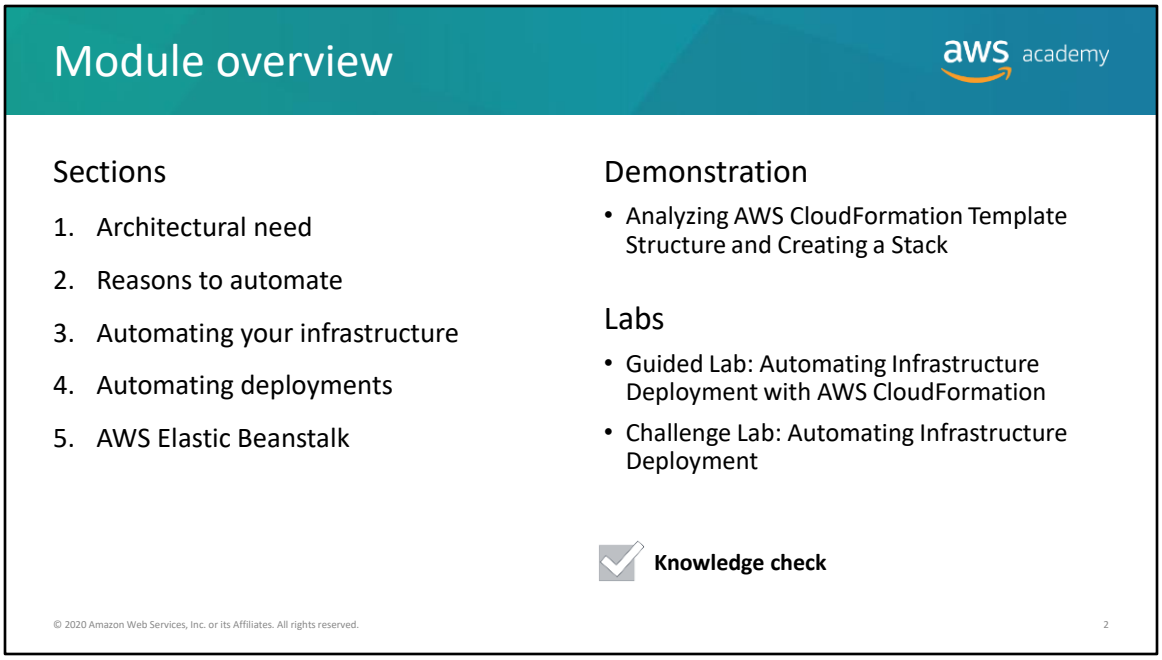
Welcome to Module 10: Automating Your Architecture.



This module contains the following sections:

1. Architectural need
2. Reasons to automate
3. Automating your infrastructure
4. Automating deployments
5. AWS Elastic Beanstalk

This module also includes:

- An educator-led *demonstration* that starts with an analysis of the structure of an AWS CloudFormation template, and then creates a stack from the template.
- A *guided lab* that provides hands-on practice with using AWS CloudFormation to create resources in your AWS account.
- A *challenge lab* where you use AWS CloudFormation to create Amazon Web Services (AWS) resources that support the Café use case.

Finally, you will be asked to complete a *knowledge check* that will test your understanding of key concepts covered in this module.

Module objectives

At the end of this module, you should be able to:

- Recognize when to automate and why
- Identify how to model, create, and manage a collection of AWS resources using AWS CloudFormation
- Use the Quick Start AWS CloudFormation templates to set up an architecture
- Indicate how to use AWS System Manager and AWS OpsWorks for infrastructure and deployment automation
- Indicate how to use AWS Elastic Beanstalk to deploy simple applications

At the end of this module, you should be able to:

- Recognize when to automate and why
- Identify how to model, create, and manage a collection of AWS resources using AWS CloudFormation
- Use the Quick Start AWS CloudFormation templates to set up an architecture
- Indicate how to use AWS System Manager and AWS OpsWorks for infrastructure and deployment automation
- Indicate how to use AWS Elastic Beanstalk to deploy simple applications

Module 10: Automating Your Architecture

Section 1: Architectural need

aws academy

Introducing Section 1: Architectural need.

Café business requirement

aws academy

The café now has locations in multiple countries and must start automating to keep growing. Their organization has many different architectures and needs a way to consistently deploy, manage, and update them.

Up to this point, the café created their AWS resources and configured their applications manually--mostly by using the AWS Management Console. This approach worked well as a way for the café to quickly develop a web presence and build out an infrastructure that supports the needs of employees and customers. However, they find it challenging to replicate their deployments to new AWS Regions so they can support new cafe locations in multiple countries.

They would also like to have separate development and production environments that reliably have matching configurations. They realize that they must start automating to support continued growth. Their organization has many different architectures, and it needs a way to consistently deploy, manage, and update these architectures quickly, consistently, and reliably.

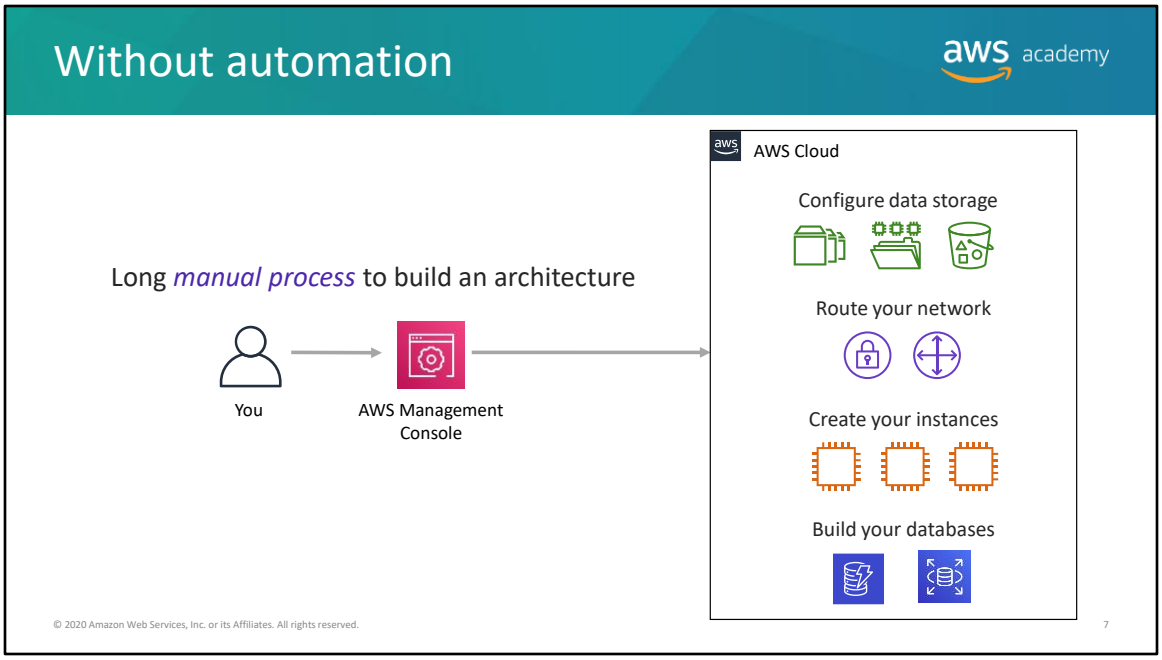
In this module, you will learn about AWS services that provide automation, including AWS CloudFormation. By using AWS CloudFormation, you will be able to help the cafe meet these new business requirements.

Module 10: Automating Your Architecture

Section 2: Reasons to automate

aws academy

Introducing Section 2: Reasons to automate.



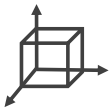
It takes significant time and energy to build a large-scale computing environment.

Many organizations will start using AWS by manually creating an Amazon Simple Storage Service (Amazon S3) bucket, or launching an Amazon Elastic Compute Cloud (Amazon EC2) instance and running a web server on it. Then, over time, they manually add more resources as they find that expanding their use of AWS can meet additional business needs. Soon, however, it can become challenging to manually manage and maintain these resources.

Some questions to ask include:

- Where do you want to put your efforts—into the design or the implementation? What are the risks of manual implementations?
- How would you ideally update production servers? How will you roll out deployments across multiple geographic regions? When things break—and they will—how will you manage the rollback to the last known good version?
- How will you debug deployments? Can you fix the bugs in your application before you roll the deployment out to the customer? How will you discover what is wrong and then fix it so that it *remains* fixed?
- How will you manage dependencies on the various systems and subsystems in your organization?
- Finally, is it realistic that you will be able to do all these tasks through manual configurations?

Risks from manual processes



Does not support repeatability at scale

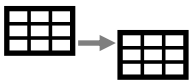
- How will you replicate deployments to multiple Regions?

No version control

- How will you roll back the production environment to a prior version?

Lack of audit trails

- How will you ensure compliance? How will you track changes to configuration details at the resource level?



Inconsistent data management

- For example, how will you ensure matching configurations across multiple Amazon Elastic Compute Cloud (Amazon EC2) instances?

Manually creating resources and adding new features and functionality to your environment *does not scale*. If you are responsible for a large corporate application, there might not be enough people to manually sail the ship.

Also, creating architecture and applications from scratch does not have inherent *version control*. If there is an emergency, it's helpful to be able to roll back the production stack to a previous version—but that is not possible when you create your environment manually.

Having an *audit trail* is important for many compliance and security situations. It's dangerous to allow anyone in your organization to manually control and edit your environments.

Finally, *consistency* is critical when you want to minimize risks. Automation enables you to maintain consistency.

Complying with AWS Well-Architected Framework principles

- Operational excellence design principles
  - Perform operations as code
  - Make frequent, small, reversible changes
- Reliability pillar design principles
  - Manage change in automation

Creating and maintaining AWS resources and deployments by following a manual approach does not enable you to meet these guidelines.

Also consider how well manual approaches comply with the design principles of the AWS Well-Architected Framework. One of the six design principles of operational excellence is to *perform operations as code*. In the cloud, you can apply the same engineering discipline that you use for application code to your entire environment. You can define your entire workload (applications, infrastructure, and other resources) as code and update it with code. You can script your operations procedures and automate when they run by triggering them in response to events. By performing operations as code, you limit human error and enable consistent responses to events.

Another operational excellence design principle is to *make frequent, small, reversible changes*. That is, design workloads to enable regular updates to components so that you can increase the flow of beneficial changes into your workload. Make changes in small increments that can be reversed if they do not help you identify and resolve issues that were introduced to your environment. When possible, try not to affect customers when you make these changes.

Finally, one of the design principles of the Well-Architected reliability pillar is to *manage change in automation*. Changes to your infrastructure should be done via automation. Therefore, the changes that must be managed are changes to the automation. Making changes to production systems is one of the largest risk areas for many organizations. In operations, use automation wherever it's practical, like for testing and deploying changes, adding or removing capacity, and migrating data.

