

Setting Up a Deployment Pipeline

In this lesson, you will learn about the deployment pipeline and to set it up using AWS Lambda.

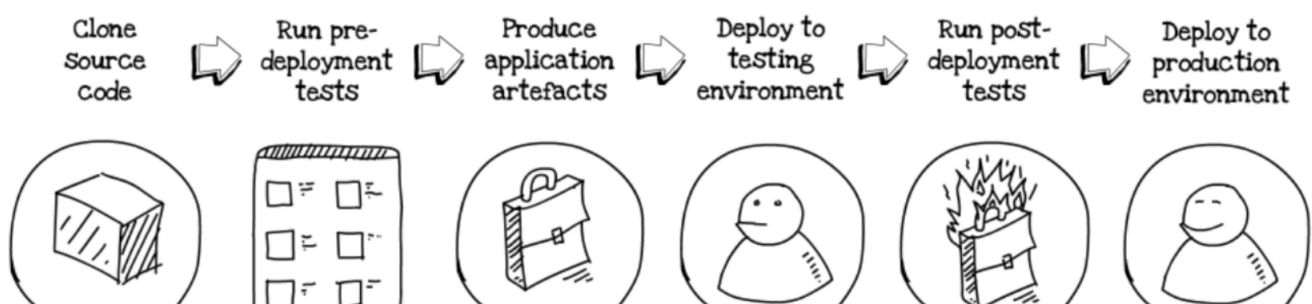
WE'LL COVER THE FOLLOWING ^

- Deployment pipelines
- AWS CodeStar
- AWS CodePipeline

So far, you've always executed the build and deployed commands from a single computer, with the same AWS account used for development. If you work alone or in a very small team, that deployment flow is probably good enough. For an even slightly larger group, though, you'll probably want to set up a deployment pipeline.

Deployment pipelines

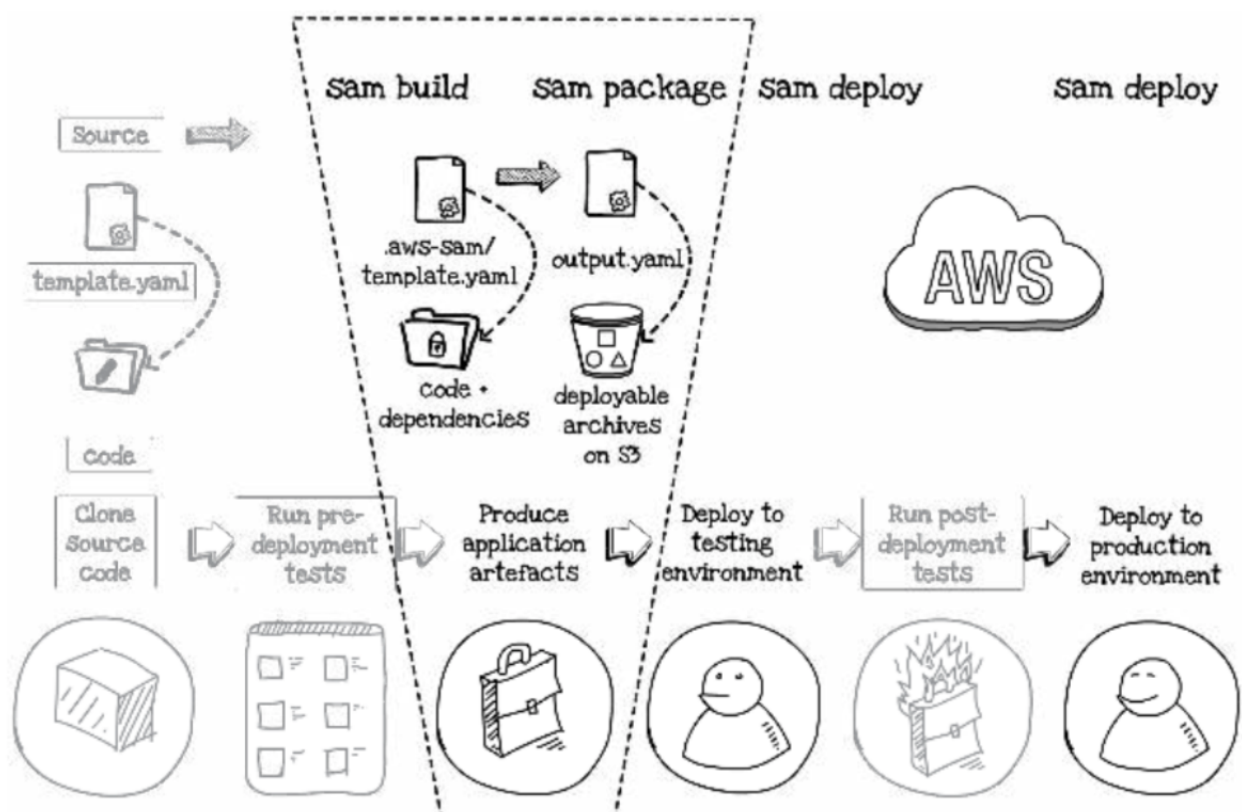
The purpose of a pipeline is to reduce errors by making software releases reproducible and reliable. A deployment pipeline does that by orchestrating the work required to convert source code into a fully deployed application, automating repetitive and error-prone tasks. The deployment tasks usually involve downloading a clean version of the source code, running automated tests, producing application binaries or package artefacts, and installing the application in approval or release environments. Some of those steps might also involve manual approval, requiring humans to verify release candidates or perform additional manual tests.




