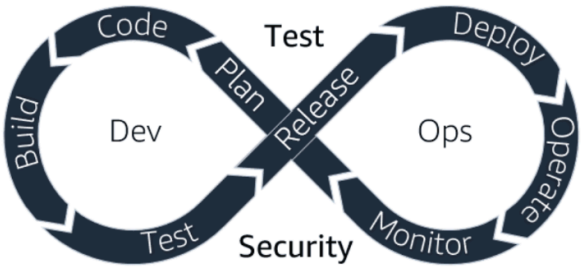
About Course

- Beginner-level course for technical learners in the development and operations domains.
- Basic concepts of DevOps in AWS.
- Learn about culture, practice, and tools used in the DevOps environment.
- Concepts for developing and delivering secure applications at high velocity on AWS.

What is and why DevOps?

With DevOps, teams embrace the culture, practices, and tools to develop and deliver software applications at high velocity compared to traditional approaches to software development. And quickly respond to customer needs.

DevOps emphasizes better collaboration and efficiencies so teams can innovate faster and deliver higher value to businesses and customers.



An infinity loop formed by the software lifecycle stages is a common depiction of the ongoing collaboration and efficiencies suggested by DevOps.

DevOps is short for Development (Dev) and Operations (Ops). Dev are the people and processes that create software. Ops are the teams and processes that deliver and monitor the software.

DevOps brings together formerly siloed roles (development, IT operations, quality engineering, and services) to optimize the productivity of developers and the reliability of operations.

DevOps is a combination of:

- Cultural philosophies for removing barriers and sharing end-to-end responsibility
- Processes developed for speed and quality, that streamline the way people work
- Tools that align with processes and automate repeatable tasks, making the release process more efficient and the application more reliable

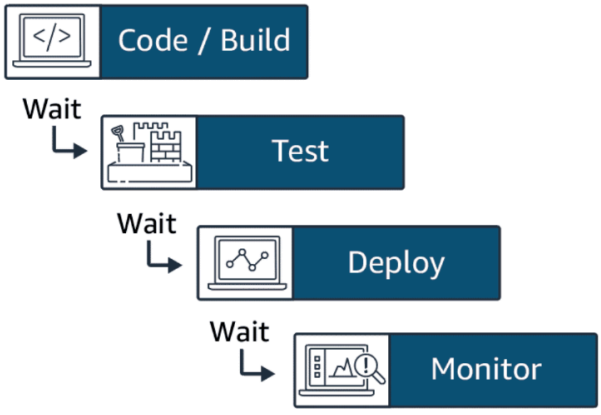
As you go through the course, keep in mind that teams practice DevOps to increase innovation and agility, improve quality, release faster, and lower cost. DevOps improves delivery efficiency and predictability of the application and services.

Traditional ways of software development have proven slow and ineffective. Waterfall developments are slow, not iterative, resistant to changes, and have long release cycles.

Waterfall development projects are slow, not iterative, resistant to change, and have long release cycles. Some reasons for this include:

- Requirements are rigid, set at project start, and will likely not change.
- Development phases are siloed, each starting after the previous phase has ended. Each phase is supported by highly specialized teams.

  Hand offs from one phase to the other are long, often requiring teams to switch tools and spend time clarifying incomplete or ambiguous information.
- Testing and security come after implementation, making corrective actions responsive and expensive.