Module 10: Automating Your Architecture

Section 3: Automating your infrastructure

aws academy

Introducing Section 3: Automating your infrastructure.

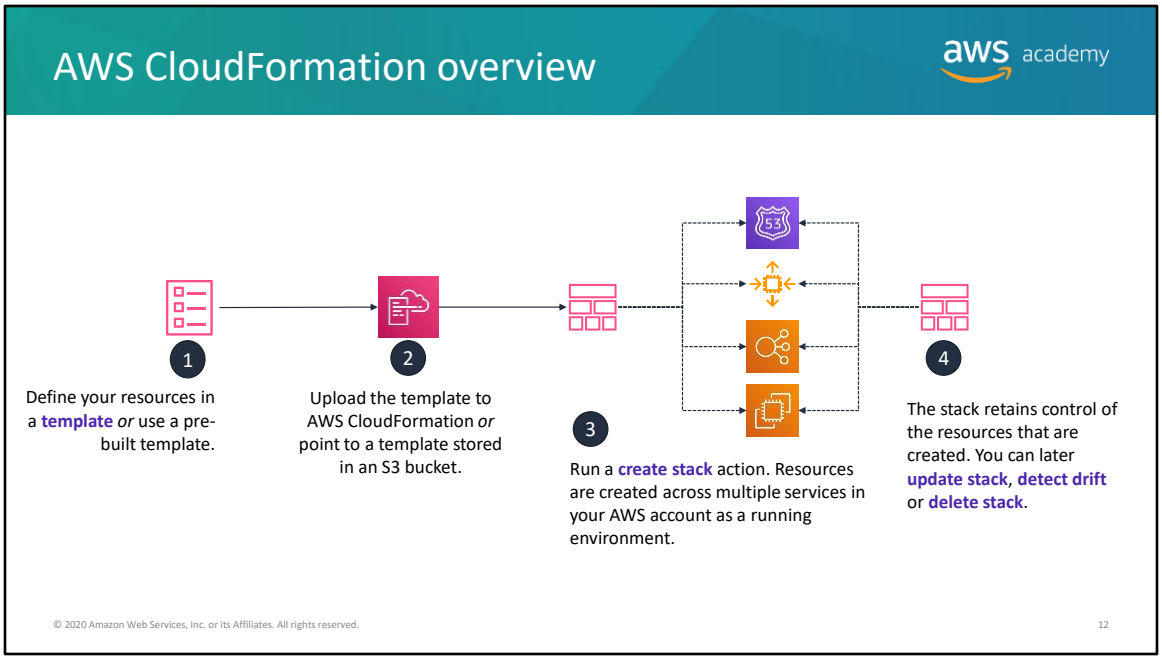
Automating your infrastructure

- AWS CloudFormation provides a simplified way to model, create, and manage a collection of AWS resources
  - Collection of resources is called an AWS CloudFormation stack
  - No extra charge (pay only for resources you create)
- Can create, update, and delete stacks
- Enables orderly and predictable provisioning and updating of resources
- Enables version control of AWS resource deployments

AWS CloudFormation provisions resources in a repeatable manner. It enables you to build and rebuild your infrastructure and applications without needing to perform manual actions or write custom scripts. With AWS CloudFormation, you author a document that describes what your infrastructure should be, including all the AWS resources that should be a part of the deployment. You can think of this document as a *model*. You then use the model to create the reality, because AWS CloudFormation can actually create the resources in your account.

When you use AWS CloudFormation to create resources, it is called an AWS CloudFormation *stack*. You create a stack, update a stack, or delete a stack. Thus, you can provision resources in an orderly and predicable way.

Using AWS CloudFormation enables you to treat your infrastructure as code (IaC). Author it with any code editor, check it into a *version control* system such as GitHub or AWS CodeCommit, and review files with team members before you deploy into the appropriate environments. If the AWS CloudFormation document that you create to model your deployment is checked in to a version control system, you could always delete a stack, check out an older version of the document, and create a stack from it. With version control, you can use essential rollback capabilities.



This diagram demonstrates how AWS CloudFormation works. First, you define the AWS resources that you want to create. In the example here, it creates a few EC2 instances, a load balancer, an Auto Scaling group, and an Amazon Route 53 hosted zone. You define the resources in an AWS CloudFormation *template*. You can create the template from scratch, or you can use a pre-built template. Many sample templates are also available.

Although AWS CloudFormation offers broad support for AWS services, not all resources can be created by AWS CloudFormation. See the list of resources that are supported by AWS CloudFormation for details.

Next, you upload the template to AWS CloudFormation. Alternatively, you can store the template on Amazon S3 and point AWS CloudFormation to the location where it is stored.

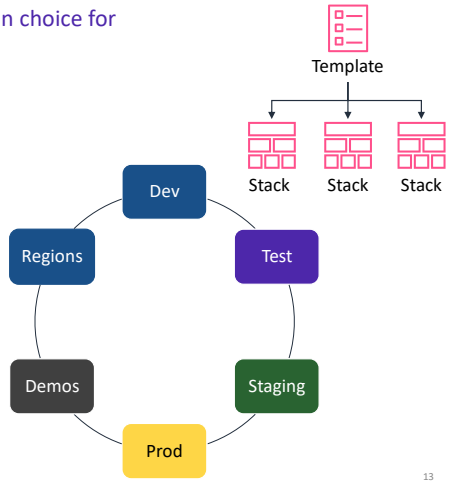
Third, you run the *create stack* action. When you do this, the AWS CloudFormation service reads through what is specified in the template and creates the desired resources in your AWS account. A single stack can create and configure resources in a single Region across multiple AWS services.

Finally, you can observe the progress of the stack-creation process. After the stack has successfully completed, the AWS resources that it created exist in your account. The stack object remains, and it acts like a handle to all the resources that it created. This is helpful when you want to take actions later. For example, you might want to *update the stack* (to create additional AWS resources or modify existing resources) or *delete the stack* (which will clean up and delete the resources that were created by the stack).

Infrastructure as code (IaC)

For AWS Cloud development, the built-in choice for IaC is AWS CloudFormation.

- IaC is the process of provisioning and managing your cloud resources by writing a template file that is –
  - Human readable
  - Machine consumable
- It is infrastructure you can replicate, re-deploy, re-purpose
- You can roll back to the last good state on failures

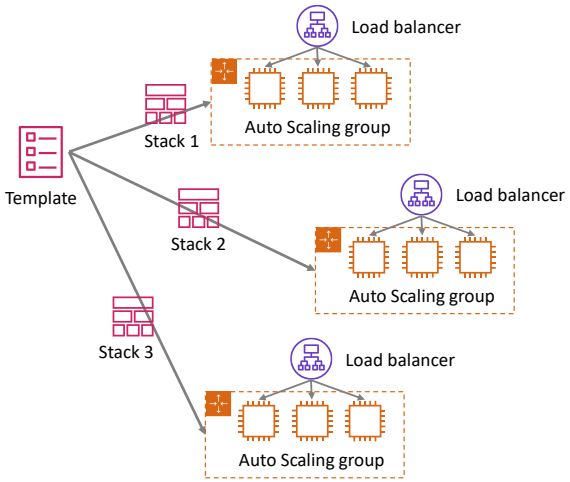


Infrastructure as code (IaC) is an industry term that refers to the process of provisioning and managing cloud resources by defining them in a template file that is both human-readable and machine-consumable.

IaC continues to grow in popularity because it provides a workable solution to challenges, like how to replicate, re-deploy, and re-purpose infrastructure, easily, reliably, and consistently.

The transactional nature of AWS CloudFormation is one of its biggest advantages from a customer perspective. AWS CloudFormation is transactional—the service will *roll back* to the last good state on failures.

Infrastructure as code: Benefits



Reduce multiple matching environments

- Rapid deployment of complex environments
- Provides configuration consistency
- Simple clean up when wanted (deleting the stack deletes the resources created)
- Easy to propagate a change to all stacks
  - Modify the template, run update stack on all stacks

Benefits

- Reusability
- Repeatability
- Maintainability

Now, consider some of the benefits of IaC in more detail. If you build infrastructure with code, you gain benefits like the ability to rapidly deploy complex environments. With one template (or a combination of templates), you can build the same complex environments repeatedly.

In the example here, a single template can be used to create three different stacks. Each stack can be created rapidly, usually in a matter of minutes. Each stack replicates complex configuration details consistently.

If Stack 2 is your test environment and Stack 3 is your production environment, you can have greater confidence that if your test jobs performed well in the test environment, that they will also perform well in the production environment. The template minimizes the risk that the test environment is configured differently from the production environment.

Also, if you must make a configuration update in the test environment, you update the template with the change and update all the stacks. This process helps ensure that the modifications to a single environment will be reliably propagated to all environments that should receive the update.

Another benefit is that it easier to *clean up* all the resources that were created in your account to support a test environment after you no longer need them. This helps reduce cost associated with no resources that you no longer need, and helps keep your account clean of cruft.

AWS CloudFormation template syntax

aws academy

AWS CloudFormation templates

- Author in JavaScript Object Notation (JSON) or YAML Ain't Markup Language (YAML)
- YAML advantages –
  - Less verbose (no {},"", characters)
  - Supports embedded comments
- JSON advantages –
  - More widely used by other computer systems (for example, APIs)
- Recommendation – Treat templates as source code
  - Store them in a code repository

```
{
 "AWSTemplateFormatVersion": "2010-09-09",
 "Resources" : {
 "awsexamplebucket1" : {
 "Type" : "AWS::S3::Bucket"
 }
 }
}
```

