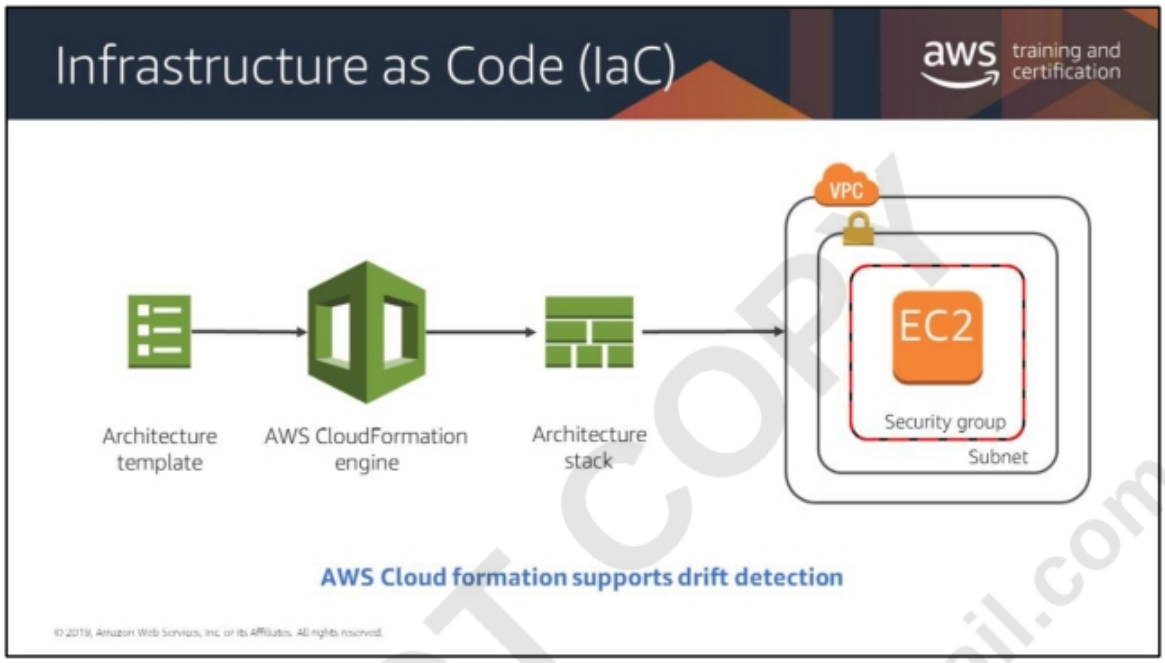


AWS CloudFormation makes deploying a set of Amazon Web Services (AWS) resources as simple as submitting a template to the service.

A *template* is a text file that describes and defines the resources to be deployed in an environment.

AWS CloudFormation is the engine that processes the template. The output of AWS CloudFormation is called a *stack*.
A *stack* is a collection of AWS resources deployed together as a group.

Additional points for an AWS CloudFormation template:
- Treat it as code and manage it using a version control system.
- Define an entire application stack (all resources required for your application) in a JSON or YAML template file.
- Define runtime parameters for a template (Amazon EC2 instance size, Amazon EC2 key pair, etc.).

For a list of resources supported by AWS CloudFormation, see https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html.

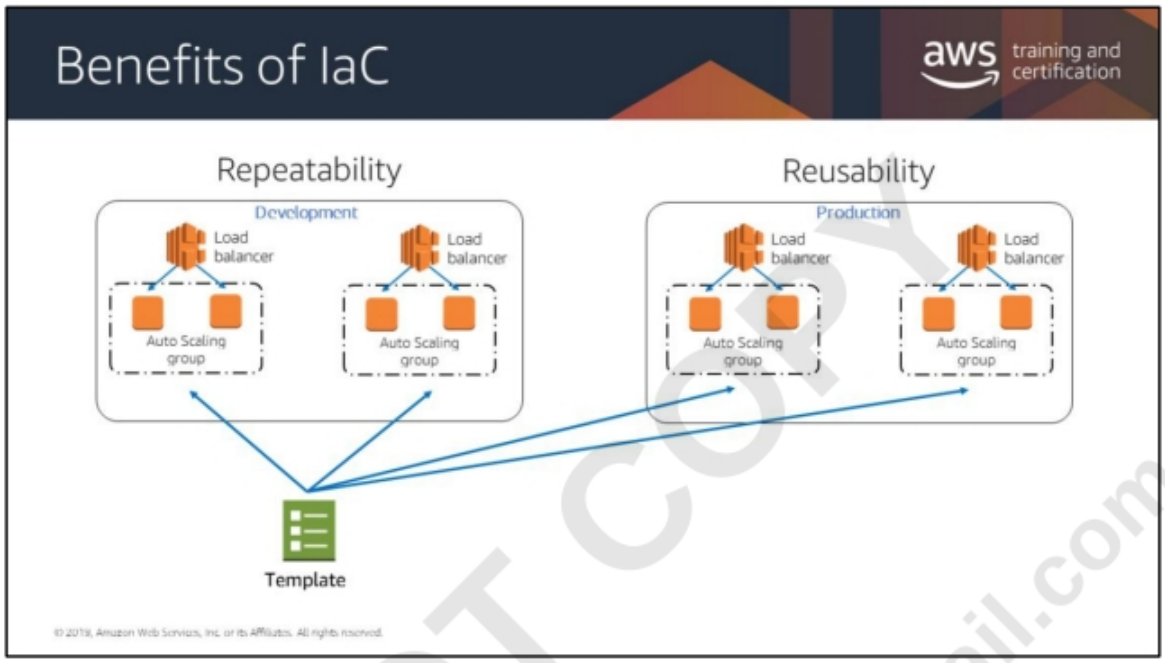
/

For sample templates, see https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-sample-templates.html.

Performing a drift detection operation on a stack determines whether the stack has drifted from its expected template configuration, and returns detailed information about the drift status of each resource in the stack that *supports* drift detection. Drift detection can be enabled on a stack by clicking on "Detect Drift for current stack" dialog box. The process will continue the drift detection even if you close the dialog box and review the drift details later. For more information, see these sites:

- https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/detect-drift-stack.html
- https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-stack-drift.html

/

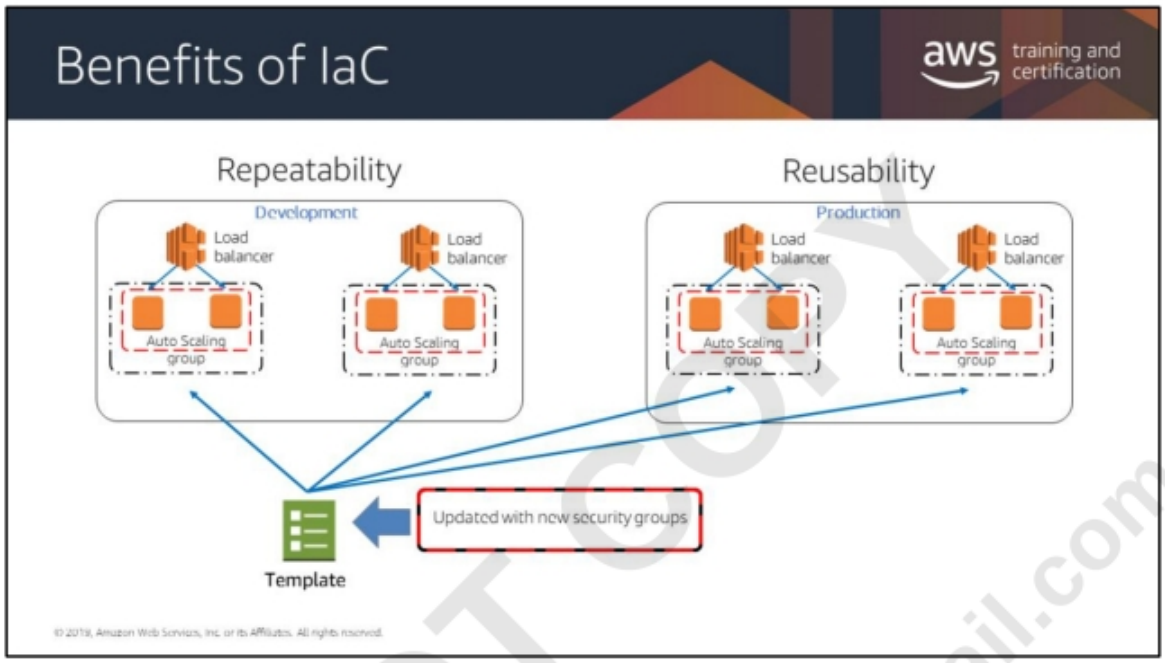


If you build infrastructure with code, you gain the benefits of repeatability and reusability while building your environments.

With one template (or a combination of templates), you can build the same complex environments over and over again.

When doing this with AWS, you can even create environments dependent upon conditions, so that what ends up being built is specific to the context in which you've created it. For instance, a template can be designed so that different AMIs are used based on whether or not this template was launched into the development or the production environments.

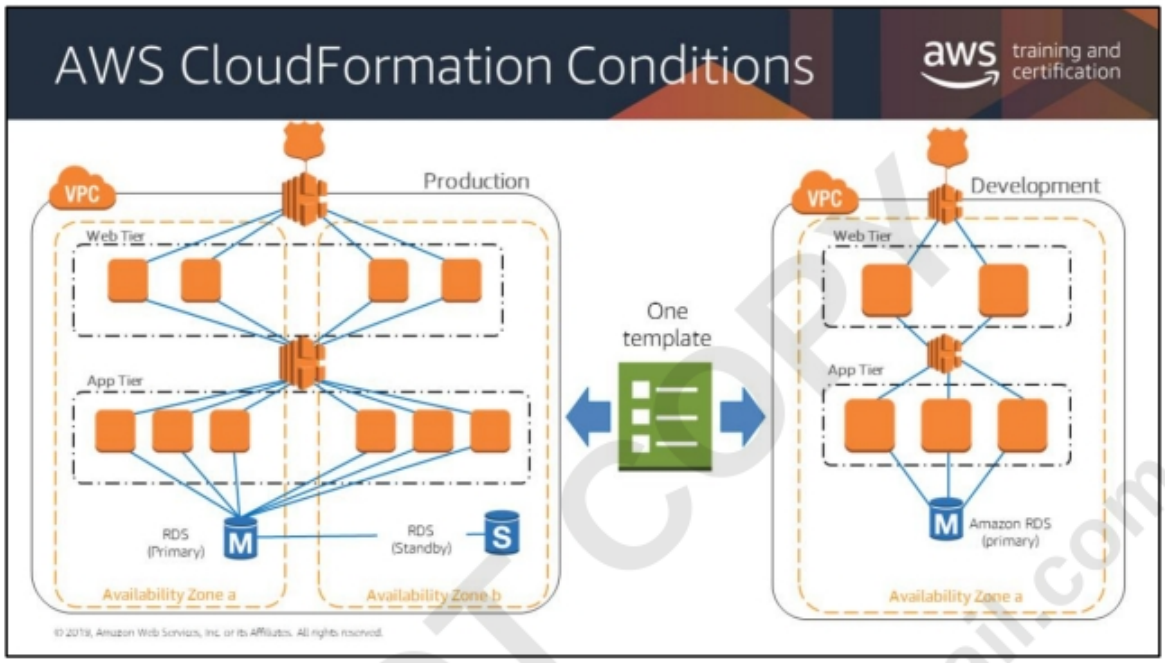
/



In this scenario, the template has been updated to add new security groups to the instance stacks.

With one change to the template used to launch these environments, all four environments can have the new security group resource added.

This feature provides the benefit of easier maintainability of resources, as well as greater consistency and a reduction in effort through parallelization.

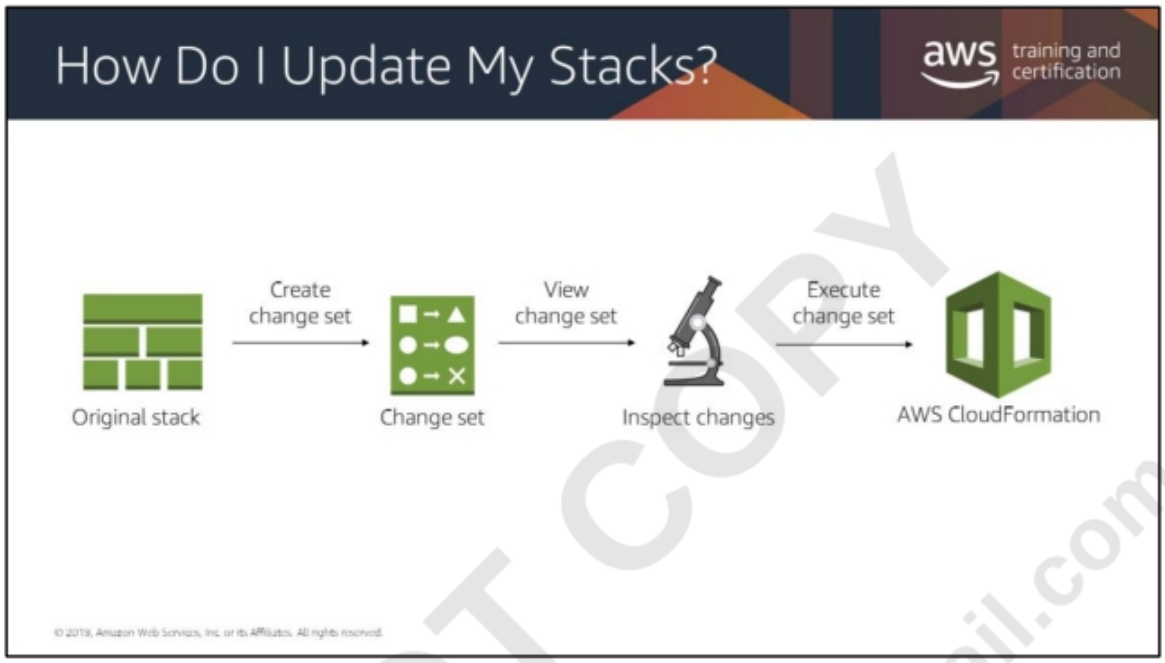


Your production environment and development environment must be built from the same stack. This ensures that your application works in production the way it was designed and developed.