Delivery pipelines orchestrate tasks to convert source code into a fully deployed application.

A delivery pipeline is closely tied to the workflow of a particular team. It depends on a mix of technology choices including source code repositories and programming language-specific build systems, but also on company or team policies for source code branching, testing procedures, and deployment authorisations. Fully covering all important options for deployment pipelines is outside the scope of this section, as that topic itself would likely require longer than this whole course to cover properly. In this lesson, you'll focus on what's important for pipelines using AWS SAM.

SAM provides convenient tools for development and deployment tasks (see the figure [here](#)), explained in Chapter 3. In the context of most deployment pipelines, running `sam build` and `sam package` steps is sufficient to create deployment artefacts (refer to the figure below). Unlike typical deployment pipelines where compiled code packages pass between steps, `sam package` already uploads deployable Lambda function archives to an S3 bucket, so you don't have to worry about storing those. During the packaging step, SAM will replace local paths from the input template with uploaded package URLs and store the results in an output template (in the previous chapter you used `output.yaml`). Your pipeline needs to keep the output file produced by the packaging step and make it available to CloudFormation during deployment steps.





SAM tools for building and packaging can be used to create application artefacts in a delivery pipeline. The SAM deployment tool can be used to create testing and production environments.

AWS CodeStar

If you're starting from scratch and you don't mind hosting everything inside AWS, check out [AWS CodeStar](#). CodeStar is a visual interface for setting up a full project development environment in AWS, including source code hosting, build systems, and deployment pipelines. It integrates nicely with CloudFormation and AWS SAM, and will allow you to set up a relatively standard delivery pipeline in just a few clicks.

AWS CodePipeline

If you want to host code somewhere outside AWS, but want to run the deployment pipeline inside AWS, consider using [AWS CodePipeline](#). It's a continuous delivery pipeline management system with a convenient wizard setup process, allowing you to define packaging and build steps in a few clicks. The benefit of using CodePipeline is that authentication to access other AWS resources is trivial to set up since it's all controlled by IAM. (CodeStar will actually set up a CodePipeline instance for you.) When using CodePipeline, instead of `sam deploy`, use the built-in AWS CloudFormation deployment provider for CodePipeline and point it to the output template generated by `sam package`.

If you're using a different pipeline product, be sure to upload the output template to S3 after the packaging step. CloudFormation can directly deploy from templates on S3, so just upload the template to the same bucket where `sam package` is sending function code packages. Instead of `sam deploy`, use `aws cloudformation update-stack` and point it to the URL of your output template using `--template-url`.

Regardless of the delivery pipeline product you use, creating a separate account for the pipeline is the easiest way to nicely isolate environments. With CodePipeline, you don't even have to generate API access keys, as it will manage everything through the IAM policies assigned to the sub-account.

Using a single pipeline account would isolate development and deployment

Using a single pipeline account would isolate development and deployment access keys, preventing developers from accidentally touching production

resources. However, the same pipeline AWS account would own the testing and production environments. This is usually fine for small and medium-size organisations. Large corporations often want to isolate production environments from any testing resources, for security reasons and to catch cross-account isolation problems during testing.

To fully isolate application resources, it's best to create a separate AWS account for each environment. Multi-account pipelines are, by design, much more complex than single-account pipelines. Different resources belong to different accounts, so tasks can't just reuse the outputs of other tasks. To avoid rebuilding everything from scratch at every step of your pipeline, consider creating a common S3 bucket for the artefact repository and allowing all the sub-accounts read-only access to that bucket. Check out the page [How can I provide cross-account access to objects that are in Amazon S3 buckets?](#) from AWS support for more information on the required IAM policies for this kind of setup. The start of the pipeline can produce application artefacts by running `sam package` and uploading the executable Lambda packages to the common S3 bucket. All the other sub-accounts can use `sam deploy` to create their own resources. Be sure to also upload the resulting CloudFormation template (`output.yaml` in the previous examples) to the artefact repository.

SAM can also help with deployments by automatically configuring AWS resources for gradual roll-outs. This makes it easy to run multiple versions of an application at the same time and direct traffic from the old version to the new one gradually. You'll look into that in the next chapter.

Now, get ready to move to the 'Interesting Experiments' section in the next lesson!