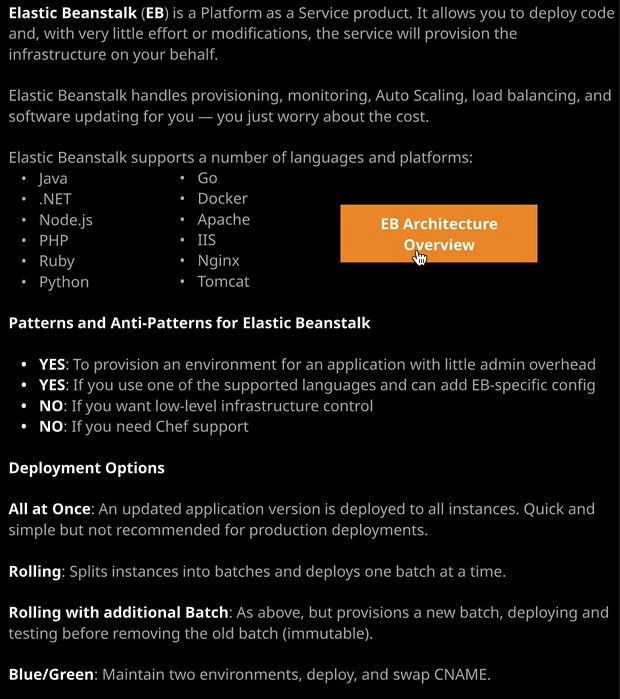
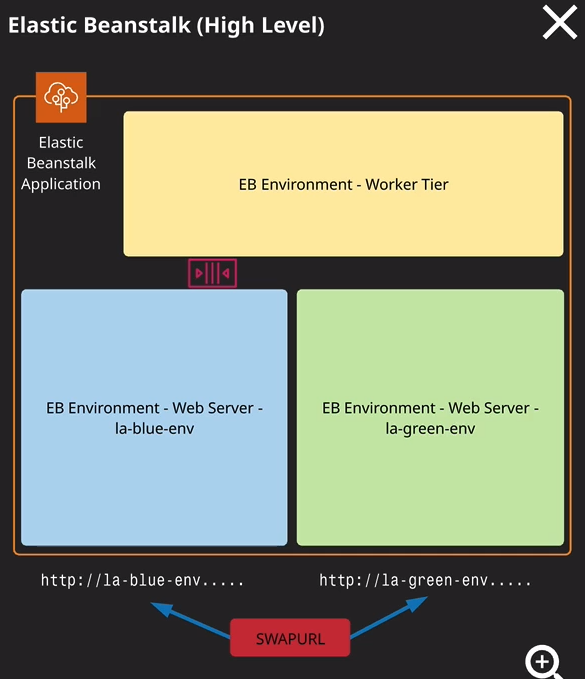
CloudFormation is **infrastructure as code**.

**You can create a template and use the template to create the same infrastructure over and over again in a consistent repeatable way. CloudFormation can also be used as part of the CICD system to create temporary infrastructure. Imagine every time a committee is made to a repository a version of that code could be used together with a CloudFormation template to create a test environment.** Maybe it's rolling some automated test suite, and you've also learned that confirmation could be used for drive purposes. Taking a backup and a template It'd architecture and spinning up a brand new environment in a new region if you have major systems problems but CloudFormation is still very close to the infrastructure. You have a lot of flexibility, but there is still a lot of admin overhead. You need to create the template understand the basic or advanced syntax as well as having to manage the resultant infrastructure.

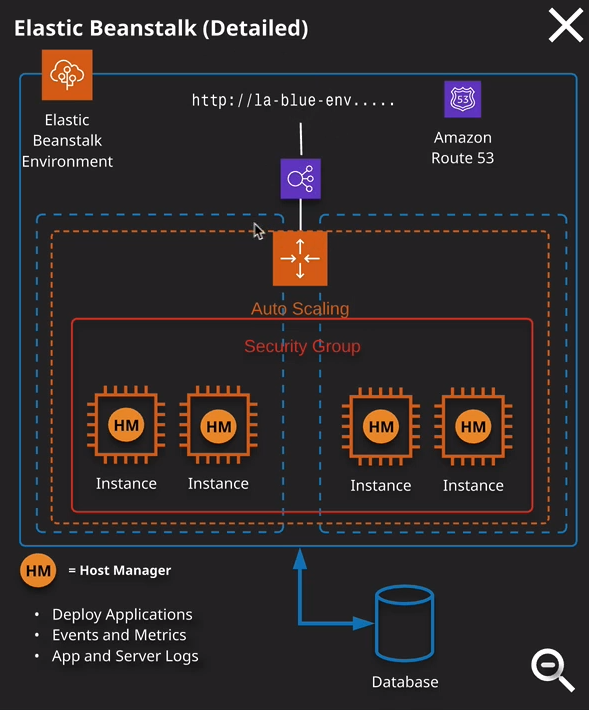


**Elastic Beanstalk is a platform as a service product. Now that means that it manages the infrastructure for any code that you or a developer provide. It supports Java, .net, node.js, PHP, Ruby, Python, Go, Docker, Apache LS, Nginx, and Tomcat based applications on the product provides automated provisioning, monitoring, auto scaling, load balancing, and updating**.