JSON example

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
 awsexamplebucket1:
 Type: AWS::S3::Bucket
```

YAML example

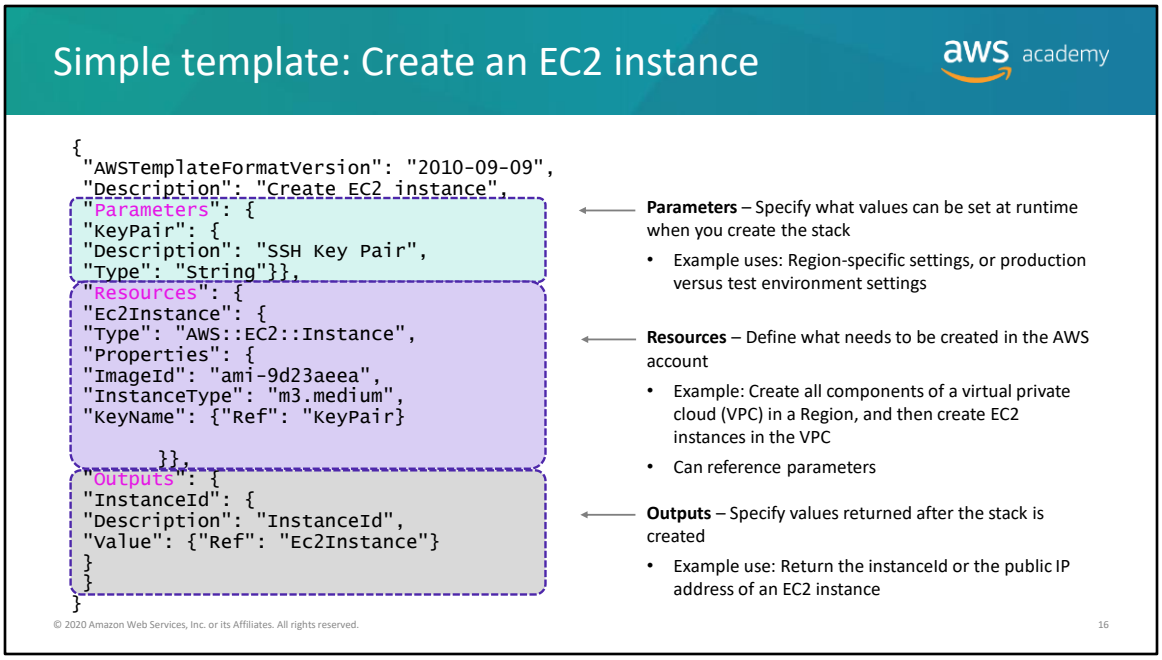
Templates can also be authored in the AWS CloudFormation Designer—a graphical design interface in the AWS Management Console.

An AWS CloudFormation template can be authored in either JavaScript Object Notation (JSON) or YAML Ain't Markup Language (YAML).

YAML is optimized for readability. The same data that is stored in JSON will take fewer lines to store in YAML because YAML does not use braces ({}), and it uses fewer quotation marks (“”). Another advantage of YAML is that it natively supports embedded comments. You might find it easier to debug YAML documents compared with JSON. With JSON, it can be difficult to track down missing or misplaced commas or braces. YAML does not have this issue.

Despite the many advantages of YAML, *JSON* offers some unique advantages. First, it is widely used by computer systems. Its ubiquity is an advantage because data that is stored in JSON can reliably be used with many systems without needing transformation. Also, it is usually easier to generate and parse JSON than it is to generate and parse YAML.

The AWS Management Console provides a graphical interface, called the *AWS CloudFormation Designer*, which can be used to author or view the contents of AWS CloudFormation templates. It can also be used to convert a valid JSON template to YAML or to convert YAML to JSON. It provides a drag-and-drop interface for authoring templates that can be output as either JSON or YAML.

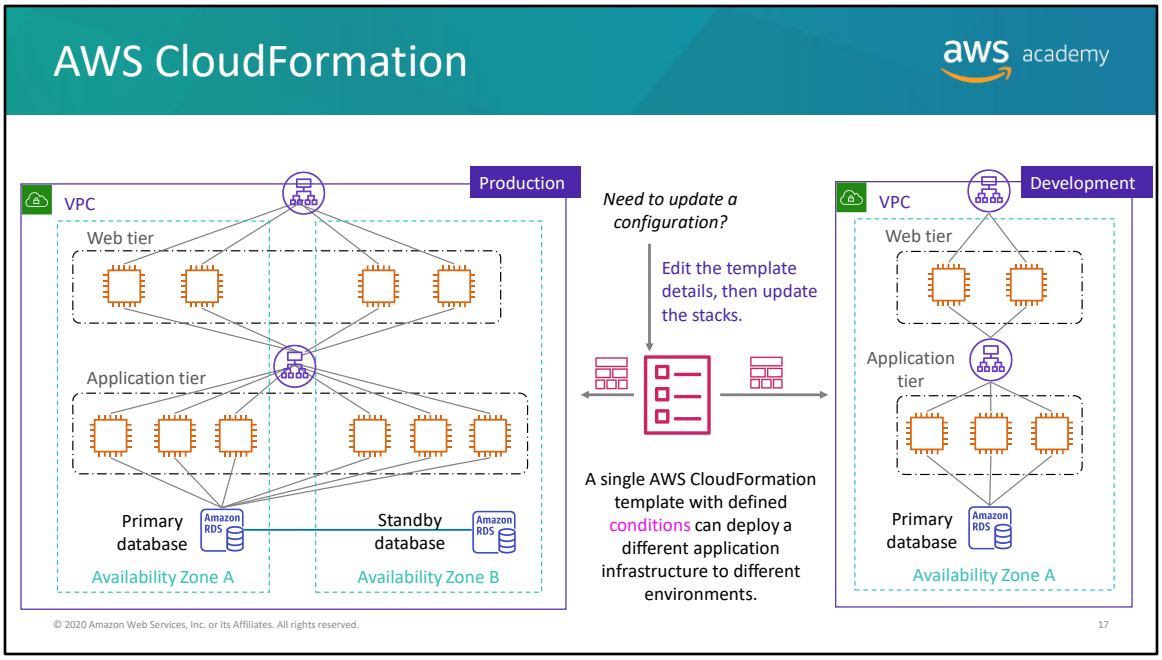


This example AWS CloudFormation template creates an EC2 instance. Although this example does not illustrate all possible sections of a template, it does highlight some of the most commonly used sections, including parameters, resources, and outputs.

Parameters is an optional section of the template. Parameters are values that are passed to your template at runtime (when you create or update a stack). You can refer to parameters from the **Resources** and **Outputs** sections of the template. A parameter's name and description appear in the *Specify Parameters* page when a user launches the *Create Stack* wizard in the console.

Resources is a required section for any template. Use it to specify the AWS resources to create, along with their properties. In this example, a resource of type *`AWS::EC2::Instance`* is specified, which creates an EC2 instance. The example resource includes both statically defined properties (*`ImageId`* and *`InstanceType`*) and a reference to the *`KeyPair`* parameter.

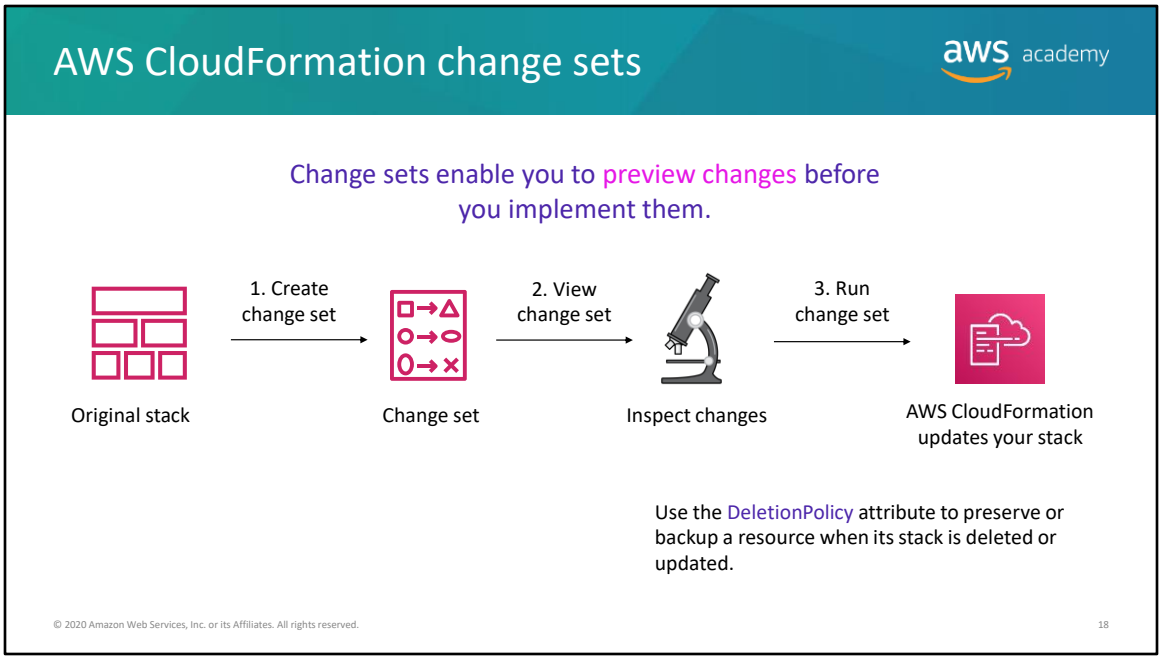
Finally, the example shows the **Outputs** section. **Outputs** describes the values that are returned when you view your stack's properties. In the example, an `InstanceId` output is declared. After the stack is created, you can see this value in the stack details in the AWS CloudFormation console, by running the *`aws cloudformation describe-stacks`* AWS Command Line Interface (AWS CLI) command, or by using the AWS software development kits (SDKs) to retrieve this value.



You can use the same AWS CloudFormation template to create both your production environment and development environment. This approach can help ensure that (for example) the same application binaries, same Java version, and same database version are used in both development and production. Thus, the template can help ensure that your application behaves in production the same way that it behaved in the development environment.

In the example, you see that both the production and development environments are created from the same template. However, the production environment is configured to run across two Availability Zones and the development environment runs in a single Availability Zone. These kinds of deployment-specific differences can be accomplished by using *conditions*. You can use a *Conditions* statement in AWS CloudFormation templates to help ensure that—though they are different in size and scope—development, test, and production environments are otherwise configured identically.

You might need several testing environments for functional testing, user acceptance testing, and load testing. Creating those environments manually is risky. However, creating them with AWS CloudFormation helps ensure consistency and repeatability.



One way to update a stack (and thus update your AWS resources) is to update the AWS CloudFormation template that you used to create the stack, and then run the **Update Stack** option.

However, you might want to gain additional insight about the specific changes that AWS CloudFormation will implement if you run that command—before you actually run an update. If you want this type of insight, you can use a *change set*.

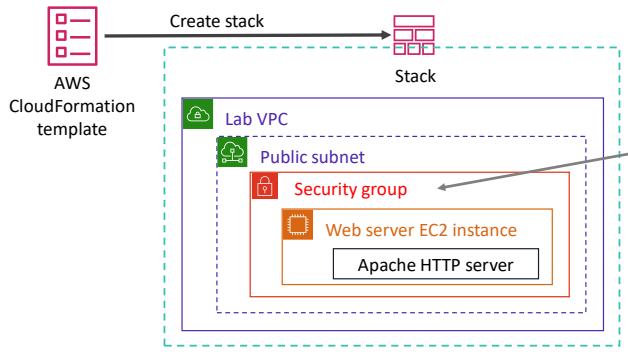
Change sets enable you to preview the changes, verify that they align with your expectations, and then approve the updates before you proceed.

Follow this basic workflow to use AWS CloudFormation change sets:

1. **Create a change set** by submitting changes for the stack that you want to update.
2. **View the change set** to see which stack settings and resources will change. If you want to consider other changes before you decide which changes to make, create additional change sets.
3. **Run the change set**. AWS CloudFormation updates your stack with those changes.

If you use change sets, you might want to set a deletion policy on some resources. The *DeletionPolicy* attribute can be used to preserve (or, in some cases, back up) a resource when its stack is deleted or updated. If a resource has no DeletionPolicy attribute, AWS CloudFormation deletes the resource.

Drift detection



Scenario:

1. An application environment is created by an AWS CloudFormation stack.
2. Later, someone ***manually modifies* the security group** and opens a new inbound TCP port.
3. Drift detection is run on the stack.
4. All resources except the security group show the result IN_SYNC, but the security group shows a status of MODIFIED, with details.

Question: In this scenario, what would be a better approach if the team wants to modify the security group setting?

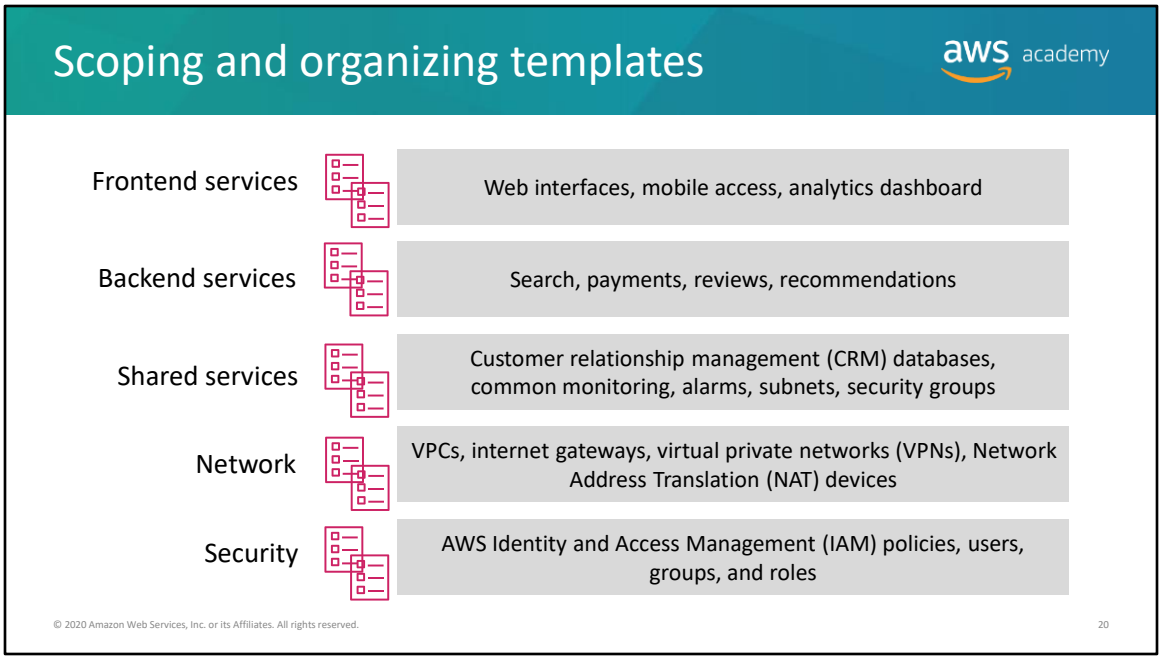
Answer: Modify the AWS CloudFormation template security group settings. Then, run Update Stack. AWS CloudFormation will update the security group. Keeps the *model* deployment synchronized with the actual deployment.