Additionally, your development environment and testing environment must use the same stack. All environments will have identical applications and configurations.

You might need several testing environments for functional testing, user acceptance testing, and load testing. Creating those environments manually comes with great risk.

You can use a *Conditions* statement in AWS CloudFormation templates to ensure that, while different in size and scope, development, test, and production are configured identically.



One way to update your stacks is to edit your existing templates and run them again.

However, if you need additional insight about the changes AWS CloudFormation is planning to perform when it updates a stack, you can use a *change set*.

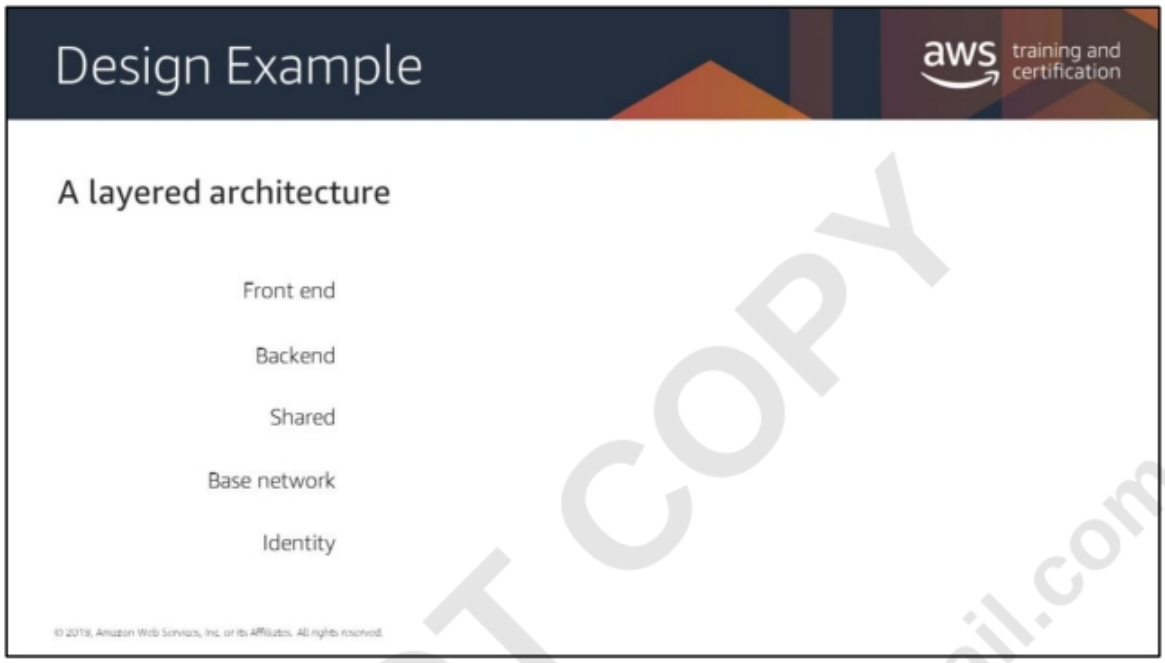
Change sets will let you preview the changes, allow you to verify they are in line with expectations, and then approve the update before proceeding.

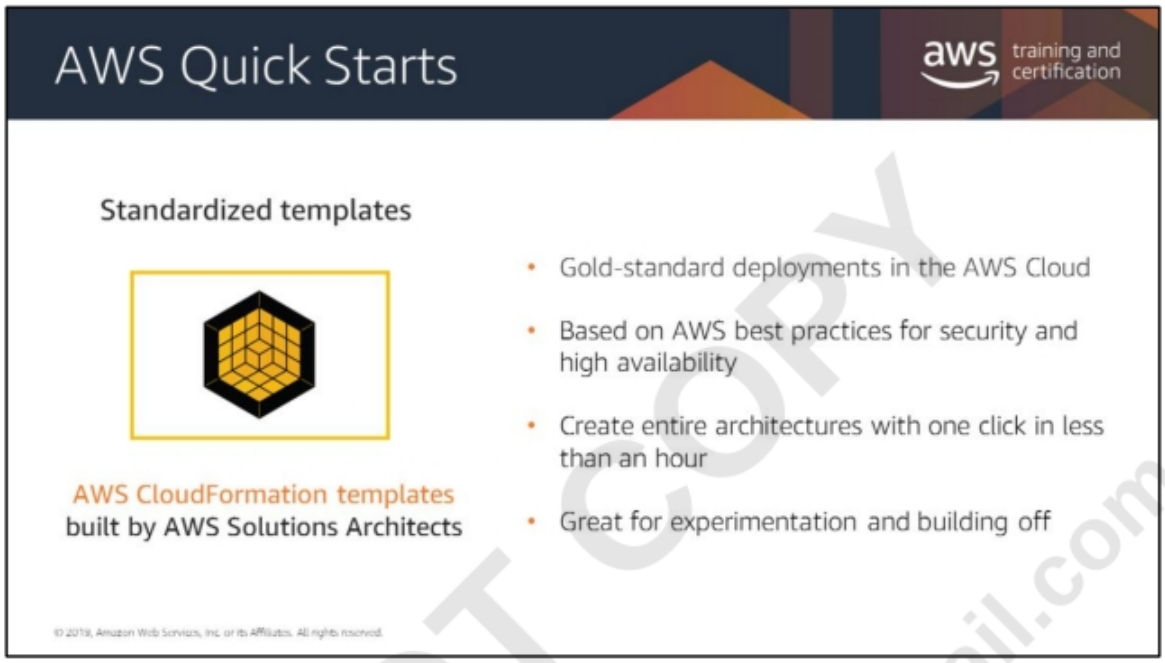
This is the basic workflow for using change sets:

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.
3. If you want to consider other changes before you decide which changes to make, create additional change sets.
4. Execute the change set. AWS CloudFormation updates your stack with those changes.

For more information, see:
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-changesets.html.

/





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 implement key technologies automatically on the AWS Cloud, often with a single click and in less than an hour. You can build your test or production environment in a few steps, and start using it immediately.

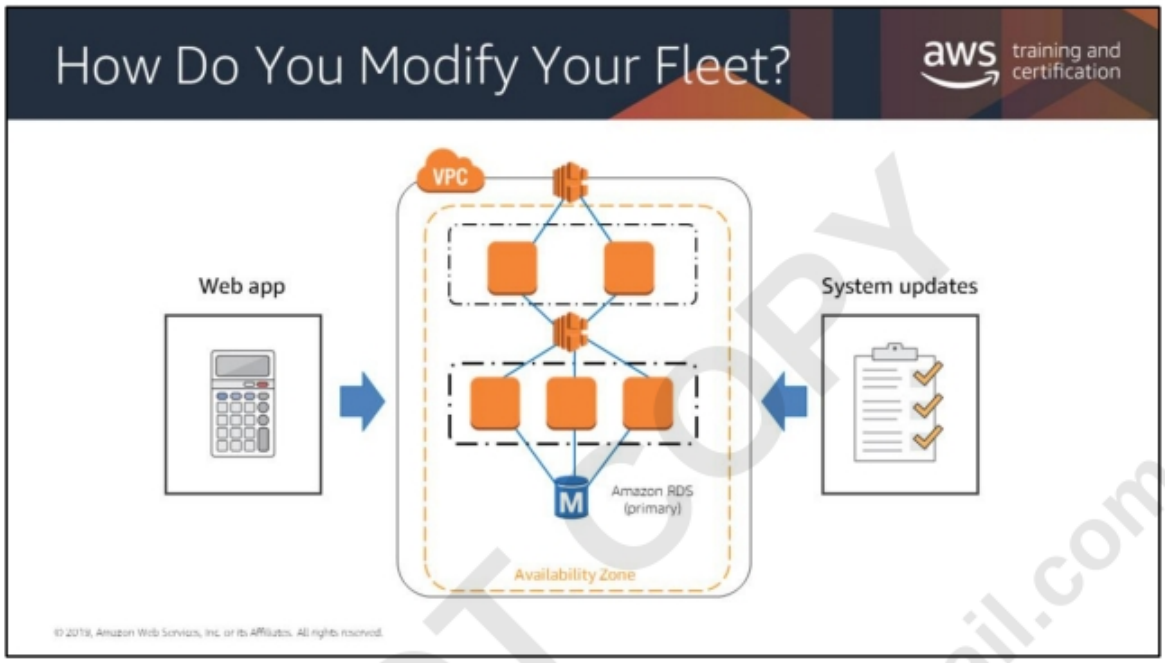
/

The Quick Start is made of an AWS CloudFormation template and associated scripts to create the environment in your AWS account. It deals with all the bootstrapping and deploying on your behalf. There is also a deployment guide that will tell you how everything was created.

You will be charged for the resources used in the creation of, and running of this environment.

For more information, see https://aws.amazon.com/quickstart/

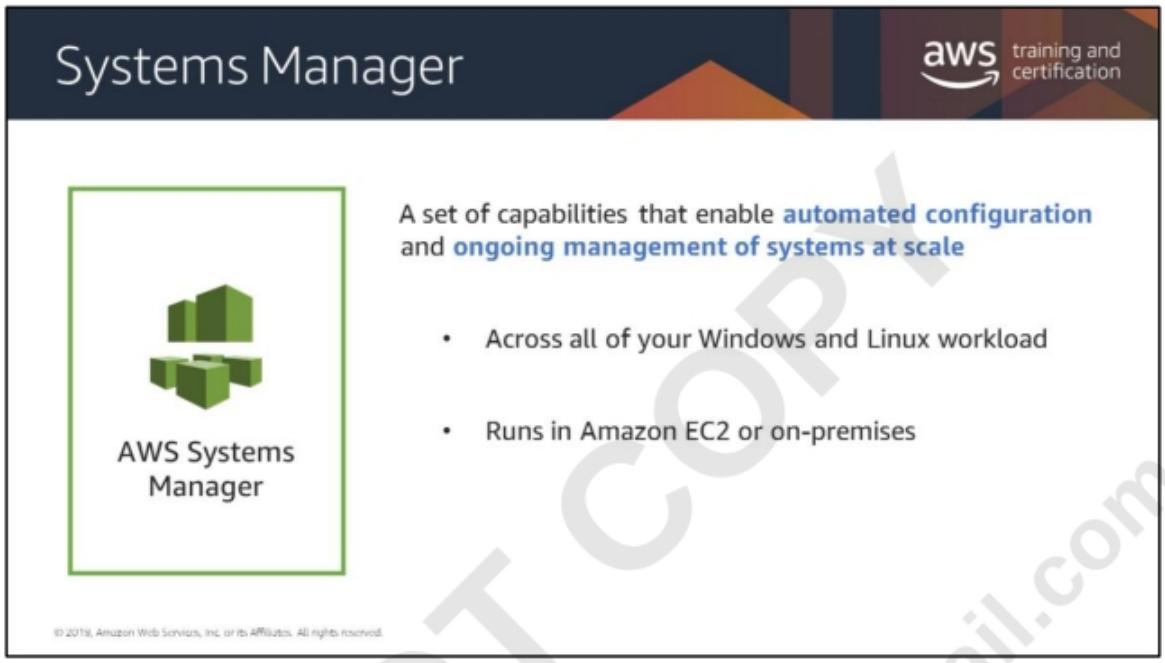
/



You can create an entire infrastructure automatically by using AWS CloudFormation, but there are still some important questions to ask.

How do update your Amazon EC2 instances? Are you supposed to log into each box and run update commands yourself? Download the newest version of your web app? How do you revert a change if something goes wrong? What if you have 100 servers that run three different apps?