So for the exam, what you really need to know about Elastic Beanstalk is when you would select to use it as an architecture**. If you need load level infrastructure control than Elastic Beanstalk isn't the best tool if you need to deploy applications in a completely unaltered way then Beanstalk is not the right solution because code does need to be designed for Beanstalk or tweet for being stuck in order to work correctly.** So if you see an exam question **where you absolutely cannot alter the code, then Beanstalk is not the right solution.**   
Now, **if you're a developer and use one of the support of runtimes, then Beanstalk is potentially a great solution, and if the scenario that you face in an exam question states that you want no admin overhead or the absolute minimal amount of admin overhead then Beanstalk is probably the right solution as long as your application meets all of the other criteria**. Now for the Solutions Architect Associate exam and you don't need to know exactly how Beanstalk works. It's an architecture exam. The developer associate exam will expect you to know a lot more but for the Solutions Architect Associate exam, pretty much the only reason I'm covering in this course is for confidence reasons. I want you to be equipped.



I want you to know what Beanstalk is and the situations where you should and shouldn't use being stock. So let's quickly talk about the key components of Elastic Beanstalk Now, just so I can show you both the architecture on how this works in practice, I'm just going to move across to the Elastic Beanstalk, part of the AWS console. The base entity of Elastic Beanstalk is an **application**, and it's really easy to create an application. You just hit, create type and name for the application. So I'm going to call my application catpics and then the description I'm going to choose all the best cat videos. So nice and simple I'm going to create an application called cat pics. Now, this base entity of Elastic Beanstalk. This application you need to think of this as a container and I don't want you to confuse this application, which is a container with an actual version of your application code that you'll deploy using Elastic Beanstalk. With an Elastic Beanstalk, you can use it to deploy many different applications and if you're using Elastic Beanstalk in production then your developers might have many different applications being used inside Beanstalk that could be cat flicks. They might be dog flicks. There might be snake flicks. Any application that you're running or your developer is running will have its own application container. So that's how we need to think of these applications. **They're just containers now inside an application container or environments.** Now, these can either be components of your application so frontend web tiers, or they can be backend work in tiers and depending on how your internal deployment processes work. If use blue green deployment, for example, then you might have individual environments that represent different stages or different versions of your application, so you might deploy a certain version of your application to the blue tier. You might deploy a slightly newer version to the green tier. Open that up for some people to test, and then when you're ready, you can swap that over and bring into production, and I'll talk about exactly how you can do that later in the lesson so if I go to actions and then create an environment. I'll be prompted to either create a web server environment or a worker environment. Now these two different roles they mirror the examples that I've used elsewhere in the course. So a web server environment might be an environment where you're presented with a login page and you login and you can upload various different cat videos and when you upload a cat video, it might place a message onto an SQS queue for further processing. A worker environment could represent a different part of the application where it polls this SQS queue messages that require processing. So videos that require processing and then auto scales and processes those videos. So you've got two different types of environment tiers available within Elastic Beanstalk, web server environments, and worker environments. I'm gonna go ahead and create a web server environment so I select that option so each environment has an environment name. In this case, it's catflix-env and every environment is given a unique DNS name that could be used to access this environment. So it's based on the environment name. If you look at the diagram on the right of my screen, in this case, we got la for Linux Academy -blue-env and then the corresponding green environment is la-green-env. So every Elastic Beanstalk environment has its own unique DNS name. Additionally, it's on an environment level that you define the platform that you're going to use for that environment. So in this case, I'm going to click the dropdown and select an environment to use. In this case, I'm going to use Python. You can deploy your own application to an environment which I'll talk about in a second just for demonstration purposes, I'm going to go ahead and use the sample application, and then hit create.