Consider this scenario. An application environment is created by running an AWS CloudFormation stack. Then, someone decides to manually modify the deployed environment settings. They create a new inbound rule in the security group that was created by the stack. However, they make this change outside the context of AWS CloudFormation—for example, by using the Amazon EC2 console. As the architect of this application, you would want to know that your *deployed* environment no longer matches the *model* environment that is defined in the AWS CloudFormation template.

How would you know which resources were modified so that they no longer exactly conform with the specifications in the stack?

Drift detection can be run on a stack by choosing **Detect Drift** from the **Stack actions** menu in the console. Performing drift detection on a stack shows you whether the stack has *drifted* from its expected template configuration. Drift detection returns detailed information about the drift status of each resource that supports drift detection in the stack.

When you delete a stack that has drift, the drift is not handled by the AWS CloudFormation resource cleanup process. If the stack has unresolved resource dependencies, they might cause the delete stack action to fail. In such cases, you might need to manually resolve the issue. For details, see the list of resources that support drift detection.



As your organization starts to use more AWS CloudFormation templates, it will be important for you to have a strategy for templates. This strategy will define the scope of what a single template should create, and the general characteristics that would make you want to define your AWS infrastructure in more than one template.

The diagram offers some ideas about how you might organize your templates so that they are easier to maintain, and so that they can be used in combination with each other in a way that makes sense. A good strategy is to group your resource definitions in templates similar to the way that you would organize the functionality of a large enterprise application into different parts.

Think about the more tightly connected components of your infrastructure, and consider putting them in the same templates. In this example, AWS CloudFormation templates are scoped to create and maintain AWS resources in one of five areas: frontend services, backend services, shared services, network, and security. In each area, you might maintain a template that is scoped to a single application or a single department's needs.

Regardless of how you organize and scope each AWS CloudFormation template, treat your templates as code that needs version control. Store your templates in a source control system.

AWS Quick Starts

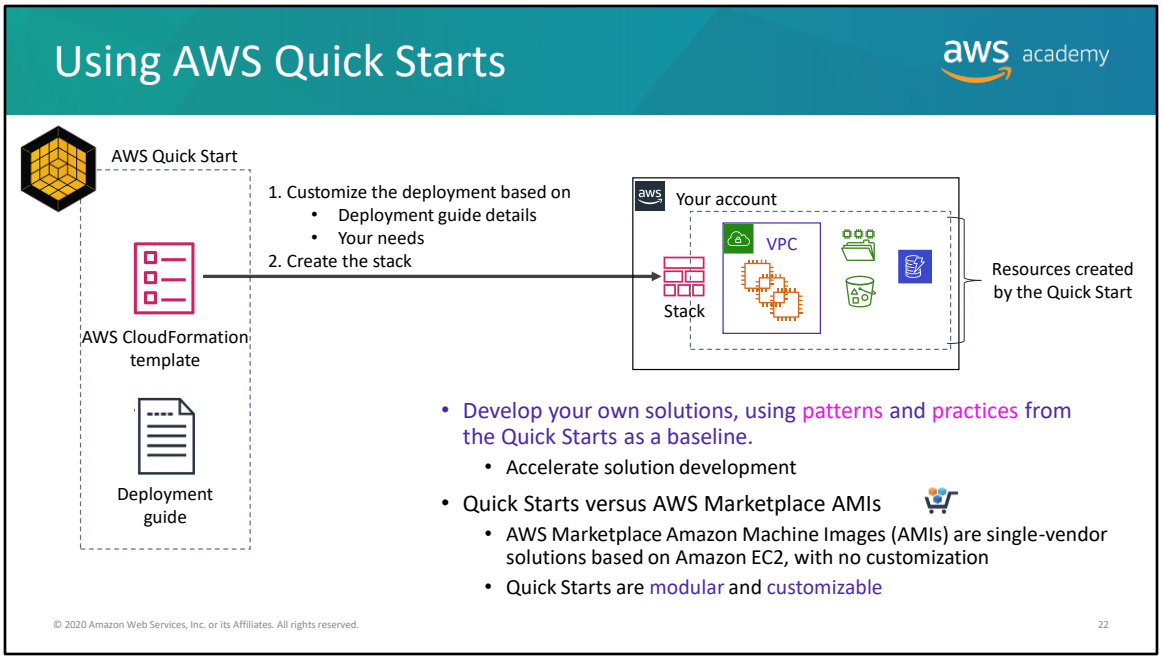
AWS Quick Starts

AWS CloudFormation templates built by AWS solutions architects

- Are gold-standard deployments
- Are based on AWS best practices for security and high availability
- Can be used create entire architectures with one click in less than an hour
- Can be used for experimentation and as the basis for your own architectures

AWS Quick Starts provide AWS CloudFormation templates. The Quick Starts are built by AWS solutions architects and partners to help you deploy popular solutions on AWS, based on AWS best practices for security and high availability. These reference deployments can be provisioned in your AWS account in less than an hour, often in only minutes. They help you build Well-Architected test or production environment in a few steps. You can use them to create entire architectures, or you can use them to experiment with new deployment approaches.

Go to the AWS Quick Starts page for details.



Each Quick Start consists of an AWS CloudFormation template and a deployment guide. The guide provides details about deployment options and how to configure the deployment to match your needs.

Customize the deployment to match your needs and create the stack. Depending on the AWS resources that must be created, the Quick Start will finish deploying in a matter of minutes or a few hours.

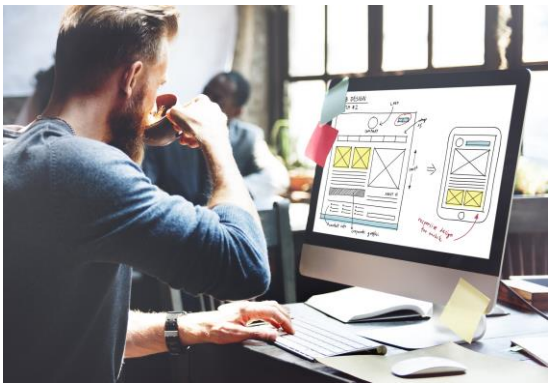
Even if you do not use the Quick Starts, you may find it helpful to look at a few of them to see the types of *patterns* and *practices* that the Quick Starts follow. You might find that they help accelerate your own template development when you borrow a section of a Quick Start and embed it in your own template.

AWS Marketplace Amazon Machine Images (AMIs) are another solution that people sometimes use. These resources can be launched from the Amazon EC2 console. AWS Marketplace AMIs provide single-vendor solutions that run on EC2 instances. In contrast, AWS Quick Starts are modular and more customizable solutions that might (or might not) use Amazon EC2.

Demonstration: Analyzing AWS CloudFormation Template Structure and Creating a Stack

The educator might choose to demonstrate the structure of an AWS CloudFormation template and then create an AWS CloudFormation stack by using the template.

Module 10 – Guided Lab: Automating Infrastructure Deployment with AWS CloudFormation



You will now complete Module 10 - Guided Lab: Automating Infrastructure Deployment with AWS CloudFormation.

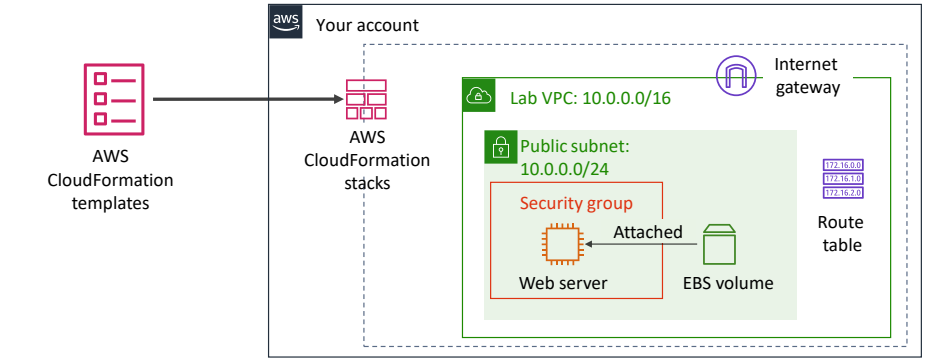
Guided lab: Tasks

1. Deploying a networking layer
2. Deploying an application layer
3. Updating a stack
4. Exploring templates with AWS CloudFormation Designer
5. Deleting the stack

In this guided lab, you will complete the following tasks:

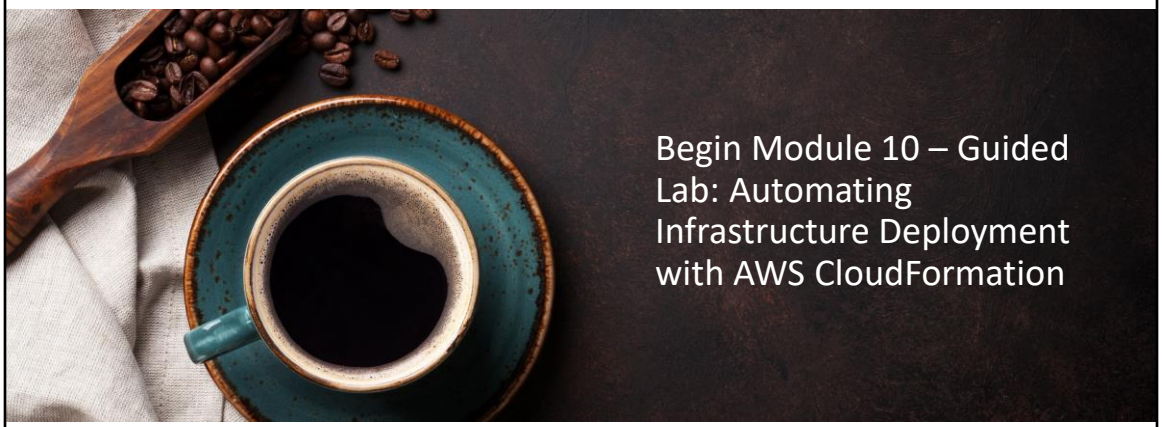
1. Deploying a networking layer
2. Deploying an application layer
3. Updating a stack
4. Exploring templates with AWS CloudFormation Designer
5. Deletimg the stack

Guided lab: Final product



By the end of this guided lab, you will have used AWS CloudFormation to create the resources in the diagram. The resources in the network layer are created when you create the first stack. The EC2 instance, security group, and Amazon Elastic Block Store (Amazon EBS) volume are created when you create the second stack. The security group settings are then updated in the template that is used to create the second stack. This modification is applied to the security group resource when you run the update stack action.

~ 20 minutes



Begin Module 10 – Guided Lab: Automating Infrastructure Deployment with AWS CloudFormation

It is now time to start the guided lab.

Guided lab debrief: Key takeaways