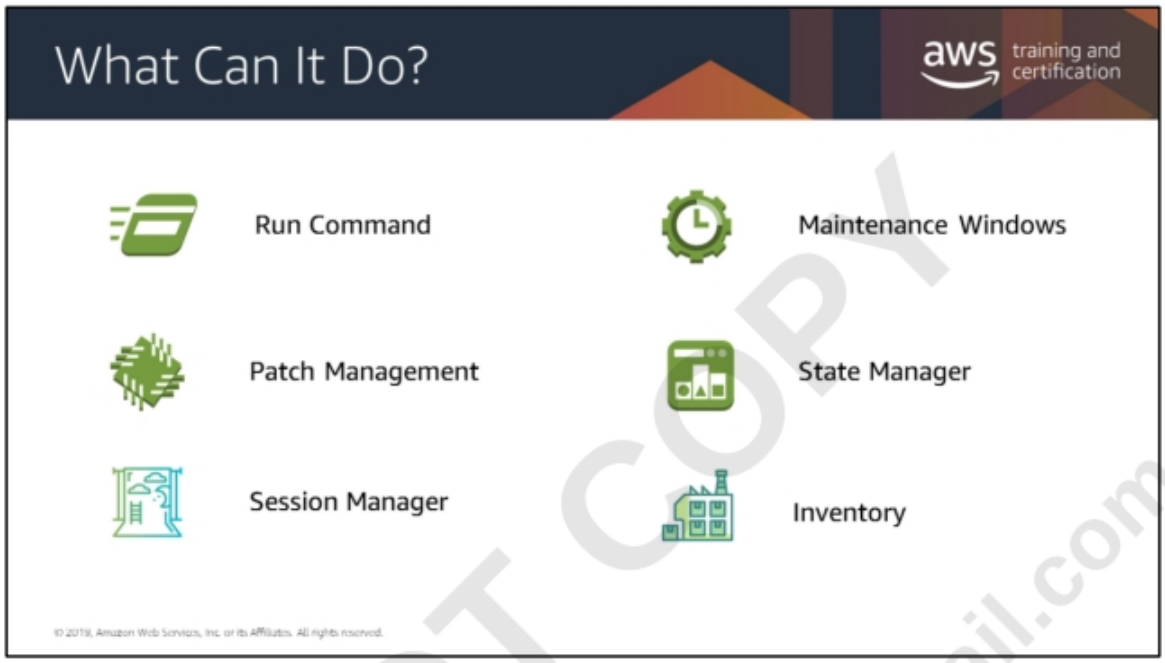
There are traditional tools that can help with these scenarios, but an out-of-the-box solution would be more convenient.



AWS Systems Manager is a management service that helps you automatically collect software inventory, apply OS patches, create system images, and configure Windows and Linux operating systems. These capabilities help you define and track system configurations, prevent drift, and maintain software compliance of your Amazon EC2 and on-premises configurations. By providing a management approach that is designed for the scale and agility of the cloud but extends into your on-premises data center, AWS Systems Manager makes it easier for you to seamlessly bridge your existing infrastructure with AWS.

You can open AWS Systems Manager from the Amazon EC2 console. Select the instances you want to manage, and define the management tasks you want to perform. AWS Systems Manager is available at no cost to manage both your Amazon EC2 and on-premises resources.

/



Use Systems Manager Run Command to remotely and securely manage the configuration of your managed instances at scale. Use Run Command to perform on-demand changes like updating applications or running Linux shell scripts and Windows PowerShell commands on a target set of dozens or hundreds of instances.

For more information, see: https://docs.aws.amazon.com/systems-manager/latest/userguide/execute-remote-commands.html

Use Patch Manager to automate the process of patching your managed instances. This capability enables you to scan instances for missing patches and apply missing patches individually or to large groups of instances by using Amazon EC2 instance tags. For security patches, Patch Manager uses patch baselines that include rules for auto-approving patches within days of their release, as well as a list of approved and rejected patches. Security patches are installed from the default repository for patches configured for the instance. You can install security patches on a regular basis by scheduling patching to run as a Systems Manager Maintenance Window task. For Linux operating systems, you can define the repositories that should be used for patching operations as part of your patch baseline.

This allows you to ensure that updates are installed only from trusted repositories regardless of what repositories are configured on the instance. For Linux, you also have the ability to update any package on the instance, not just those that are classified as operating system security updates.

For more information, see: https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-patch.html

Use Maintenance Windows to set up recurring schedules for managed instances to execute administrative tasks like installing patches and updates without interrupting business-critical operations.

For more information, see: https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-maintenance.html

Use Systems Manager State Manager to automate the process of keeping your managed instances in a defined state. You can use State Manager to ensure that your instances are bootstrapped with specific software at startup, joined to a Windows domain (Windows instances only), or patched with specific software updates.

For more information, see: https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-state.html

Use Session Manager to manage your Amazon EC2 instances through an interactive one-click browser-based shell or through the AWS CLI. Session Manager provides secure and auditable instance management without the need to open inbound ports, maintain bastion hosts, or manage SSH keys. Session Manager also makes it easy to comply with corporate policies that require controlled access to instances, strict security practices, and fully auditable logs with instance access details, while still providing end users with simple one-click cross-platform access to your Amazon EC2 instances.

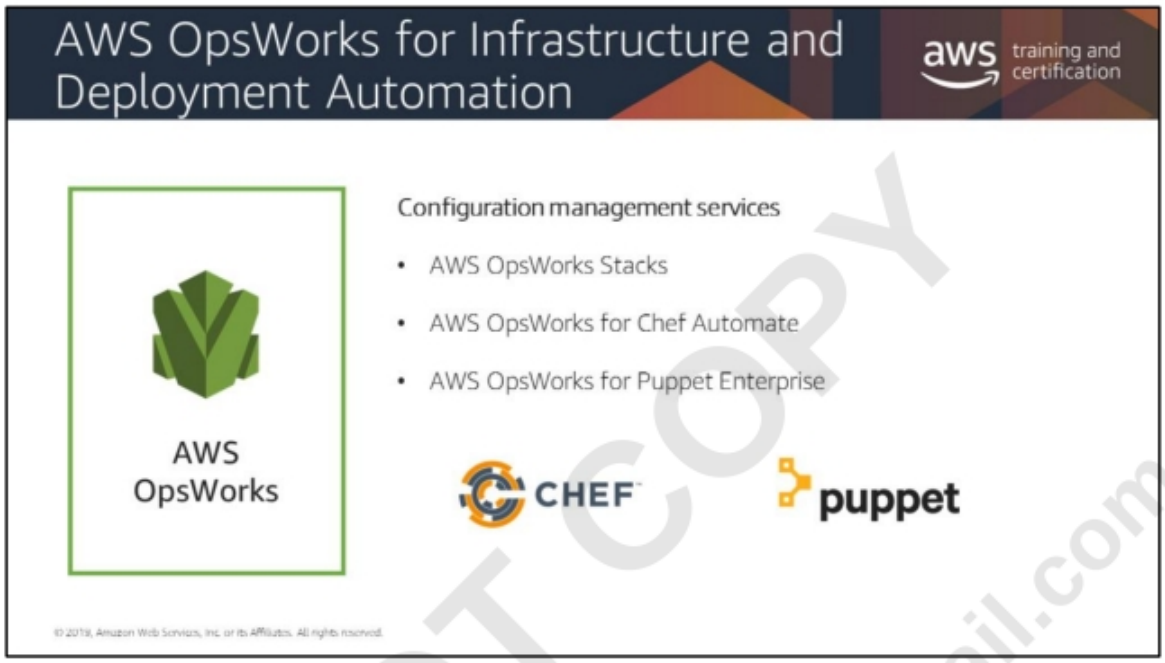
For more information, see: https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html

Use Inventory for visibility into your Amazon EC2 and on-premises computing environment. You can use Inventory to collect metadata from your managed instances.

For more information, see: https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-inventory.html

/

Session Manager icon made by smalllikeart from www.flaticon.com
Inventory Icon made by wanicon from www.flaticon.com



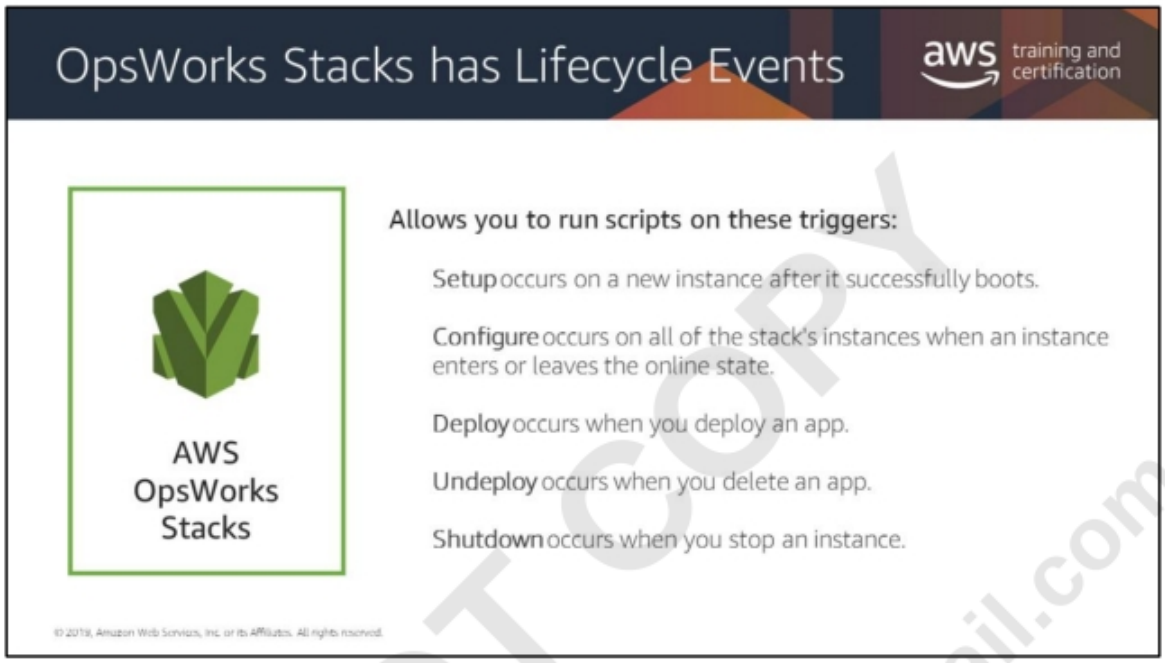
AWS OpsWorks Stacks is a configuration management service that helps you configure and operate applications of all shapes and sizes using Chef. You can define the application's architecture and the specification of each component, including package installation, software configuration and resources such as storage. Start from templates for common technologies such as application servers and databases or build your own to perform any task that can be scripted. AWS OpsWorks Stacks includes automation to scale your application based on time or load and dynamic configuration to orchestrate changes as your environment scales.

AWS OpsWorks for Chef Automate provides a fully managed Chef Automate server and suite of automation tools that give you workflow automation for continuous deployment, automated testing for compliance and security, and a user interface that gives you visibility into your nodes and their status. The Chef Automate platform gives you full stack automation by handling operational tasks such as software and operating system configurations, continuous compliance, package installations, database setups, and more. The Chef server centrally stores your configuration tasks and provides them to each node in your compute environment at any scale, from a few nodes to thousands of nodes. OpsWorks for Chef Automate is completely compatible with tooling and cookbooks from the Chef community and automatically registers new nodes with your Chef server.

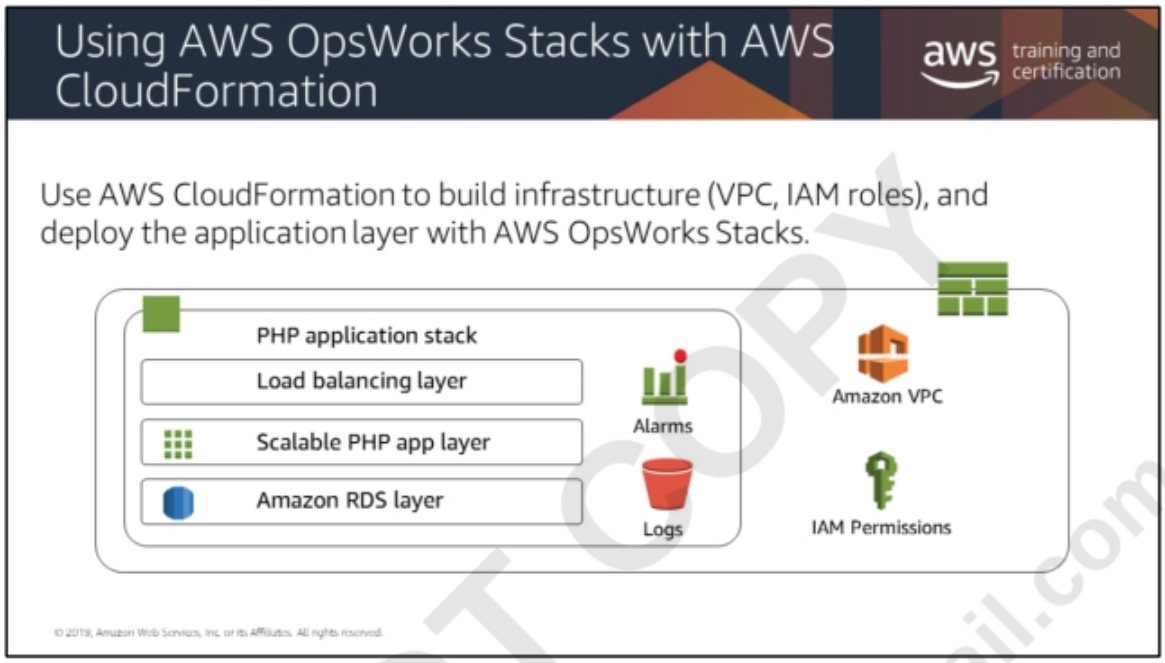
/

AWS OpsWorks for Puppet Enterprise provides a managed Puppet Enterprise server and suite of automation tools that give you workflow automation for orchestration, automated provisioning, and visualization for traceability. The Puppet Enterprise server gives you full stack automation by handling operational tasks such as software and operating system configurations, package installations, database setups, and more. The Puppet Master centrally stores your configuration tasks and provides them to each node in your compute environment at any scale, from a few nodes to thousands of nodes.

