**AWS** **CloudFormation** - which is an automation product available in AWS.

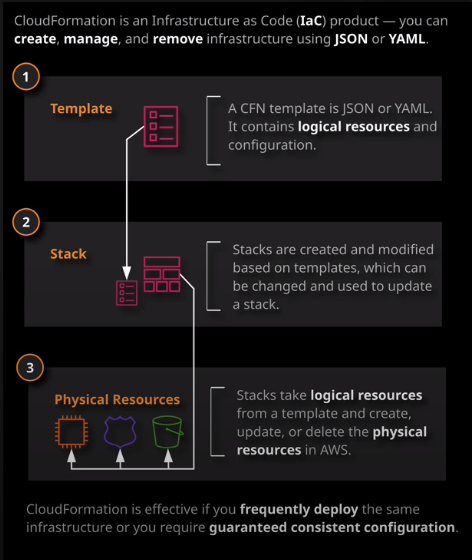
CloudFormation is an IaC or Infrastructure as Code product. What this means is that rather than creating infrastructure manually, using the console UI, or the command line tools to do it for you, you can create the infrastructure using a template and a template is just a file that's written in one of two formats JSON or YAML and you can use this template to create your infrastructure.

If you recall the last lesson so far, we have created infrastructure in AWS, but we've done it in a manual way. So, in the last lesson where I introduce S3 I created these two S3 buckets catpic-ac and dogpics-ac. In this lesson I want to show you how you can utilize CloudFormation to do a similar thing but in an automated way.

Now there are lots of benefits to using infrastructure as code as a solutions architect. Firstly, you can do **the work of planning infrastructure in advance**. You can **test it in a test environment, iron out all the bugs, and then when you come to create it, it's really safe to do so**. If you create the infrastructure using a **template, then you can reuse it**. You could create a template that creates a cat pictures blog and use that template to create 100 Cat Pictures blogs. You could do that without much additional effort and who does not want 100 cat pictures blogs. Teams across a business they can reuse the work that other people do so you can build up internally in your business a library of templates. You might have a template that creates a Microsoft Active Directory system. You might have a template that creates an exchange email server or a Citrix desktop deployment. The possibilities are endless but **by building up an internal library of templates, you can reduce the effort required to consistently rollout systems.** If you are consultancy business or a type of business that roles out the same type of infrastructure day after day for different clients then by using CloudFormation, you could potentially **save yourself a lot of time**. Now, because its automated infrastructure can also be created for temporary periods. So if you've got development teams who want to roll out and test new versions of an application then they can use a CloudFormation template to deploy that infrastructure, install the code, test it, and then delete it afterwards. **Infrastructure can be created for one specific job and then removed**.

CloudFormation can also be used for **disaster recovery**, so you can use it **to decrease your recovery time objective because everything is automated, and above all it cleans up after itself**. So, any infrastructure that it creates it can automatically remove afterwards and that is important if you have got an account that you want to keep nice and clean.

Now I want to give you a quick demo of using CloudFormation in this lesson but before I do, let me quickly step through the architecture.



Everything starts with a CloudFormation template which is essentially a JSON or a YAML document, which defines **resources**. As well as defining resources, it can also specify the **configuration** options on those resources. So, if you use it to create an S3 bucket as well as creating the S3 bucket it can control other elements, such as is the bucket public? Does it use encryption? and so on. Now a template at a high level is a simple document.

Now this is a version of a template in the JSON format. So, this is JSON and it represents the high-level elements that a template needs to have some of these are mandatory and some of them are optional.

This is the YAML version of that same document. So, while the format looks slightly differently. They convey the same set of information. There is only one mandatory section of any CloudFormation template and that is the **resources** section. So, we have got the template version, which is AWS's way of future proofing the format.

|  |
| --- |
| { |
|  | "AWSTemplateFormatVersion" : "2010-09-09", |
|  | "Description" : "this template does XXXX", |
|  | "Metadata" : { |
|  | }, |
|  | "Parameters" : { |
|  | }, |
|  | "Mappings" : { |
|  | }, |
|  | "Conditions" : { |
|  | }, |
|  | "Transform" : { |
|  | }, |
|  | "Resources" : { |
|  | }, |
|  | "Outputs" : { |
|  | } |
|  | } |

So the current version is 2010-09-09 CloudFormation does make an assumption if this is not present but if you do want to be specific, you can state this template format version **description**, as the name suggests, allows you to give a free text description to the template. So, it just helps you with the description. That description is available when you are applying the template but the key thing to understand is if you are putting a description in, it does need to immediately follow this AWS template format version.