Your educator might choose to lead a conversation about the key takeaways from the guided lab after you have completed it.

Section 3 key takeaways

- AWS CloudFormation is an infrastructure as code (IaC) service that enables you to model, create, and manage a collection of AWS resources
- AWS CloudFormation IaC is defined in templates that are authored in JSON or YAML
- A stack is what you create when you use a template to create AWS resources
- Actions that are available on an existing stack include update stack, detect drift, and delete stack
- AWS Quick Starts provide AWS CloudFormation templates that are built by solutions architects that that reflect AWS best practices

Some key takeaways from this section of the module include:

- AWS CloudFormation is an infrastructure as code (IaC) service that enables you to model, create, and manage a collection of AWS resources
- AWS CloudFormation IaC is defined in templates that are authored in JSON or YAML
- A stack is what you create when you use a template to create AWS resources
- Actions that are available on an existing stack include update stack, detect drift, and delete stack
- AWS Quick Starts provides AWS CloudFormation templates that are built by solutions architects and that reflect AWS best practices

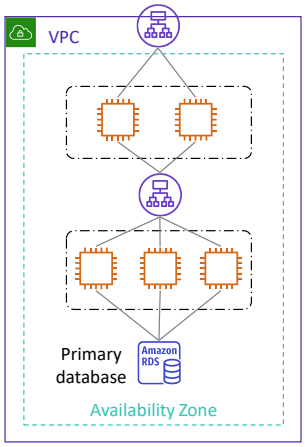
Module 10: Automating Your Architecture

Section 4: Automating deployments

aws academy

Introducing Section 4: Automating deployments.

How will you keep your fleet updated?



You might have hundreds of instances to manage.
How do you apply guest operating system (OS) patches and update the software that is installed on them?

In the previous section, you learned how to create an entire infrastructure automatically by using AWS CloudFormation. This capability is powerful, but there are still some important questions to ask.

How will you update the software on your fleet of EC2 instances? Are you supposed to remotely log in to each instance and run update commands yourself? How will you revert a change if something goes wrong? What if you have hundreds or even thousands of servers that run many different applications?

Traditional tools can help with these scenarios, but a ready to use solution would be more convenient.

AWS Systems Manager

aws academy

Gain operational insights and take action on AWS resources.

- Automates operational tasks
  - Example: Apply OS patches and software upgrades across a fleet of EC2 instances
- Simplifies resource and application management
  - Manage software inventory
  - View detailed system configurations across the fleet
- Manages servers on-premises and in the cloud

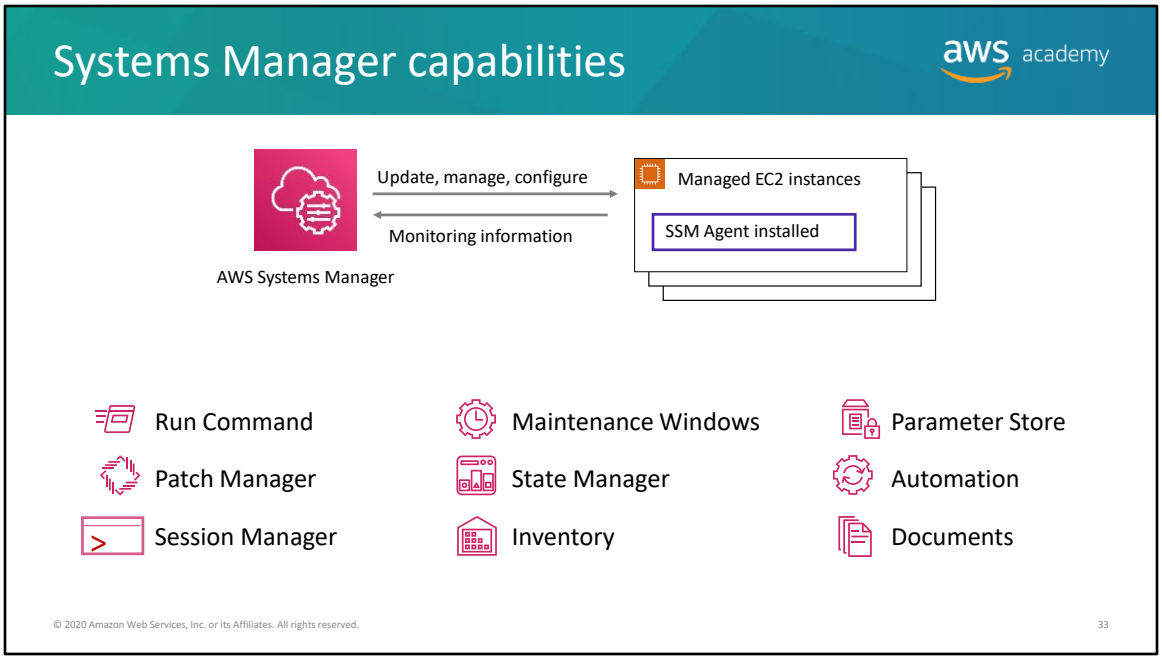
Even if you use an IaC tool like AWS CloudFormation to create and maintain your AWS resource deployments, it is helpful to have other tools that you can use. For example, these tools can address the environment's ongoing needs for configuration management. These needs can occur both after infrastructure resources are provisioned and after the infrastructure is up and running. AWS Systems Manager is a service that addresses this challenge.

AWS Systems Manager is a management service that is designed to be highly focused on automation. It enables the configuration and management of systems that run on-premises or in AWS. AWS Systems Manager enables you to identify the instances that you want to manage, and then define the management tasks that you want to perform on those instances. AWS Systems Manager is available at no cost, and it can manage both your Amazon EC2 and on-premises resources.

Some tasks you that can accomplish with AWS Systems Manager include:

- Collecting software inventory
- Applying operating system (OS) patches
- Creating system images
- Configuring Microsoft Windows and Linux operating systems

These capabilities help you define and track system configurations, prevent drift, and maintain the software compliance of your Amazon EC2 and on-premises configurations.



This example shows how Systems Manager can be used to update, manage, and configure a fleet of EC2 instances.

You can install an *AWS Systems Manager Agent (SSM Agent)* on an EC2 instance, or even on an on-premises server, or a virtual machine (VM). Once the SSM Agent is installed, it will be possible for Systems Manager to update, manage, and configure the server on which it is installed. The agent processes requests from Systems Manager and then runs them in accordance with the specification provided in the request. The agent then sends status and relevant information back to Systems Manager.

SSM Agent is preinstalled, by default, on most Microsoft Windows Server AMIs, all Amazon Linux and Amazon Linux 2 AMIs, and some Ubuntu AMIs. However, you must manually install the agent on EC2 instances that were created from other Linux AMIs. For full details, see the Working with SSM Agent AWS documentation.

AWS Systems Manager provides various tools:

- **Run Command** enables you to remotely and securely manage the configuration of your managed instances. Commands can run without Secure Shell (SSH) or Remote Desktop Protocol (RDP) access, so you can use them to reduce the need for a bastion host. Run

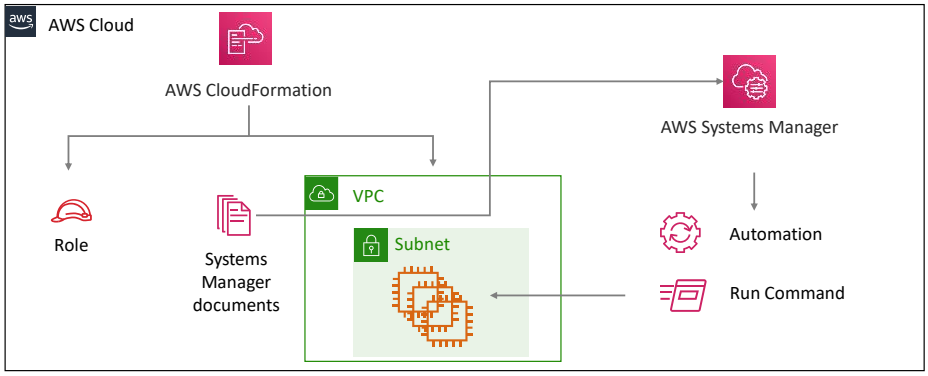
Bash, PowerShell, Salt, or Ansible scripts.

- **Maintenance Windows** enable you to define a schedule for performing potentially disruptive actions on your instances. Examples include patching an operating system, updating drivers, or installing software or patches.
- **Parameter Store** provides secure storage for configuration data and secrets management. For example, you can store passwords, database strings, and license codes as parameter values.
- **Patch Manager** automates the process of patching managed instances with both security-related updates and other types of updates.
- **State Manager** automates the process of keeping your Amazon EC2 and hybrid infrastructure in a state that you define.
- **Automation** enables you to built automation workflows to configure and manage instances and AWS resources.
- **Session Manager** enables you to manage your EC2 instances through an interactive, browser-based shell.
- **Inventory** provides visibility into your Amazon EC2 and on-premises computing environments. You can use Inventory to collect metadata from your managed instances.
- **Documents** define the actions that Systems Manager performs on your managed instances. You can use more than a dozen pre-configured documents by specifying parameters at runtime. You can also define your own documents in JSON or YAML, and specify steps and parameters.