/

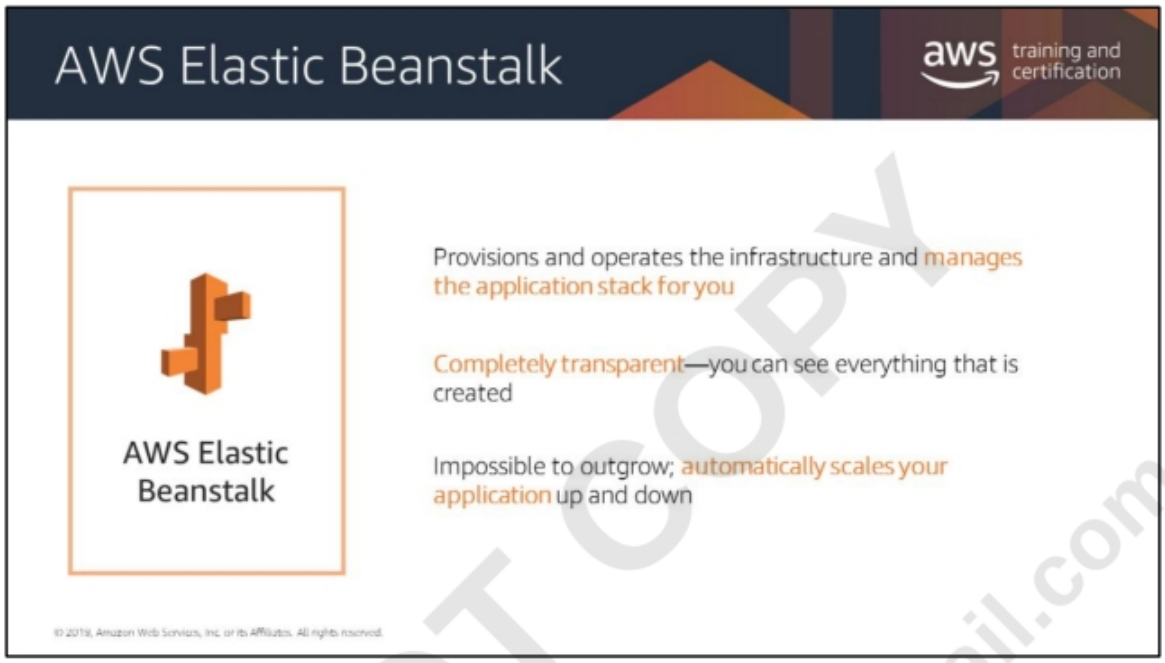


/

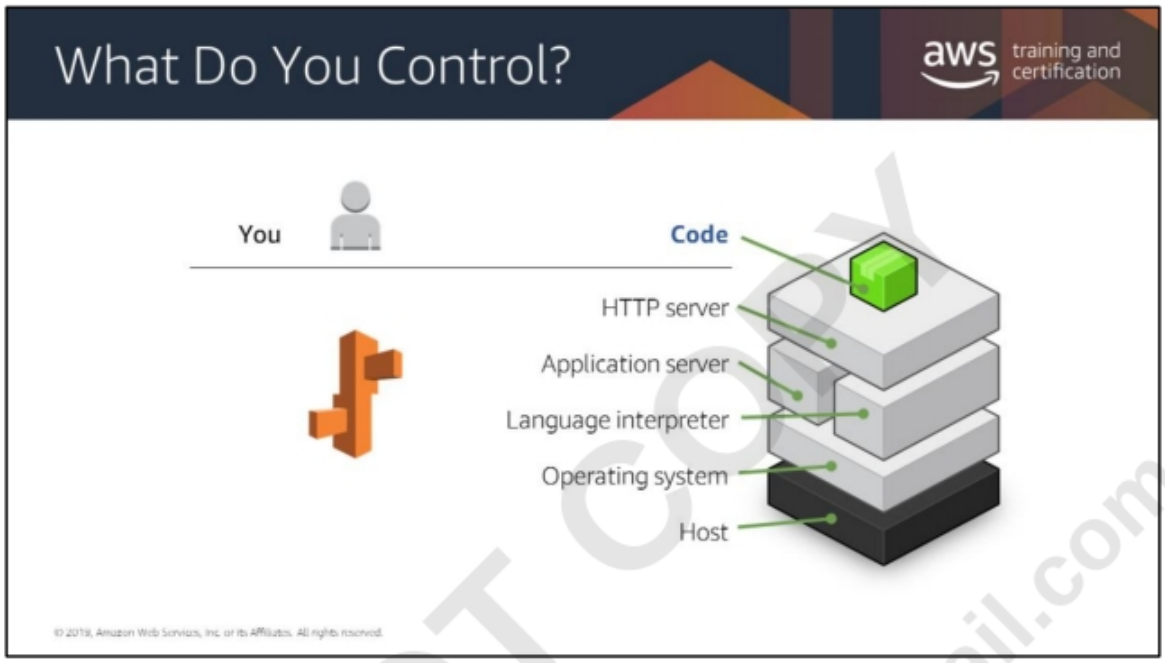


Because AWS OpsWorks Stacks can be created via AWS CloudFormation, the use of the two technologies is not mutually exclusive. You can use a set of tiered AWS CloudFormation templates in which one template creates the infrastructure for your environment (e.g., an Amazon VPC, IAM roles, an Amazon SQS queue for communication with external applications) and then uses a separate AWS CloudFormation template to create the AWS OpsWorks Stacks stack that will be deployed within that infrastructure.

/

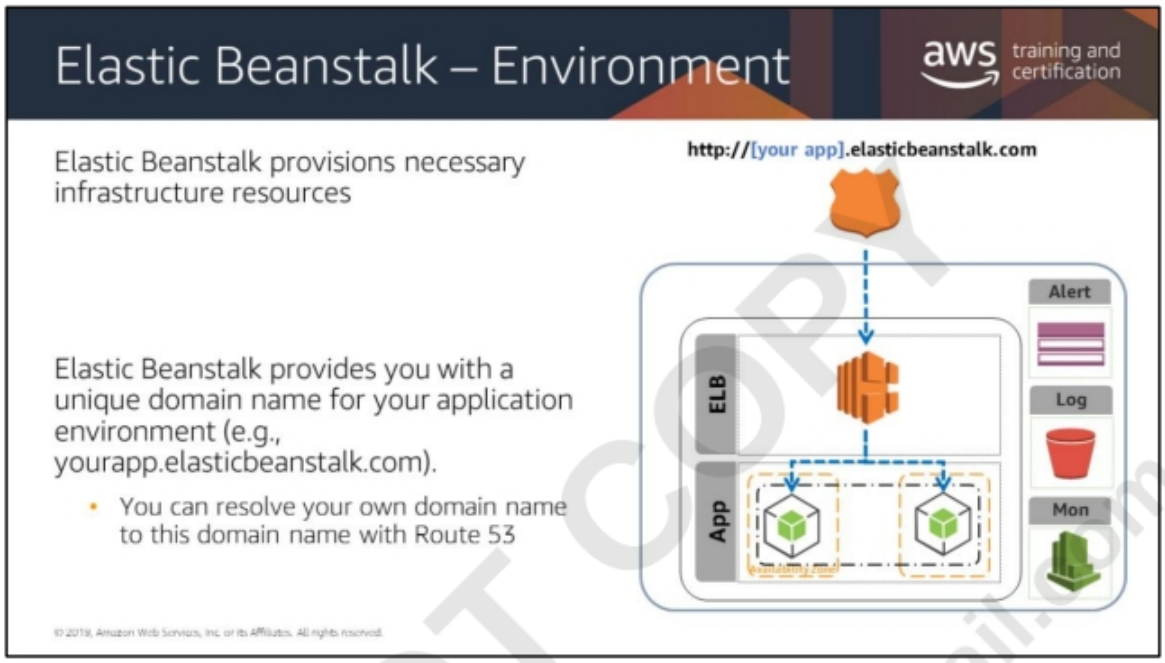


/



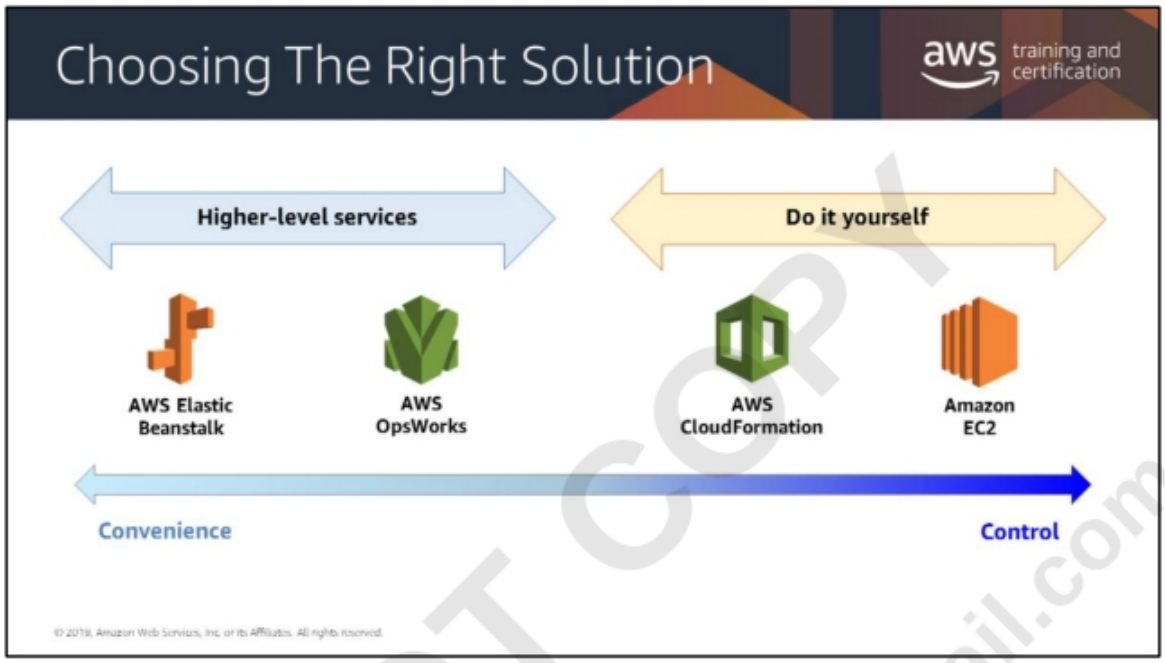
The goal of AWS Elastic Beanstalk is to help developers deploy and maintain scalable web applications and services in the cloud without having to worry about the underlying infrastructure. Elastic Beanstalk configures each EC2 instance in your environment with the components necessary to run applications for the selected platform. You don't have to worry about logging into instances to install and configure your application stack.

/



There are two types of environments you can choose from when working with AWS Elastic Beanstalk. The single-instance environment will allow you to launch a single EC2 instance and will not include load balancing or Auto Scaling. The other type of environment can launch multiple EC2 instances and includes load balancing and Auto Scaling configuration.

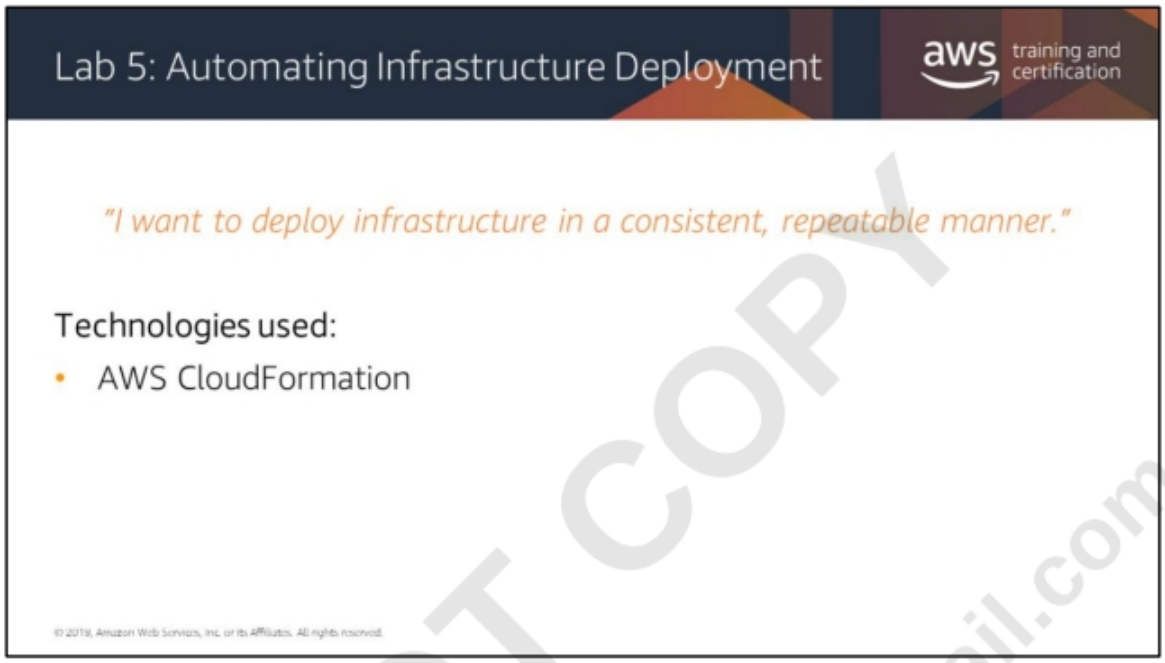
Elastic Beanstalk provisions necessary infrastructure resources, such as ELB, Auto Scaling groups, security groups, and databases (optional).