One key rule that if a template has a description in it needs to immediately follow the template format version. **Parameters** is a way that a template can ask the input from a user. So maybe you have got a template is creating a WordPress blog well the parameters section is a way the template can ask for the size of the blog. So, what amount of CPU and memory is the instance going to have that is going to run the blog. Parameters is a great way for the template to adjust based on input.

The **outputs** part of the template is the way in which CloudFormation can generate some outputs to be delivered back to the user or the automated process that is using CloudFormation, an example of this might be if you're making a WordPress blog. This could contain the URL or web address to access that blog.

The resources section of the template is where we **define our resources**. So, let us look at the natural example. I have got another template here called S3 bucket got JSON.

|  |
| --- |
| { |
|  | "Resources" : { |
|  | "catpics" : { |
|  | "Type" : "AWS::S3::Bucket" |
|  | } |
|  | } |
|  | } |

Now, this is a pretty simple example of a CloudFormation template in that it only contains the resources section and inside resources, it only contains one element, which is cat pics, and you might spot that it says AWS S3 buckets so this creates an S3 bucket for cat pictures.

Templates contain what are known as **logical resources**. So anything that's contained inside the resources section of the template is a logical resource. In this case, we're defining a logical resource called cat pics. Now we can use this template to create a **stack**. Think of a stack as a container. Like a S3 bucket contains objects, a stack contains logical resources together with other things but more on this in a second. **You can use a template to create a stack. You can change a template and use that to update a stack, and you can simply delete a stack. A stack has an important function. It creates physical resources.**

For every logical resource that the stack has and remember, these are defined in the template, it creates a matching physical resource. With the template that we were just looking at, that's a physical S3 bucket in the region that we're in in this account, **the stacks job is to create the physical resources based on the logical resources that the stack has and also to update the physical resources if the logical ones update and in most cases, delete the physical resources if the logical resources are removed**. Keep this in mind the stack's primary role is to keep that link between the logical resources that are in the template and the physical resources there in your account logical to physical.

Now a stack has a name, and that name needs to be unique in that region in your account.

So, when you create a stack, it steps through the template, looks at all of the logical resources, and for each of those, creates a physical resource. In this particular case, it is a simple example of a CloudFormation template with only a single logical resource called cat pics and so the process of creating the stack is itself pretty simple with only a single major step.

Note that in the template we didn't specify a physical name for this resource and when we do that, CloudFormation generates one randomly, and it's always named the name of the stack then the name of the logical resource and then a random string and that's to ensure that it's always going to be unique.

Now, CloudFormation templates can be much more complex than the one we've looked at. If I move back over to my code editor, I've got another example here, which is a JSON CloudFormation template which can create a WordPress instance. So this is something you can use to create a ready made blog, and I'll make sure that this is linked in the lesson description but as you can see this is much more complex. It's got things in the parameter section. If I scroll down, it's got things in the mapping section. It's got resources so a security group and an EC2 instance, and then right down at the bottom it's also got something in the output section, which is the example I referred to earlier in this lesson, where it will actually generate the URL to access the Wordpress instance.

Now stacks can also be updated, not just created. When you update a stack, you're essentially updating the template used to create the stack and then using that new template to update the stack. When you do that stack updates its logical resources if you add a logical resource to the template, then CloudFormation adds a new physical resource. If you remove a logical resource then CloudFormation deletes the corresponding physical resource. If you change a logical resource this is where the risk gets introduced because if you change a logical resource then CloudFormation will change the physical resource. The way that CloudFormation handles this change depends on what the change is. It can handle the change by making it without any disruption and an example of this is that you might add tags onto an S3 bucket and that's something that can happen with no disruption.

With the example of the WordPress instance that I was showing you if you decide to change the size of this EC2 instance. There's a section of this EC2 instance where the size of the instance is defined. So that's this instance type section and if I was to change this in the template and then update the stack, that would do a change with what's known as some disruption and in this case, that would be a restart of the EC2 instance to allow that size to change but some things if we wanted, for example, to change the name of the S3 bucket that requires a replacement of the resource. The original resource is removed and a new one replaces it. Now with the example of changing the name of an S3 bucket that might not be all that critical but if you're changing something on a database that could potentially cause data loss.

The template contains a logical resources. A template can be used to create a stack. A stack contains those logical resources but also it creates and updates physical resources based on those logical resources. When you update stack it updates those physical resources and when you delete the stack it will, of course, delete those physical resources as we just saw with those two S3 buckets.