AWS CloudFormation and Systems Manager complement each other

AWS CloudFormation works well for defining AWS Cloud resources.

Systems Manager works well for automating within guest operating systems.



Now that you learned about the features of both AWS CloudFormation and AWS Systems Manager, consider how the two services complement each other.

Systems Manager works well for automating *within* a guest OS. In contrast, AWS CloudFormation works well for defining AWS Cloud resources.

You can use AWS CloudFormation at the AWS Cloud layer to define AWS resources. As the diagram demonstrates, you can then use AWS Systems Manager to configure the OS of the instances that were created by the AWS CloudFormation stack.

By maintaining the cloud resources with AWS CloudFormation, you keep the ability to deploy a stack and then delete a stack from a single template. In contrast, Systems Manager gives you a way to perform ongoing tasks, such as updating the EC2 instance guest OS with patch updates and centrally aggregating logs to Amazon CloudWatch.

For more information about an example solution that uses these two services together, see the Using AWS Systems Manager Automation and AWS CloudFormation together blog post.

AWS OpsWorks

AWS OpsWorks is a configuration management service.

- Automate how servers are configured, deployed, and managed
- Provides managed instances of Chef and Puppet
  - Chef and Puppet are popular automation platforms
- Three versions available –
  - AWS OpsWorks for Chef Automate
  - AWS OpsWorks for Puppet Enterprise
  - AWS OpsWorks Stacks

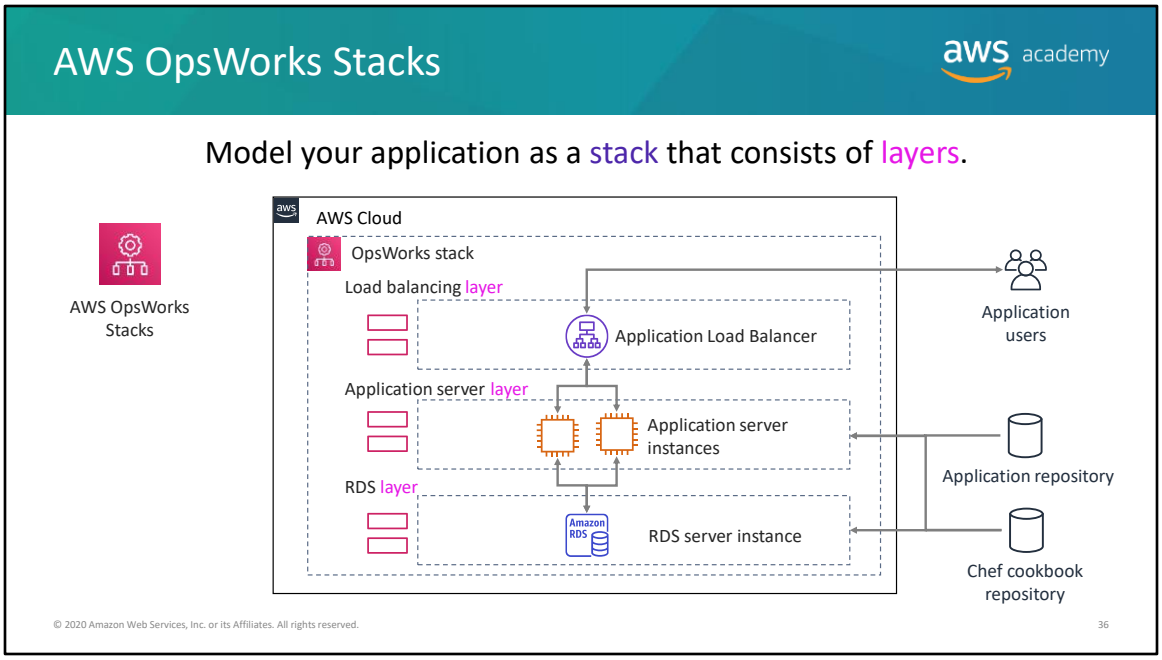
AWS OpsWorks is a service for configuration management. You can use OpsWorks to automate how EC2 instances are configured, deployed, and managed.

AWS OpsWorks comes in three different versions:

- **AWS OpsWorks for Chef Automate** provides a fully managed Chef Automate server that provides workflow automation for continuous deployment, and automated testing for compliance and security. The Chef Automate platform handles operational tasks, such as software and operating system configurations, continuous compliance, package installations, database setups, and more. You can use Chef Automate to create and manage dynamic infrastructure that runs on the AWS Cloud. A Chef Automate server manages the configuration of nodes in your environment by telling the chef-client which Chef recipes to run on the nodes. It also stores information about nodes, and serves as a central repository for your Chef cookbooks.

- **AWS OpsWorks for Puppet Enterprise** provides a managed Puppet Enterprise server and a suite of automation tools that provide workflow automation for orchestration, automated provisioning, and visualization for traceability. With Puppet Enterprise, you can define configurations for your servers in a format that you can maintain and version like your application source code. The primary Puppet servers are designed to consistently configure and maintain your other Puppet servers (or nodes). You can also configure your nodes dynamically based on the state of other nodes.

- **AWS OpsWorks Stacks** is a configuration management service that helps you configure and operate applications of all kinds and sizes by using Chef. You can define the application's architecture and the specification of each component—including package installation, software configuration, and resources (such as storage).



This example demonstrates how a basic application might be managed with AWS OpsWorks Stacks. The fundamental unit of creation in an OpsWorks Stacks application is the *stack*.

After a stack is created, you can add multiple *layers* to that stack. You can thus build out your application as a set of interacting layers of related functionality.

In this case, a group of application servers runs in an *application server layer*. The applications run behind an Elastic Load Balancing load balancer that is defined in a *load balancing layer*. This example also includes a backend Amazon Relational Database Service (Amazon RDS) database server, which is defined in an *RDS layer*.

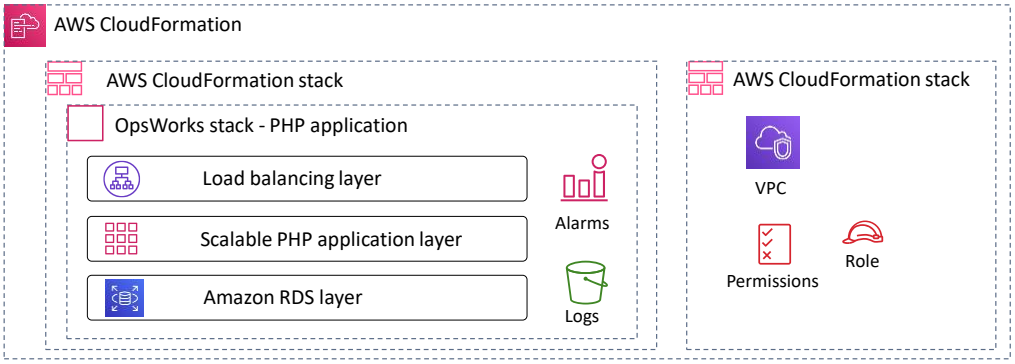
Layers depend on Chef recipes to handle tasks like installing packages on instances, deploying applications, running scripts, and so on. OpsWorks Stacks uses Chef cookbooks to handle tasks like installing and configuring packages and deploying applications. Your custom cookbooks must be stored in an online repository—an archive (such as a .zip file), or a source control manager (such as Git).

One key OpsWorks Stacks feature is a set of *lifecycle events*—including Setup, Configure, Deploy, Undeploy, and Shutdown—which automatically run a specified set of recipes at the appropriate time on each instance.

Each layer can have a set of recipes that are assigned to each lifecycle event. These recipes handle various tasks for that event and layer.

OpsWorks Stacks complements AWS CloudFormation

1. Use AWS CloudFormation to create the infrastructure (VPC, IAM roles, and so on).
2. Deploy the application layer with OpsWorks Stacks.



Because OpsWorks Stacks can be created via AWS CloudFormation, the use of the two technologies is complementary.

For example, you could use one AWS CloudFormation template to create the AWS resources infrastructure for your environment, including a VPC. You could then use another AWS CloudFormation template to create the OpsWorks stack that will be deployed in that VPC. After both AWS CloudFormation stacks are created in your account, you can use the OpsWorks stack to manage the application.

Module 10: Automating Your Architecture

Section 5: AWS Elastic Beanstalk

aws academy

Introducing Section 5: AWS Elastic Beanstalk.

General challenges

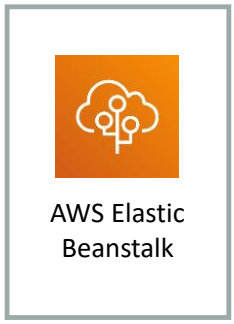
aws academy

- Managing infrastructure around application deployment can be *difficult*
- It can be *time-consuming* to manage and configure servers
- You might have *lack of consistency* across multiple projects or applications

Now, consider some general challenges that you might face as you approach managing your cloud infrastructure.

First, consider that deploying an application can be *difficult*. How will you ensure that it is highly available and that it can support user requests even during peak usage? How can you make sure that the application is resilient, and that regular backups are taken for disaster recovery (DR) purposes? It can be *time-consuming* to manage and configure servers. Meanwhile, you would like to maintain *consistency* across projects and applications, but this consistency can be difficult to achieve.