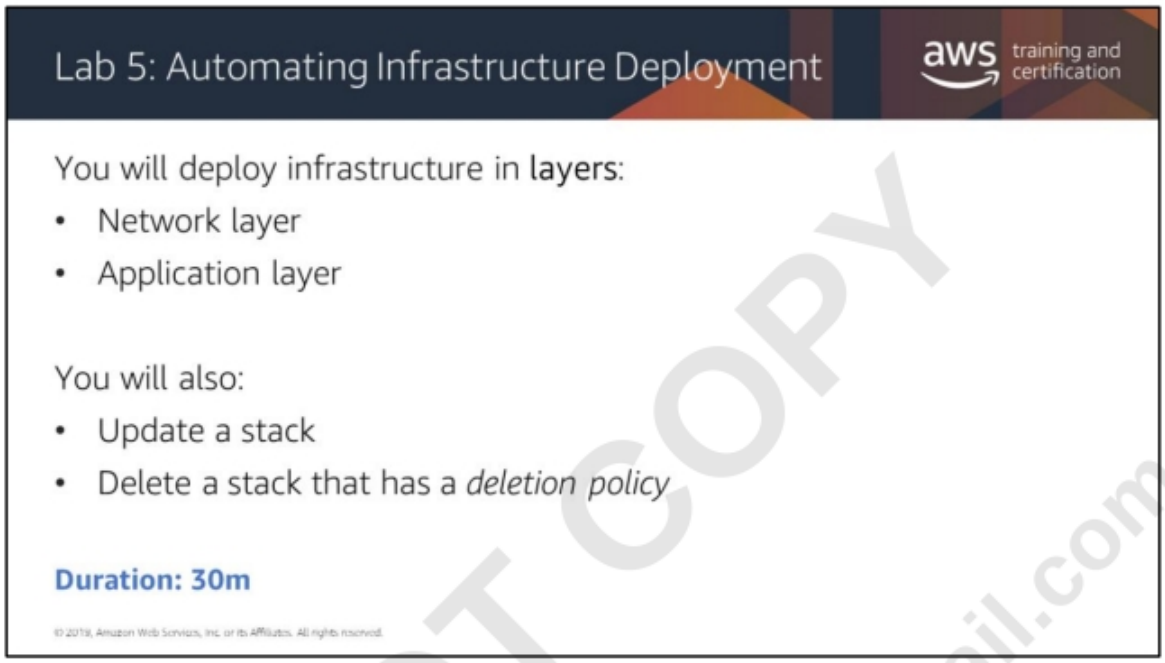


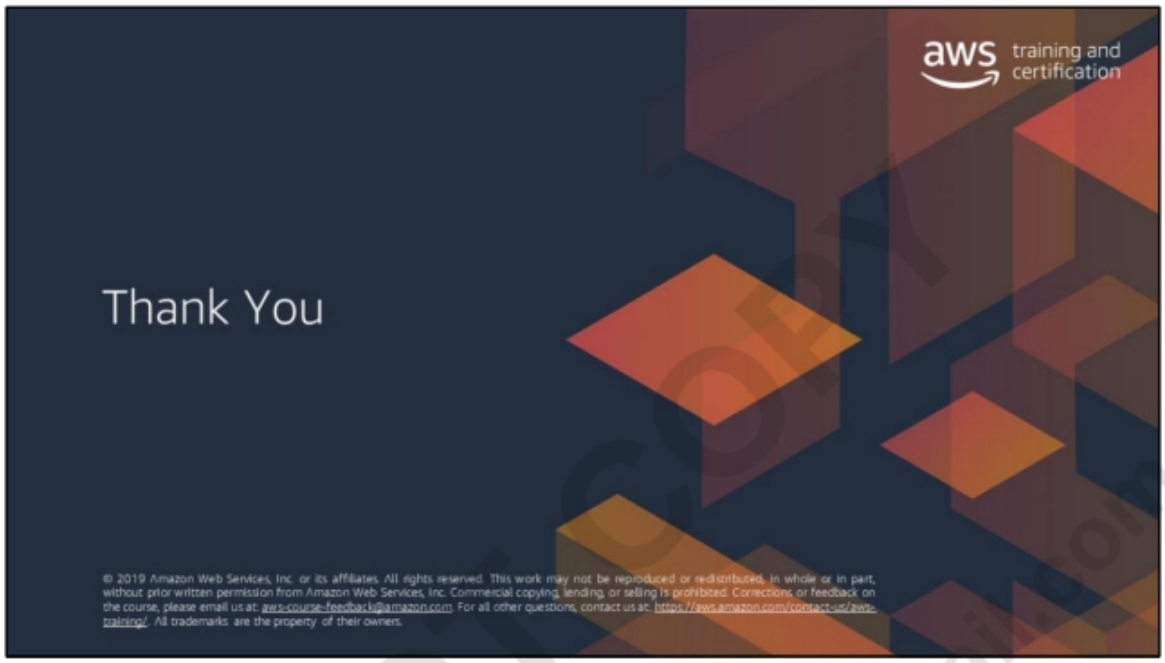
One frequently asked question is about the multiple services that provide application management capabilities, and where the line between them is. It really depends on the level of convenience and control that you need.

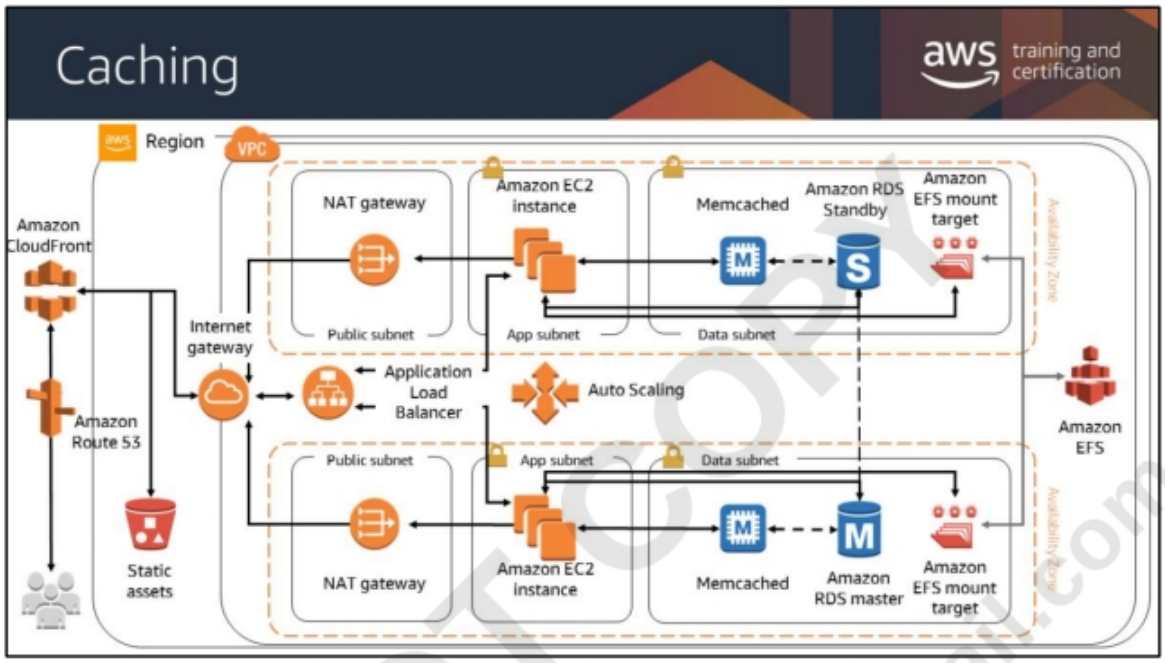
AWS Elastic Beanstalk is an easy-to-use application service for building web applications with a popular container: Java, PHP, Node.js, Python, Ruby, and Docker. If you want to upload your code and don't need to customize your environment, Elastic Beanstalk may be a good choice for you.

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



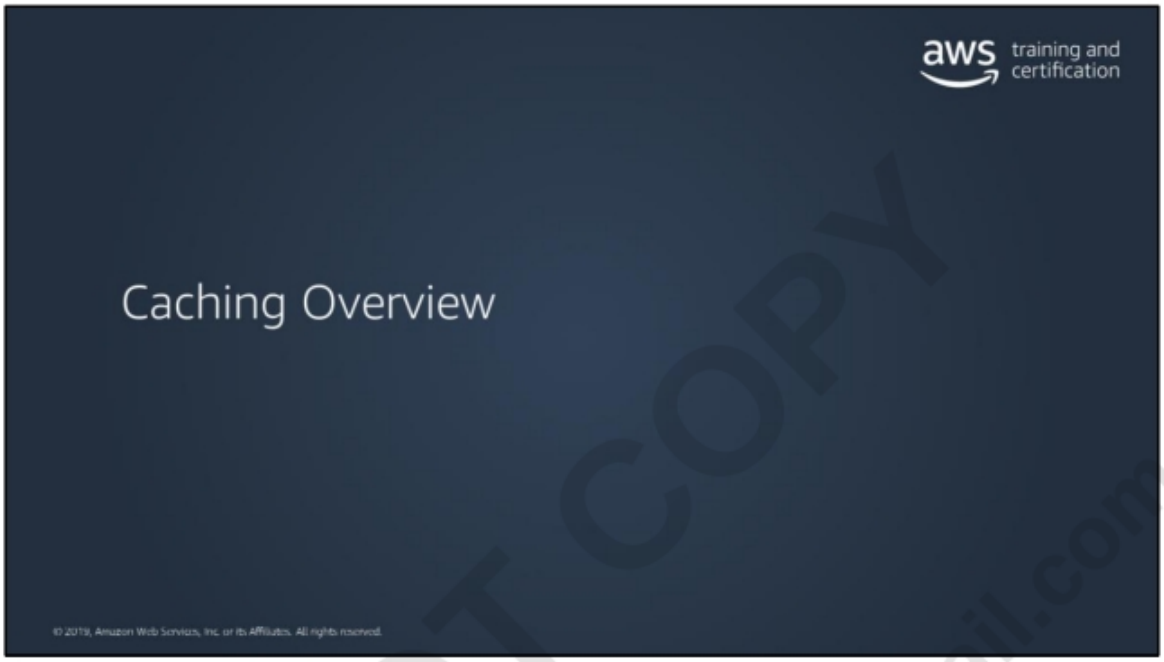




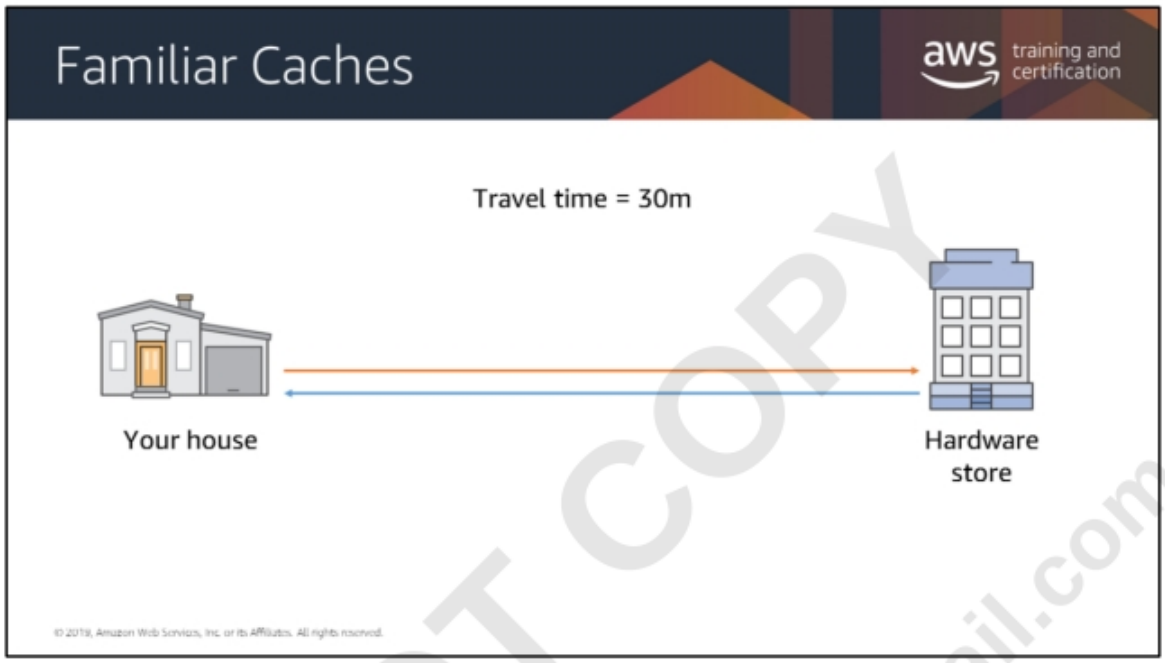


By the end of class, you will be able to understand all of the components of this architectural diagram. You will also be able to construct your own architectural solutions that are just as large and robust.

