using Infrastructure as Code, your organization can automatically deploy consistently built environments that, in turn, can help your organization to improve its overall maturity.

Contributors

The following individuals and organizations contributed to this document:

- Hubert Cheung, solutions architect, Amazon Web Services
- Julio Faerman, technical evangelist, Amazon Web Services
- Balaji Iyer, professional services consultant, Amazon Web Services
- Jeffrey S. Levine, solutions architect, Amazon Web Services

Resources

Refer to the following resources to learn more about our best practices related to Infrastructure as Code.

Videos

- AWS re:Invent 2015 – DevOps at Amazon[66]
- AWS Summit 2016 - DevOps, Continuous Integration and Deployment on AWS[67]

Documentation & Blogs

- DevOps and AWS[68]
- What is Continuous Integration[69]
- What is Continuous Delivery[70]
- AWS DevOps Blog[71]

Whitepapers

- Introduction to DevOps on AWS[72]
- AWS Operational Checklist[73]
- AWS Security Best Practices[74]
- AWS Risk and Compliance[75]

AWS Support

- AWS Premium Support[76]
- AWS Trusted Advisor[77]

Notes

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

[2] https://aws.amazon.com/ec2/

[3] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-changesets.html

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

[5] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cloudformation-limits.html

[6] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-stack.html

[7] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/walkthrough-crossstackref.html

[8] http://docs.aws.amazon.com/AWSCloudFormation/latest/APIReference/API_ValidateTemplate.html

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

[10] https://stelligent.com/2016/04/07/finding-security-problems-early-in-the-development-process-of-a-cloudformation-template-with-cfn-nag/

[11] https://www.npmjs.com/package/cfn-check

[12] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html

[13] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#organizingstacks

[14] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#use-iam-to-control-access

[15] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#reuse

[16] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#nested

[17] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#cross-stack

[18] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#creds

[19] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#parmtypes

[20] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#parmconstraints

[21] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#cfninit

[22] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#helper-scripts

[23] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#validate

[24] https://aws.amazon.com/ec2/systems-manager/parameter-store/

[25] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#donttouch

[26] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#cfn-best-practices-changesets

[27] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#stackpolicy

[28] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#cloudtrail

[29] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#code

[30] http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/best-practices.html#update-ec2-linux

[31] https://aws.amazon.com/ec2/systems-manager/

[32] https://aws.amazon.com/opsworks/chefautomate/

[33] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/execute-remote-commands.html

[34] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/systems-manager-inventory.html

[35] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/systems-manager-state.html

[36] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/systems-manager-ami.html

[37] https://aws.amazon.com/ec2/systems-manager/patch-manager/

[38] https://aws.amazon.com/ec2/systems-manager/automation/

[39] https://aws.amazon.com/ec2/systems-manager/parameter-store/

[40] https://aws.amazon.com/blogs/mt/replacing-a-bastion-host-with-amazon-ec2-systems-manager/

[41] http://docs.aws.amazon.com/systems-manager/latest/userguide/send-commands-multiple.html

[42] http://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-configuring-access-iam-create.html

[43] https://aws.amazon.com/blogs/mt/replacing-a-bastion-host-with-amazon-ec2-systems-manager/

[44] http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/ec2-configuration-manage.html

[45] https://aws.amazon.com/blogs/security/how-to-remediate-amazon-inspector-security-findings-automatically/

[46] http://docs.aws.amazon.com/systems-manager/latest/userguide/ssm-sharing.html

[47] http://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-paramstore.html

[48] http://docs.aws.amazon.com/systems-manager/latest/userguide/sysman-paramstore-walk.html

[49] https://aws.amazon.com/blogs/compute/managing-secrets-for-amazon-ecs-applications-using-parameter-store-and-iam-roles-for-tasks/

[50] https://en.wikipedia.org/wiki/Lint_(software)

[51] https://docs.chef.io/rubocop.html

[52] https://docs.chef.io/foodcritic.html

[53] https://docs.chef.io/chefspec.html

[54] https://docs.chef.io/kitchen.html

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

[56] https://aws.amazon.com/dynamodb/

[57] https://aws.amazon.com/ec2/

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

[59] https://aws.amazon.com/lambda/

[60] http://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/QuickStartEC2Instance.html

[61] http://docs.aws.amazon.com/ses/latest/APIReference/API_SendEmail.html

[62] http://docs.aws.amazon.com/sns/latest/api/API_Publish.html

[63] https://aws.amazon.com/config/

[64] http://docs.aws.amazon.com/config/latest/developerguide/evaluate-config_develop-rules.html

[65] https://aws.amazon.com/premiumsupport/trustedadvisor/

[66] https://www.youtube.com/watch?v=esEFaY0FDKc

[67] https://www.youtube.com/watch?v=DurzNeBQ-WU

[68] https://aws.amazon.com/devops/

[69] https://aws.amazon.com/devops/continuous-integration/

[70] https://aws.amazon.com/devops/continuous-delivery/

[71] https://aws.amazon.com/blogs/devops/

[72] https://d0.awsstatic.com/whitepapers/AWS_DevOps.pdf

[73] https://media.amazonwebservices.com/AWS_Operational_Checklists.pdf

[74] https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Best_Practices.pdf

[75] https://d0.awsstatic.com/whitepapers/compliance/AWS_Risk_and_Compliance_Whitepaper.pdf

[76] https://aws.amazon.com/premiumsupport/

[77] https://aws.amazon.com/premiumsupport/trustedadvisor/