AWS Elastic Beanstalk



- Easy way to get web applications up and running
- Managed service that automatically handles –
  - Infrastructure provisioning and configuration
  - Deployment
  - Load balancing
  - Automatic scaling
  - Health monitoring
  - Analysis and debugging
  - Logging

- No additional charge for using it
  - Pay only for the underlying resources that are used

AWS Elastic Beanstalk is another AWS compute service option. It is a platform as a service (PaaS) offering that facilitates the quick deployment, scaling, and management of your web applications and services. It addresses many of the challenges that you just learned about.

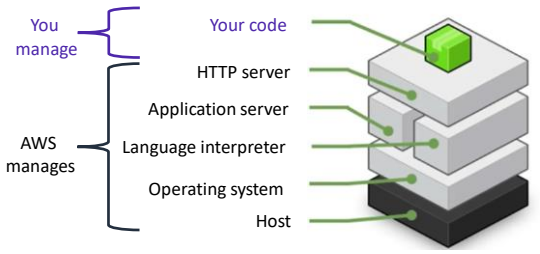
With Elastic Beanstalk, you remain in control of your code, while AWS maintains the underlying infrastructure. The required AWS resources are created and deployed by using a simple wizard in the AWS Management Console. The wizard asks you to choose the instance type and size, the database type and size, and what automatic scaling settings you would like to use. It provides you access to the server log files, and enable Secure HTTP (HTTPS) on the load balancer.

You upload your code and Elastic Beanstalk automatically handles the deployment—including capacity provisioning, load balancing, automatic scaling, and monitoring application health. At the same time, you retain full control over the AWS resources that power your application, and you can access the underlying resources at any time.

There is no additional charge for AWS Elastic Beanstalk. You pay for the AWS resources that you create to store and run your application, such EC2 instances or S3 buckets. You only pay for what you use, when you use it.

AWS Elastic Beanstalk deployments

- It supports web applications written for common platforms
  - Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker
- You upload your code
  - Elastic Beanstalk automatically handles the deployment
  - Deploys on servers such as Apache, NGINX, Passenger, Puma, and Microsoft Internet Information Services (IIS)

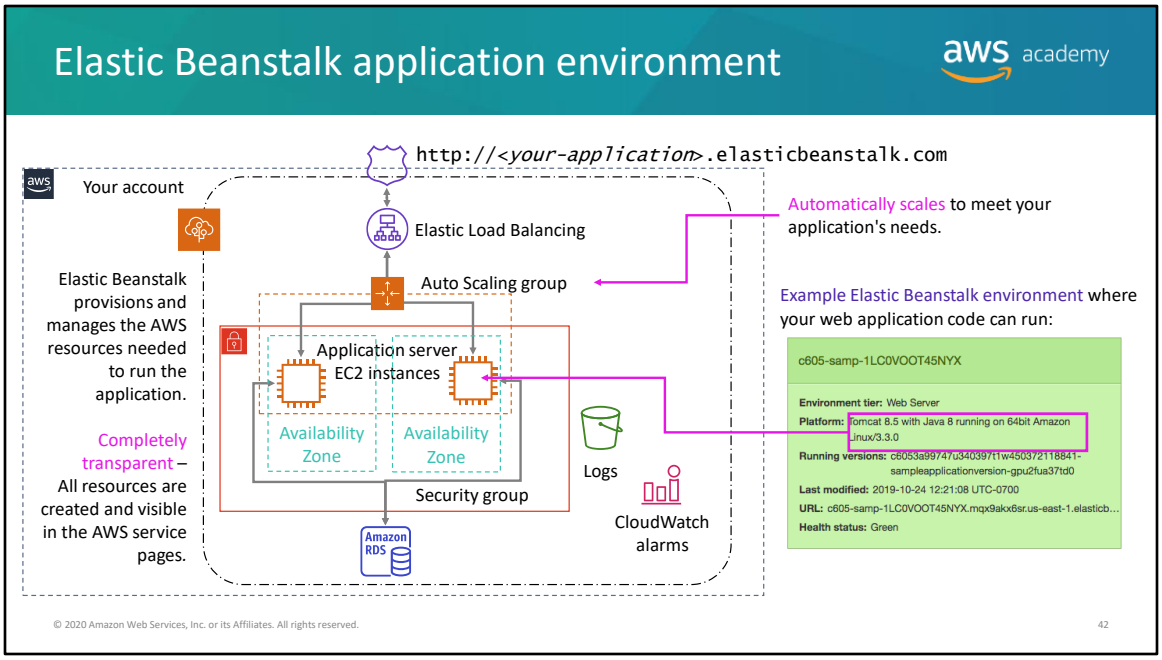


Elastic Beanstalk configures each EC2 instance in your environment with the components that are needed to run applications for the selected platform. You don't need to worry about logging in to instances to install and configure your application stack.

The only thing that you must create is your code. Elastic Beanstalk is designed to make deploying your application a quick and easy process. It supports a range of platforms, including **Docker**, **Go**, **Java**, **.NET**, **Node.js**, **PHP**, **Python**, and **Ruby**.

AWS Elastic Beanstalk deploys your code on:

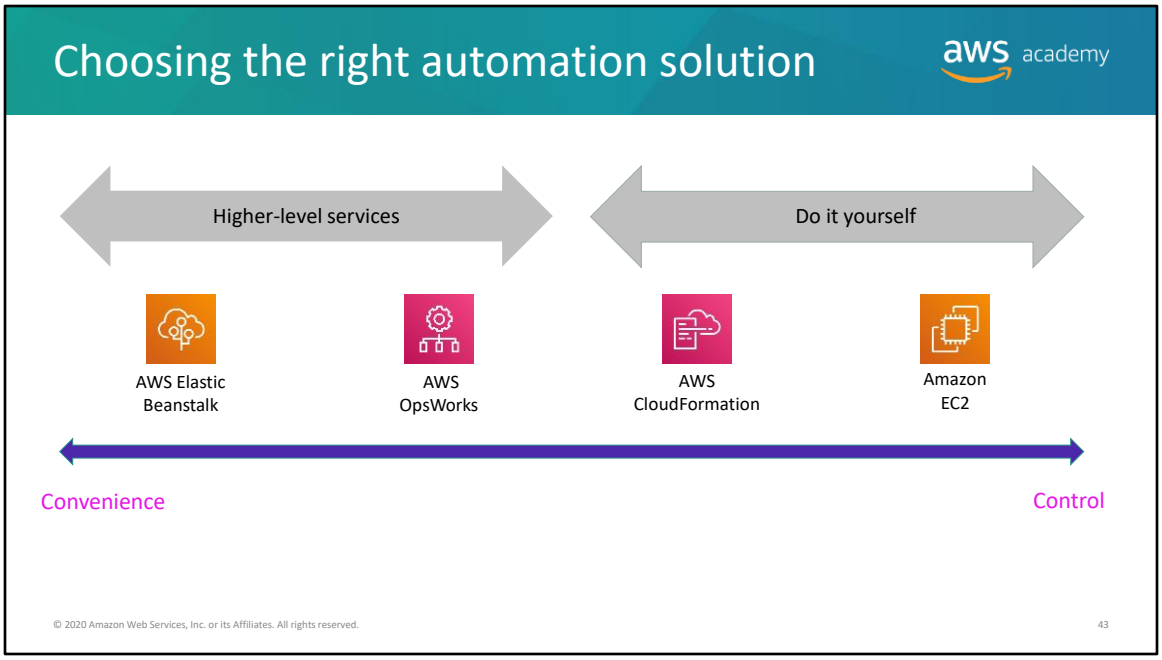
- **Apache Tomcat** for Java applications
- **Apache HTTP Server** for PHP and Python applications
- **NGINX** or **Apache HTTP Server** for Node.js applications
- **Passenger** or **Puma** for Ruby applications
- **Microsoft Internet Information Services (IIS)** for .NET, Java SE, Docker, and Go applications.



You can choose from two types of environments when you work with Elastic Beanstalk. The *single-instance environment* enables you to launch a single EC2 instance and it does not include load balancing or automatic scaling. The other type of environment—which is in this example—can launch *multiple EC2 instances,* and it includes load balancing and an automatic scaling configuration. A managed database layer is optional.

The AWS resources that are created by Elastic Beanstalk are visible in your AWS account. For example, after you create an Elastic Beanstalk application, open the AWS Management Console and then open the Amazon EC2 console. You will see the instances that Elastic Beanstalk is managing on your behalf. In the example, two EC2 instances were created. Each EC2 instance runs the Amazon Linux guest OS, with Java 8 and the Apache Tomcat 8.5 web server installed. Your application code will run on these servers. Automatic scaling is configured, so if the load starts to stress the resources on these two instances (such as excess CPU utilization for more than 5 minutes), more application server instances will be launched automatically. This example also shows that an RDS database instance is available and accessible from the EC2 instances. You can store your application data in the database and use structured query language (SQL) in your application code to access and update this data. Elastic Beanstalk manages the database instance and helps maintain connectivity between the EC2 instances and the database.

Elastic Beanstalk creates and manages scalable environments. You can configure the Auto Scaling group to automatically scale your application up to handle massive traffic loads. It also provides you with a unique domain name for your application environment. The URL syntax is `<your-application>.elasticbeanstalk.com`. You can also resolve your own domain name to the provided domain name by using Amazon Route 53.



You were introduced to at least four AWS services in this module. One frequently asked question is about the multiple services that provide application management capabilities: where is the line between them, or which service should be used under which circumstances? Your decision should depend on the relative level of convenience and control that you need.

Elastic Beanstalk is an easy-to-use application service for building web applications that run on Java, PHP, Node.js, Python, Ruby, or Docker. If you want to upload your code and don't need to customize your environment, Elastic Beanstalk might be a good choice for you.

OpsWorks enables you to launch an application, define its architecture, and define the specification for each component, including package installation, software configuration, and resources (such as storage). You can use templates for common technologies (applications servers, databases, and others), or you can build your own template.

Both Elastic Beanstalk and OpsWorks provide a higher level of service than authoring and maintaining AWS CloudFormation templates to create stacks, or managing EC2 instances directly. However, the correct choice of service (or combinations of services) to use depends on your needs. These tools are all available to you. As an architect, you must decide which services will be the most appropriate for your use case.