/

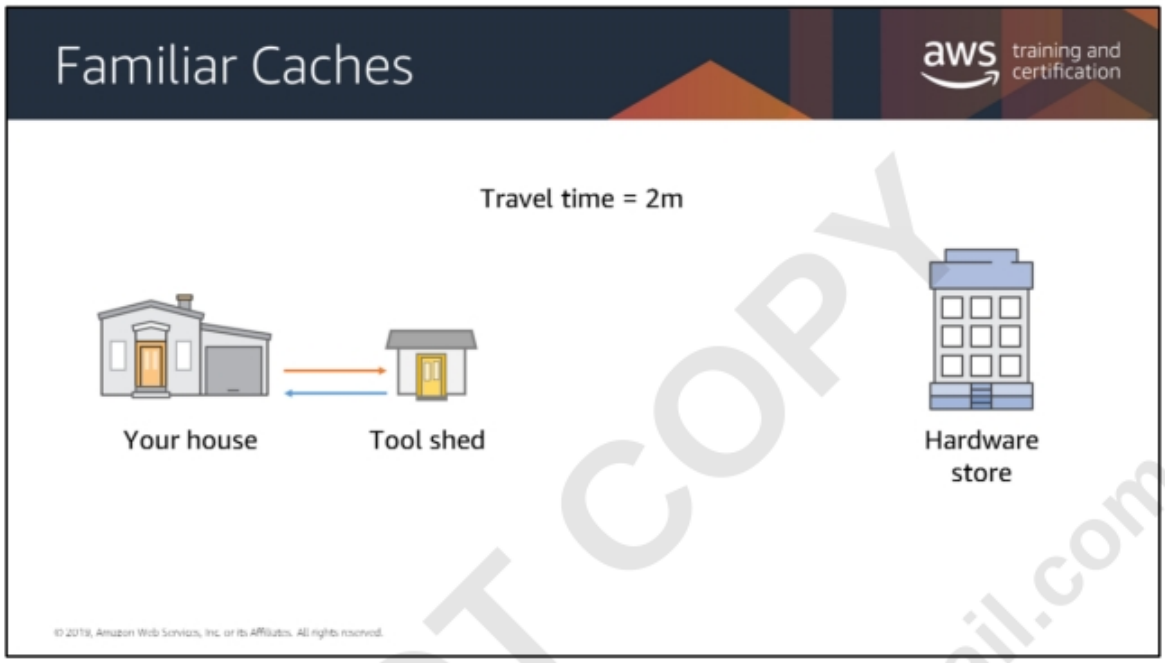


/



To illustrate how a cache improves performance, think about making a trip to a hardware store.

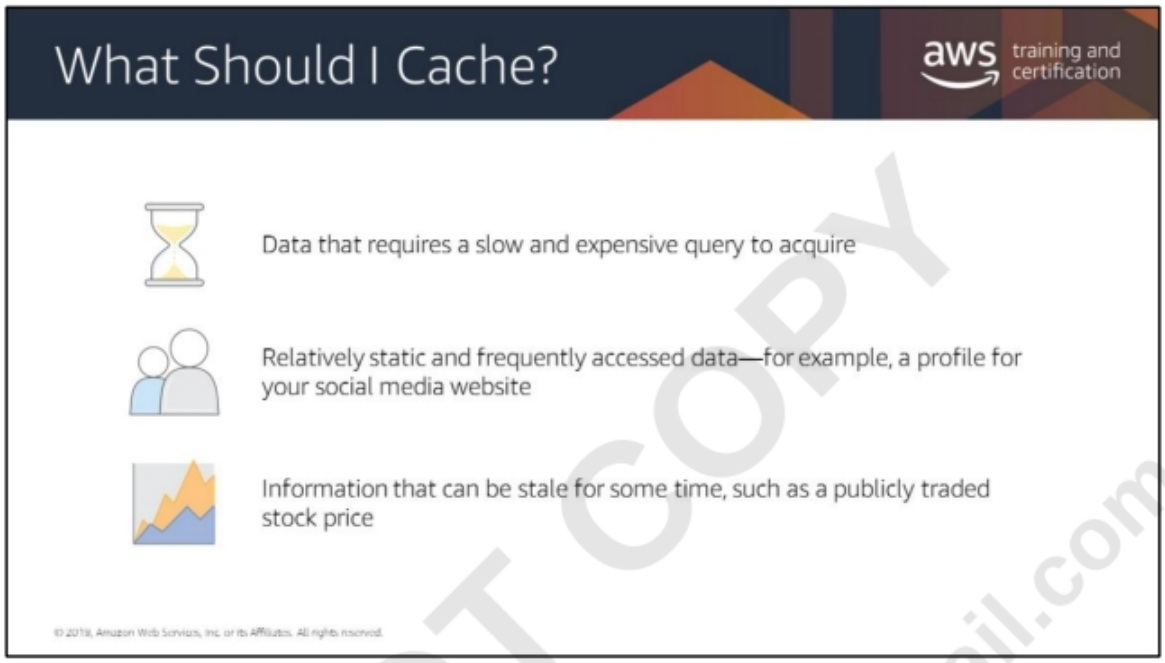
If the store is miles away, it's going to take considerable effort to go there every time you need something.



Instead, if you have a storage unit, a cache, close by, fill it with the supplies you need. Then, instead of going to the store when you need something, you can just go to your storage.

However, the store is always an option if you need to refresh your stockpile.

/



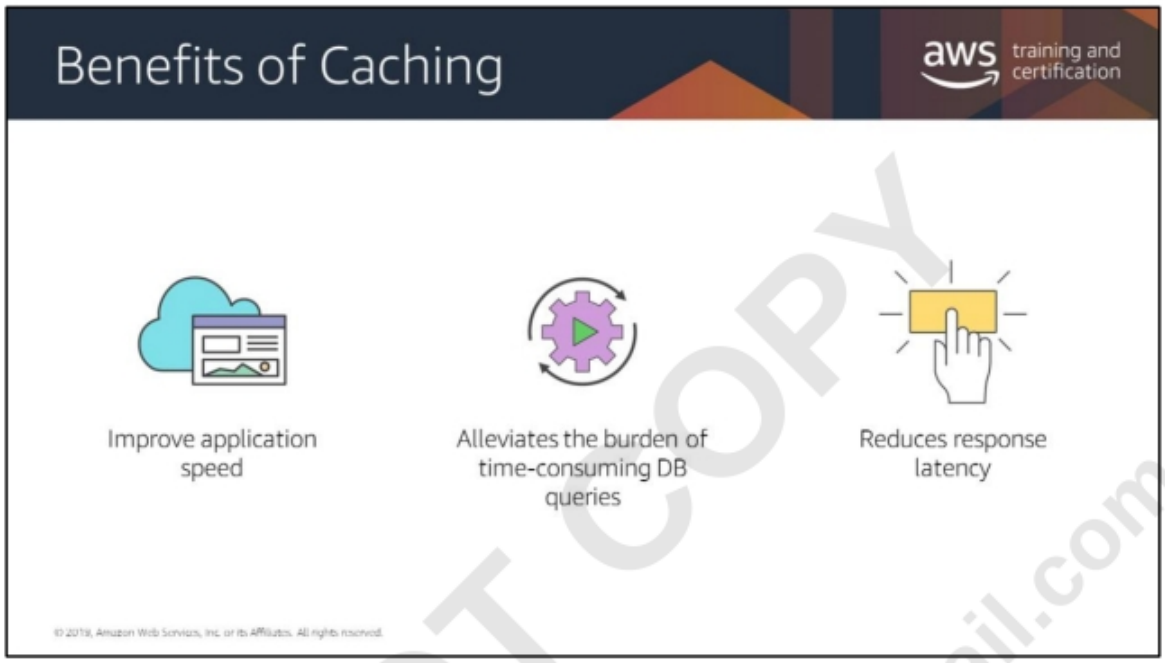
Speed and Expense – It's always slower and more expensive to acquire data from a database than from a cache. Some database queries are inherently slower and more expensive than others. For example, queries that perform joins on multiple tables are significantly slower and more expensive than simple, single-table queries. If acquiring the interesting data requires a slow and expensive query, it's a candidate for caching. If acquiring the data requires a relatively quick and simple query, it might still be a candidate for caching, depending on other factors.

Data and Access Pattern – Determining what to cache also involves understanding the data itself and its access patterns. For example, it doesn't make sense to cache data that is rapidly changing or is seldom accessed. For caching to provide a meaningful benefit, the data should be relatively static and frequently accessed, such as a personal profile on a social media site. Conversely, you don't want to cache data if caching it provides no speed or cost advantage. For example, it doesn't make sense to cache webpages that return the results of a search because such queries and results are almost always unique.

Staleness – By definition, cached data is stale data—even if in certain circumstances it isn't stale, it should always be considered and treated as stale. In determining whether your data is a candidate for caching, you need to determine your application's tolerance for stale data. Your application might be able to tolerate stale data in one context, but not another.

For example, when serving a publicly traded stock price on a website, staleness might be acceptable, with a disclaimer that prices might be up to *n* minutes delayed. But when serving up the price for the same stock to a broker making a sale or purchase, you want real-time data.

/



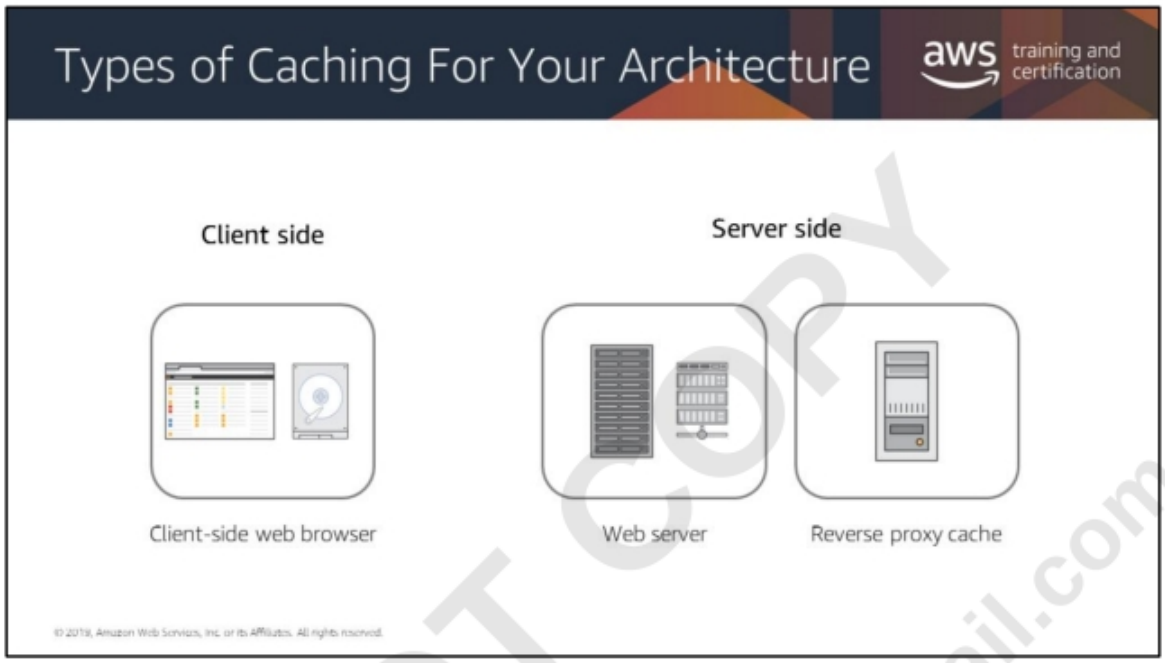
A cache provides high throughput, low-latency access to commonly accessed application data, by storing the data in memory. Caching can improve the speed of your application. Caching reduces the response latency experienced by the users of your application. Time-consuming database queries and complex queries often create bottlenecks in applications. In read-intensive applications, caching can provide significant performance gains by reducing application processing time and database access time.

Write-intensive applications typically do not see as great a benefit from caching. However, even write-intensive applications normally have a read/write ratio greater than 1, which implies that read caching will still be beneficial

You should consider caching your data if your data:

- Is slow or expensive to acquire when compared to cache retrieval
- Is accessed with sufficient frequency
- Is relatively static or if rapidly changing and staleness is not a significant issue

/



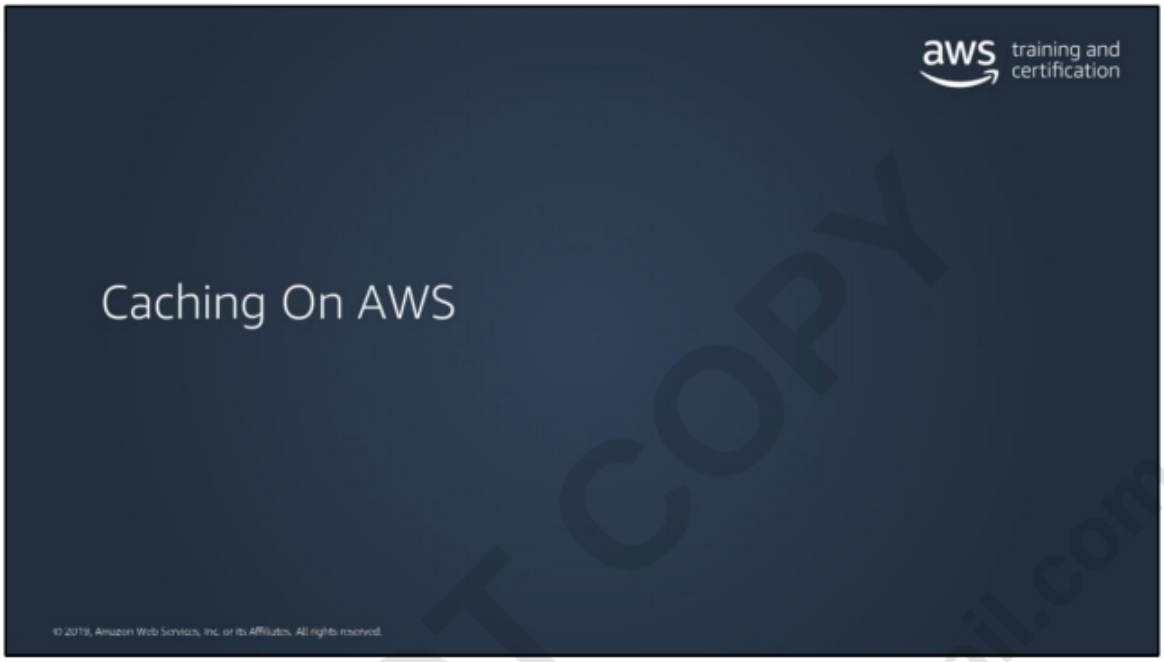
In computing, a *cache* is a high-speed data storage layer that stores a subset of data, typically transient in nature, so that future requests for that data are served up faster than is possible by accessing the data's primary storage location. Caching allows you to efficiently reuse previously retrieved or computed data.