Now that will take a little while to create while it's doing that I want talk about the environments themselves. So I'm going to zoom in by clicking on the magnifying glass on the diagram, and this is a high level overview of exactly what's inside an environment. So environments have this unique DNS name, and depending on the configuration you pick, they also have a load balancer. They have an auto scaling group. The auto scaling group is split across multiple availability zones. It has instances that are inside this auto scaling group, those instances are the thing that provides the functionality to your customers running on each of those instances we have the host manager and the host manager is responsible for maintaining any applications or server logs, looking at events and metrics, and actually deploying the applications on to these instances. Now, you don't manage these instances they're controlled completely by the Elastic Beanstalk product via this host manager but this is the architecture inside an environment, and every environment inside Elastic Beanstalk looks very similar to this. It's protected by a security group. It consists of one or more instances. You can scale in and out the environment based on different criteria, and then generally, you'd also have a database that's outside of this Elastic Beanstalk environment because these environments should be treated as transitory, you're going to be deploying different application versions into them, so you don't want to database inside it. You want to reference one from the outside. So now we've got this environment. I could go ahead and open this in a new tab and you'll see that this is just a Python based application that's running inside this Elastic Beanstalk environment. So nothing crazy now if I go back to the application container, we've got this environment now, So one single environment that's running inside this application, I could go ahead and create a new one. So I'm going to create a new environment. It's going to be a web server environment as well. I'm going to call this one a catflix-env-2. Only now I'm going to choose a different platform. So this time I'm going to pick Ruby. Again I'll be using the sample application, and I'll create an environment. So creating multiple environments inside the same application is really easy and in fact, it gives you significant advantages, which I'll talk about a little bit later on in this lesson. From an application container perspective, now you can see that it can have zero or more environments inside this application. We've also got application versions and an application version is a distinct version of your application code that's packaged into a source bundle, and the source bundle might be a zip file or a wire file, depending on what platform you using. So in either case, you've got a source bundle inside the source bundle is a specific version of your application, and you can add this to the application container and then deploy that to specific environments. So it's the application version that you deploy into an environment, and the environments live in the application container. They're really important terms to understand. Now, as always, I do question the logic in some of AWS's naming conventions because it's not the most intuitive way to talk about it. So the application container references a specific application: cat flix, dog flix, snake flix, whatever application your developers or you are architecting, you will have an application. In an application, generally one or more environments and then deployed into those environments single versions of the applications so single application versions and these application versions are contained inside source bundles essentially a zip or a wire file containing your code.

Now, **with application environments you can also configure a lot of different aspects about how that environment is architected, so you're able to modify the size and type of the instance as well as the storage type that it uses. You can change the capacity of the environment, so at the moment I've got a single instance, but you could configure it to be multiple instances you can elect a have a load balancer or not. You can scale the capacity out to whatever number of instances you need to support load, you can change the deployment style. You can change the monitoring and the security that's used. You've got a lot of different options that you can change that allow you to configure exactly how you want this environment to look,** now the reason that that matters from an exam perspective, and I'm only covering this just in case it features in the exam. I'm not entirely convinced that everyone will have a question on the deployment methods of Elastic Beanstalk, but I want to cover them anyway.

You can deploy application versions using Elastic Beanstalk in a number of different ways. You can deploy it all at once, and when you do that, you deploy an application version to all instances inside an environment at the same time. So it's nice and quick. It's simple, but if you do have any problems, you might have an outage, because then you have to deploy an older version or the existing version back to that environment. You can also use rolling deployment, so that takes all of the instances for an environment. It splits them into batches, and it deploys one batch at a time. A slight variation on that is rolling with additional batch, so that is the same process. It splits the instances for an environment up into multiple batches but then, instead of deploying onto an existing batch, it creates a new batch and new set of instances, deploys the new version onto that new batch and then replaces an existing match with that new batch. Once it's done that, it D decommissions the old batch. So that is slightly more expensive because you do maintain an additional set of instances but it means that if you have any problems with the deployment, it simply stops where it is, and it doesn't cause any outage. Now, the last method that you got for deployment is a blue green deployment style. So if I go back to this application, you'll see that I've got two environments. I've got catflix-env and catflix-env-2 and both of these have got the unique DNS name. What I might do is deploy a new application version to catflix-env-2 and let's just assume that that's the green environment on this architecture diagram. Once that deployment is successful, what I can do is go to actions and then it can swap the environment URLs. When I swapped the environment URLs, I can pick two environments, and it will simply swap the URLs that are used those individual environments. So green becomes blue, blue becomes green, so if I deploy a new version of my code to the green environment and then do a cname swap then all of my customers are now pointing at this green environment and if I have any issues, I can simply swap back. So a blue green deployment architecture is probably the safest, but it's also the most expensive because you need to run two independent environments. Now that's pretty much all I wanted to discover about Elastic Beanstalk.