Section 5 key takeaways

- Elastic Beanstalk creates and manages a scalable and highly available web application environment that enables you to focus on the application code
- You can author your Elastic Beanstalk application code in Java, .NET, PHP, Node.js, Python, Ruby, Go, or Docker
- AWS resources that are created by Elastic Beanstalk are fully transparent—they are visible in the AWS Management Console service page views
- No extra charge for Elastic Beanstalk – you pay only for the underlying resources that are used

Some key takeaways from this section of the module include:

- AWS Elastic Beanstalk creates and manages a scalable and highly available web application environment that enables you to focus on the application code
- You can author your Elastic Beanstalk application code in Java, .NET, PHP, Node.js, Python, Ruby, Go, or Docker
- AWS resources that are created by Elastic Beanstalk are fully transparent—they are visible in the AWS Management Console service page views
- No extra charge for Elastic Beanstalk – you pay only for the underlying resources that are used

Module 10 – Challenge Lab: Automating Infrastructure Deployment

You will now complete the Module 10 – Challenge Lab: Automating Infrastructure Deployment.

The business need: Implement IaC

- The café now has locations in multiple countries and must start automating to keep growing.
- They need a way to consistently deploy, manage, and update café resources across multiple AWS services.
- They want to be able to reliably create repeatable environments across AWS Regions, to serve both development and production needs.

For too long, the café has been creating their AWS resources and configured their applications manually. That approach worked well as a way for the café to quickly develop a web presence and build out an infrastructure that supports the needs of employees and customers. However, they find it challenging to replicate their deployments to new AWS Regions so that they can support new cafe locations in multiple countries.

The café would also like to have separate development and production environments that reliably have matching configurations. They realize that they must start automating to support continued growth.

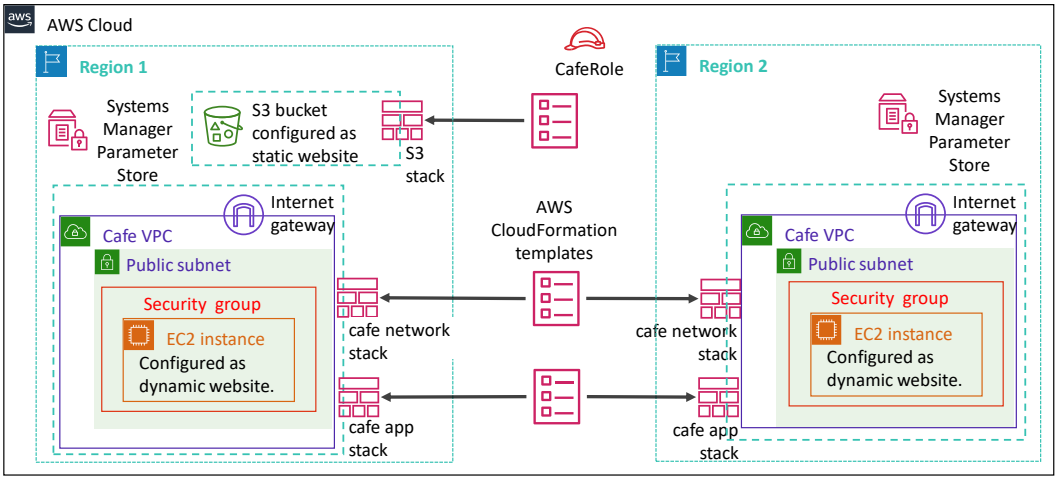
Challenge lab: Tasks

1. Creating an AWS CloudFormation template from scratch
2. Configuring the bucket as a website and updating the stack
3. Cloning a CodeCommit repository that contains AWS CloudFormation templates
4. Creating a new network layer with AWS CloudFormation, CodeCommit, and CodePipeline
5. Updating the network stack
6. Defining an EC2 instance resource and creating the application stack
7. Duplicating the café network and website to another AWS Region

In this challenge lab, you will complete the following tasks:

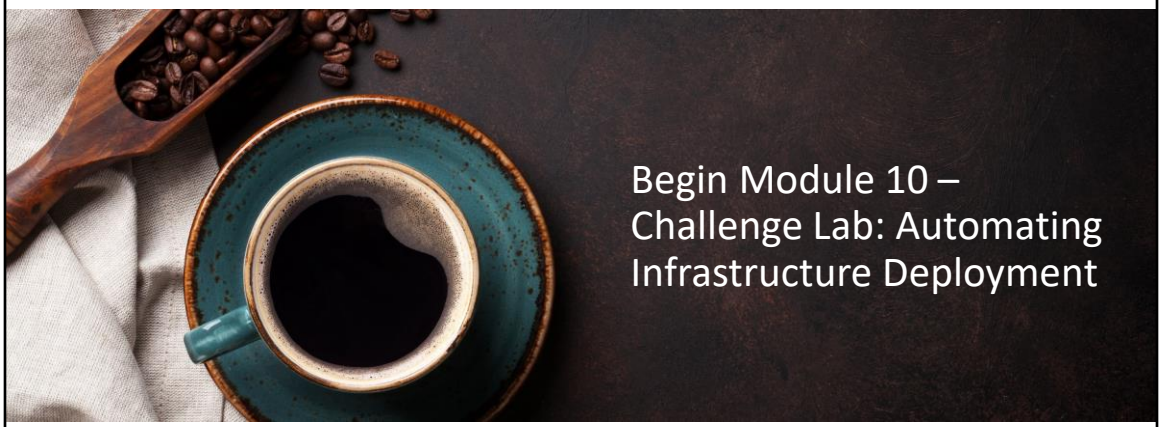
1. Creating an AWS CloudFormation template from scratch
2. Configuring the bucket as a website and updating the stack
3. Cloning a CodeCommit repository that contains AWS CloudFormation templates
4. Creating a new network layer with AWS CloudFormation, CodeCommit, and CodePipeline
5. Updating the network stack
6. Defining an EC2 instance resource and creating the application stack
7. Duplicating the café network and website to another AWS Region

Challenge lab: Final product



This diagram shows the completed architecture that you will build in the challenge lab.

~ 90 minutes



It is now time to start the challenge lab.

Challenge lab debrief: Key takeaways

Your educator might choose to lead a conversation about the key takeaways from this guided lab after you have completed it.

Module 10: Automating Your Architecture

Module wrap-up

aws academy

It’s now time to review the module and wrap up with a knowledge check and discussion of a practice certification exam question.

Module summary

aws academy

In summary, in this module, you learned how to:

- Recognize when to automate and why
- Identify how to model, create, and manage a collection of AWS resources using AWS CloudFormation
- Use the Quick Start AWS CloudFormation templates to set up an architecture
- Indicate how to use AWS System Manager and AWS OpsWorks for infrastructure and deployment automation
- Indicate how to use AWS Elastic Beanstalk to deploy simple applications

In summary, in this module, you learned how to:

- Recognize when to automate and why
- Identify how to model, create, and manage a collection of AWS resources using AWS CloudFormation
- Use the Quick Start AWS CloudFormation templates to set up an architecture
- Indicate how to use AWS System Manager and AWS OpsWorks for infrastructure and deployment automation
- Indicate how to use AWS Elastic Beanstalk to deploy simple applications

Complete the knowledge check

It is now time to complete the knowledge check for this module.

Sample exam question

Consider a situation where you want to create a single AWS CloudFormation template that is capable of creating both a production environment that spans two Availability Zones, and a development environment that exists in a single Availability Zone.

Which optional section of the AWS CloudFormation template will you want to make use of to configure the logic that will support this?

A. Resources
B. Outputs
C. Conditions
D. Description

Look at the answer choices and rule them out based on the keywords that were previously highlighted.

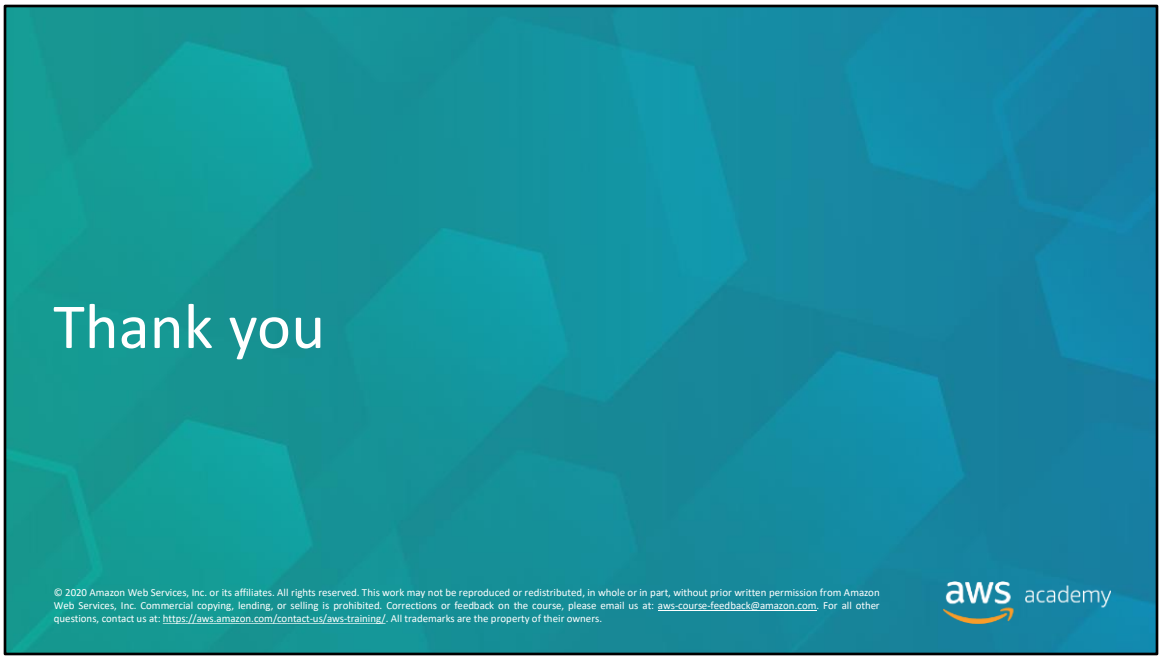
The correct answer is C – Conditions. The optional **Conditions** section contains statements that define the circumstances under which entities are created or configured. **Resources** is not an optional section of an AWS CloudFormation template. The **Outputs** section cannot affect how many AWS resources will be deployed by the template when the stack is run. The **Description** section does not affect the configuration.

Additional resources

- Overview of Deployment Options on AWS
- Working with AWS CloudFormation Templates
- AWS CloudFormation Sample Templates
- AWS OpsWorks Stacks FAQs
- AWS Systems Manager Features
- AWS Elastic Beanstalk FAQs

If you want to learn more about the topics covered in this module, you might find the following additional resources helpful:

- Overview of Deployment Options on AWS
- Working with AWS CloudFormation Templates
- AWS CloudFormation Sample Templates
- AWS OpsWorks Stacks FAQs
- AWS Systems Manager Features
- AWS Elastic Beanstalk FAQs



Thank you for completing this module.