Web caching is performed by retaining HTTP responses and web resources in the cache for the purpose of fulfilling future requests from cache rather than from the origin servers.

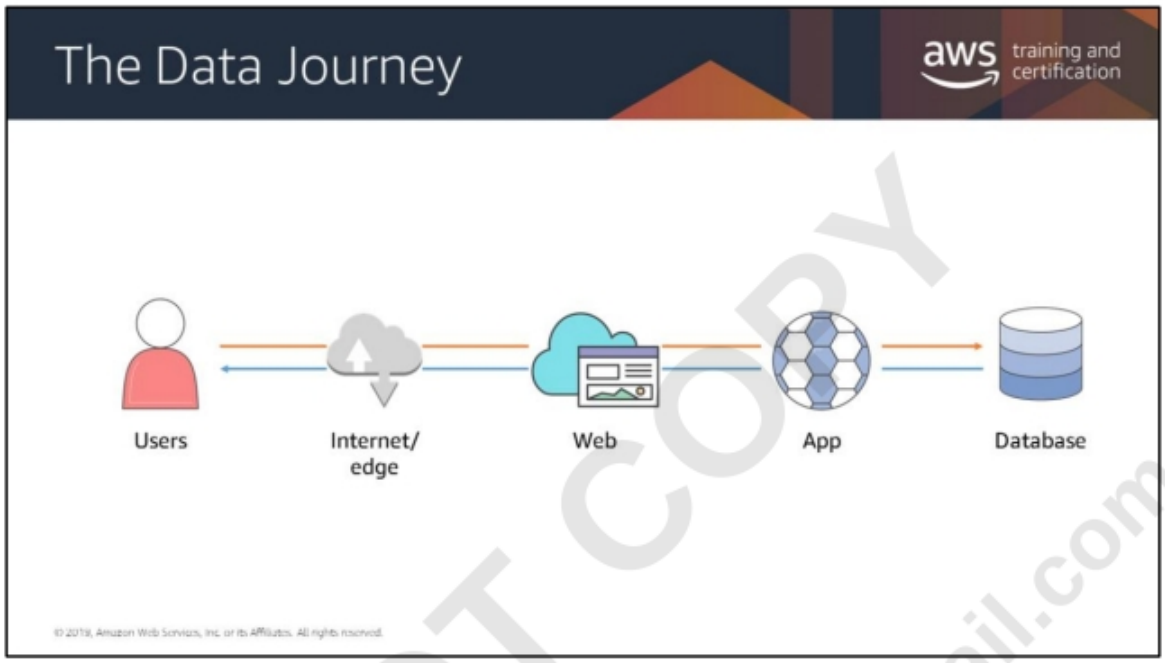
Various techniques can effectively utilize a web cache. The most basic level is client-side web caching. Data is stored in a browser rather than making repeated queries to a web server. HTTP cache headers provide the details on how long the browser can fulfill future responses from the cache for the stored web content.

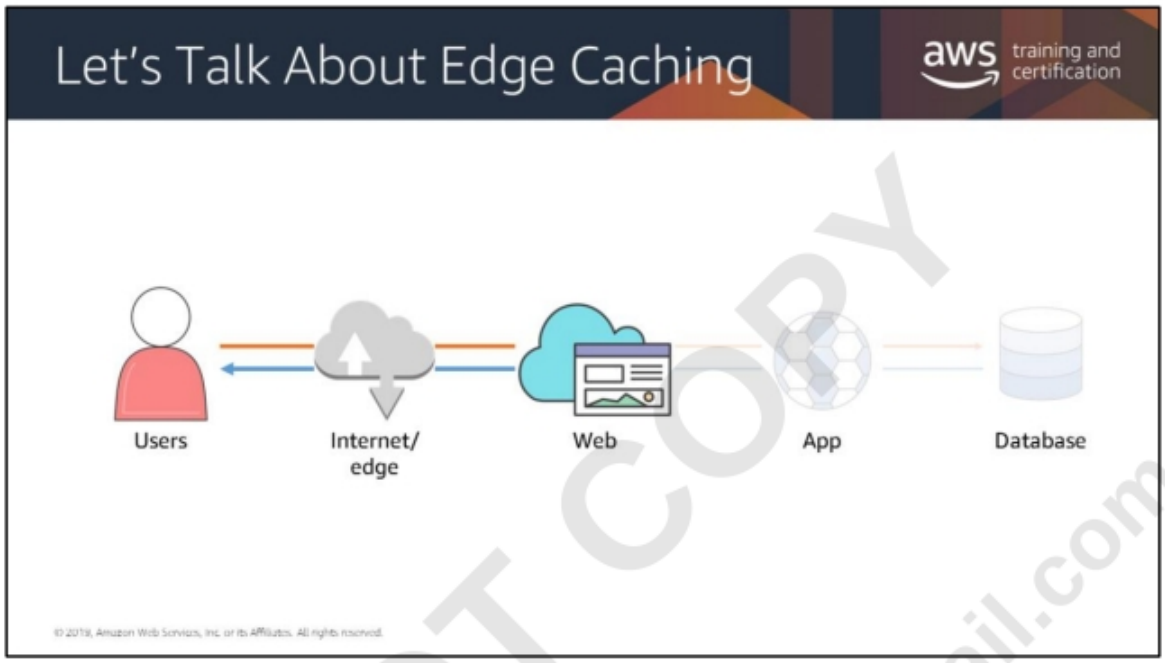
On the server side, various web caching techniques can be used to improve the performance of a website.

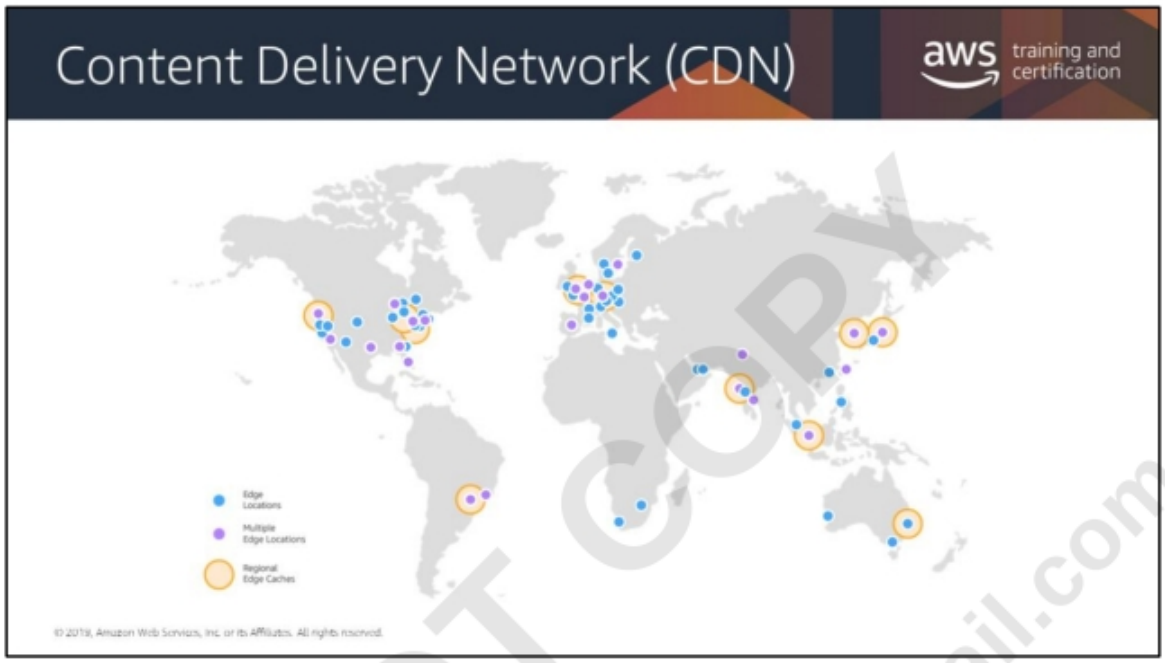
Reverse proxy caches or web application accelerators can be placed in front of application and web servers in order to serve a cached version of the HTTP responses retained from them. These caches are implemented by site administrators and act as intermediaries between the browser and origin servers. They are also commonly based on HTTP cache directives.



/







When your web traffic is geo-dispersed, it's not always feasible (and certainly not cost effective) to replicate your entire infrastructure across the globe. A CDN gives you the ability to use its global network of edge locations to deliver a cached copy of web content (such as videos, webpages, images and so on) to your customers. To reduce response time, the CDN uses the nearest edge location to the customer or originating request location. Throughput is dramatically increased, given that the web assets are delivered from cache. For dynamic data, many CDNs can be configured to retrieve data from the origin servers.

Amazon CloudFront is a global CDN service that accelerates delivery of your websites, APIs, video content or other web assets. It integrates with other AWS products to give developers and businesses an easy way to accelerate content to end users with no minimum usage commitments.

Amazon CloudFront is optimized for both, providing extensive flexibility for optimizing cache behavior, coupled with network-layer optimizations for latency and throughput.

The Content Delivery Network (CDN) offers a multi-tier cache by default, with regional Edge caches that improve latency and lower the load on your origin servers when the object is not already cached at the Edge

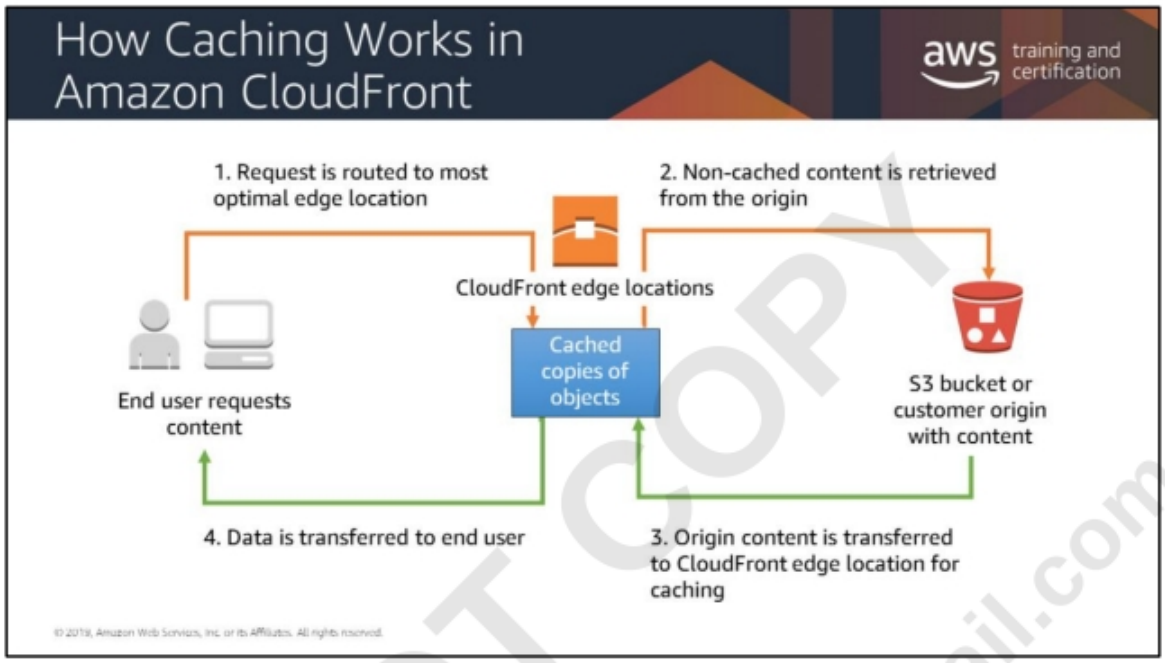
CloudFront provides an easy and cost-effective way to distribute content with low latency and high data transfer speeds. Like other AWS services, CloudFront is a self-service, pay-per-use offering, requiring no long-term commitments or minimum fees. With CloudFront, files are delivered to end-users using a global network of edge locations.

Amazon CloudFront supports real-time, bidirectional communication over the WebSocket protocol. This persistent connection allows clients and servers to send real-time data to one another without the overhead of repeatedly opening connections. This is especially useful for communications applications such as chat, collaboration, gaming, and financial trading.

Support for WebSockets in CloudFront allows customers to manage WebSocket traffic through the same avenues as any other dynamic and static content. CloudFront's global network of edge locations brings traffic closer to users, improving responsiveness and reliability, and also allows customers to take advantage of DDoS protection using the built-in CloudFront integrations with AWS Shield and AWS WAF.

The WebSockets protocol removes the overhead of regular TCP connections by removing the need to constantly open new connections any time a client would like to receive updated data or send new information. WebSocket connections remain open until closed by the client or server, and allows them to send data to one another so long as the connection remains open.

/

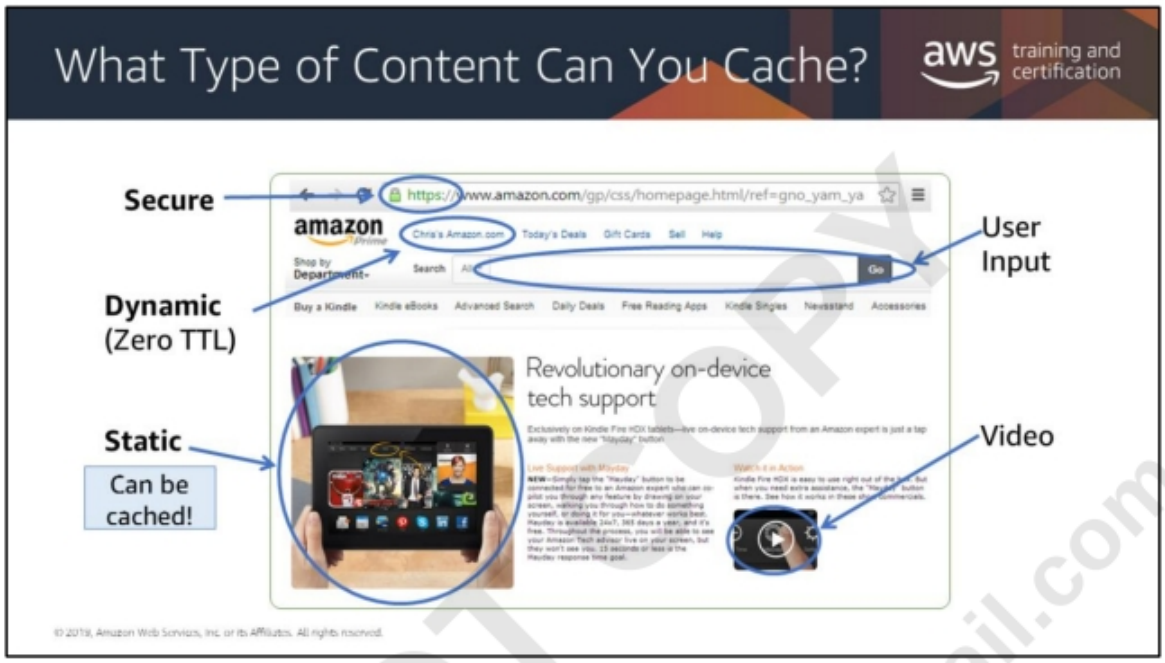


When a user requests content that you are serving with CloudFront, the user is routed to the edge location that provides the lowest latency (time delay), so content is delivered with the best possible performance. If the content is already in the edge location with the lowest latency, CloudFront delivers it immediately. If the content is not currently in that edge location, CloudFront retrieves it from an Amazon S3 bucket or an HTTP server (for example, a web server) that you have identified as the source for the definitive version of your content.

In the example above, if content isn't cached, the requested object is retrieved from the origin. Steps 1, 2, 3, and 4 are performed to retrieve and return the data requested by the user.

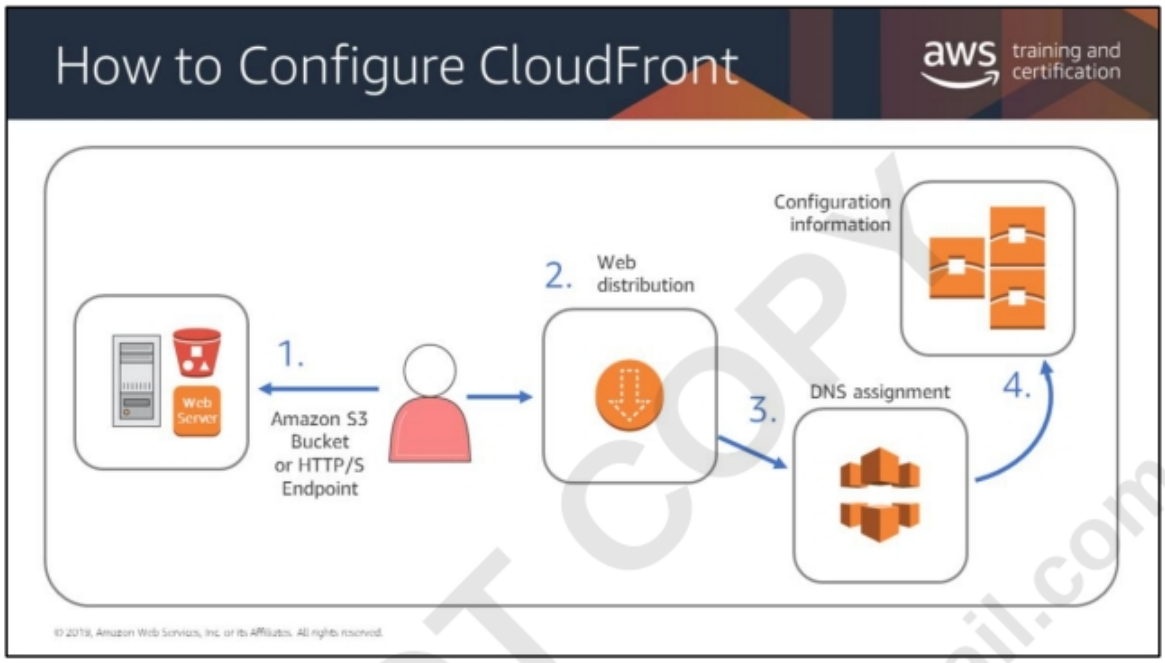
If content is cached, the cached object request is routed to the most optimal edge location and the cached objects are retrieved as shown in steps 1 and 4.

/



With the caching and acceleration technology of CloudFront, AWS can deliver all of your content from static images to user-inputted content.

Static: images, js, html, etc. with high time-to-live (TTL)
Video: rtmp and http streaming support
Dynamic: customizations and non-cacheable content
User input: http action support including Put/Post
Secure: Serve the content securely with SSL (https)



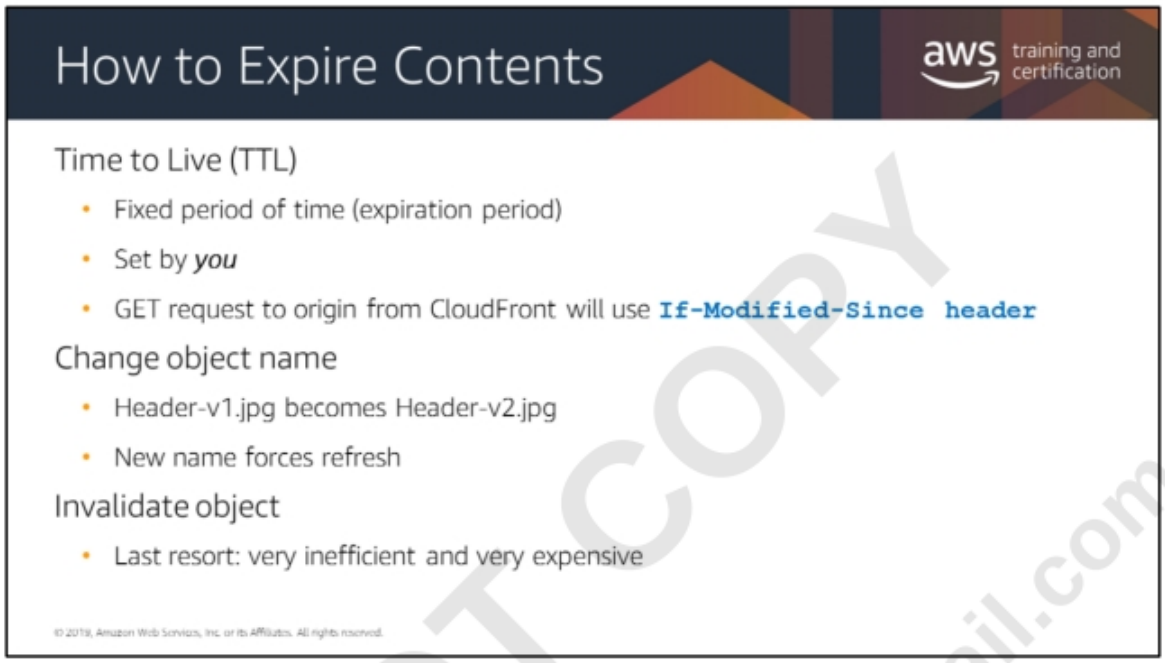
1. You specify an *origin server*, like an Amazon S3 bucket or your own HTTP server, from which CloudFront gets your files. These will be distributed from CloudFront edge locations all over the world.

An origin server stores the original, definitive version of your objects. If you're serving content over HTTP, your origin server is either an Amazon S3 bucket or an HTTP server, such as a web server. Your HTTP server can run on an Amazon Elastic Compute Cloud (Amazon EC2) instance or on server on-premises that you manage. These servers are also known as *custom origins*.

2. You create a CloudFront *distribution*, which tells CloudFront which origin servers to get your files from when users request the files through your web site or application. At the same time, you specify details such as whether you want CloudFront to log all requests and whether you want the distribution to be enabled as soon as it's created.

3. CloudFront assigns a domain name to your new distribution.

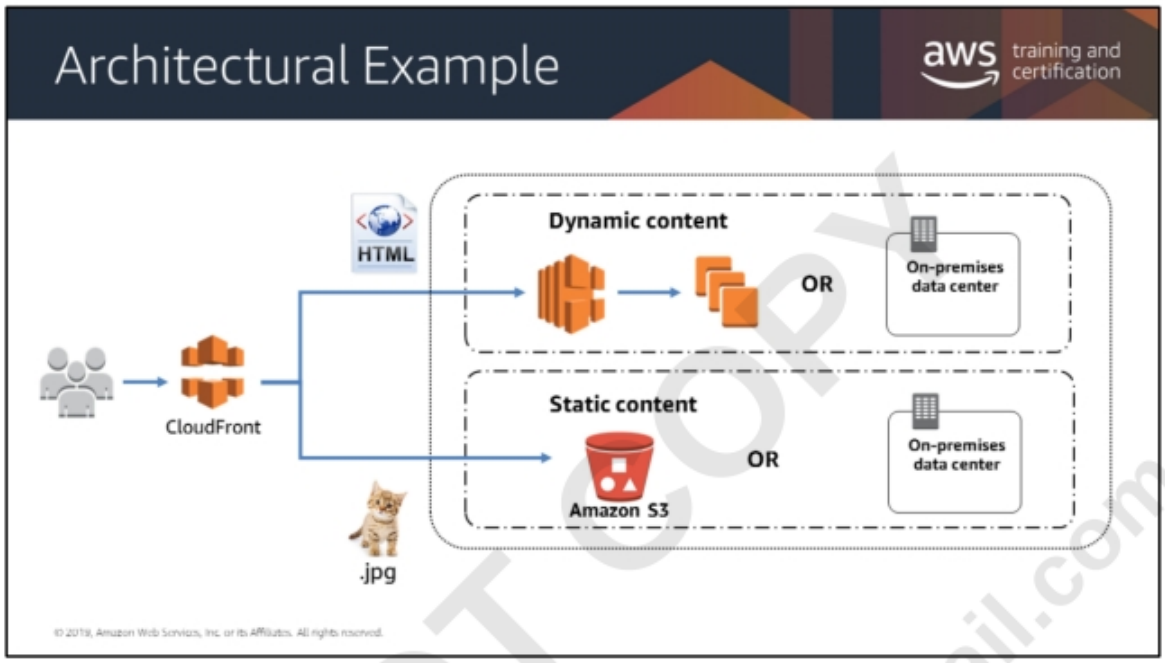
4. CloudFront sends your distribution's configuration but not the content to all of its **edge locations.** Edge locations are collections of servers in geographically dispersed data centers where CloudFront caches copies of your objects.



There are three ways to wear out cached content: the first two are much preferred, and TTL is easiest if the replacement does not need to be immediate. (For more information, see http://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/RequestAndResponseBehaviorS3Origin.html.)

If you set the TTL for a particular origin to 0, CloudFront will still cache the content from that origin. It will then make a GET request with an If-Modified-Since header, thereby giving the origin a chance to signal that CloudFront can continue to use the cached content if it hasn't changed at the origin.
The second way requires more effort but is immediate (there might be some support for this in some CMS systems). Although you can update existing objects in a CloudFront distribution and use the same object names, it is not recommended. CloudFront distributes objects to edge locations only when the objects are requested, not when you put new or updated objects in your origin. If you update an existing object in your origin with a newer version that has the same name, an edge location won't get that new version from your origin until both of the listed events occur.

The third method should only be used sparingly for individual objects: it is a bad solution (the system must forcibly interact with all edge locations).



In general, you only cache static content. However, dynamic or unique content affects the performance of your application. Depending on the demand, you might still get some performance gain by caching the dynamic or unique content in Amazon S3.

Static content off-loading:

- Create absolute URL references to your static assets instead of relative.
- Store static assets in Amazon S3.
- Optimize for "Write Once Read Many."

Additionally, you can geo-restrict your content. Geo-restriction, also known as *geoblocking*, prevents users in specific geographic locations from accessing content that you are distributing through a CloudFront web distribution. To use geo-restriction, you have two options:

- Use the CloudFront geo-restriction feature to restrict access to all of the files that are associated with a distribution and to restrict access at the country level.
- Use a third-party geolocation service to restrict access to a subset of the files that are associated with a distribution or to restrict access at a finer granularity than the country level.