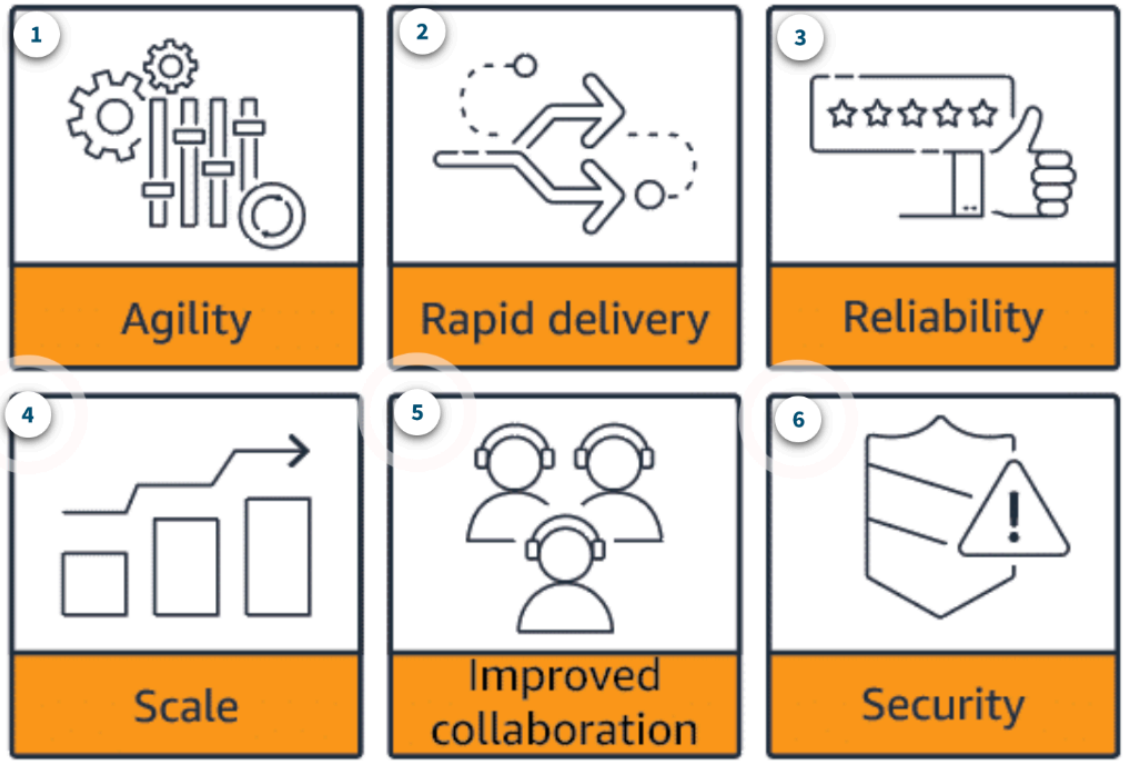


Development phases supported by siloed and specialized teams cause delays and are costly.

Monolithic applications have tightly coupled functionality.

Monolithic applications are hard to update and deploy because they:

- Are developed and deployed as a unit, so when changes are made, the entire application must be redeployed
- Have tightly coupled functionality, so if the application is large enough, maintenance becomes an issue because developers have a hard time understanding the entire application
- Are implemented using a single development stack, so changing technology is difficult and costly



DevOps Methodology

DevOps methodology increases collaboration through the entire service lifecycle.

DevOps culture is a culture of transparency, effectiveness, seamless collaboration, and common goals.

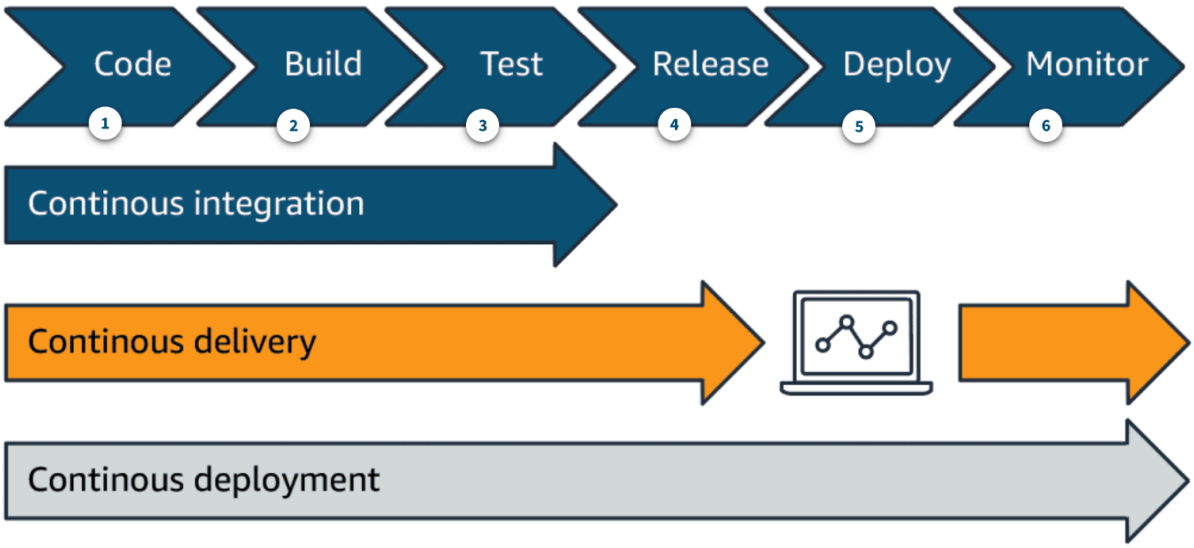
Core Principles of DevOps Culture

There are seven core principles:

1. Create a highly collaborative environment
2. Automate when possible
3. Focus on customer needs
4. Develop small and release often
5. Include Security at every phase
6. Continuously experiment and learn
7. Continuously improve

DevOps Practices

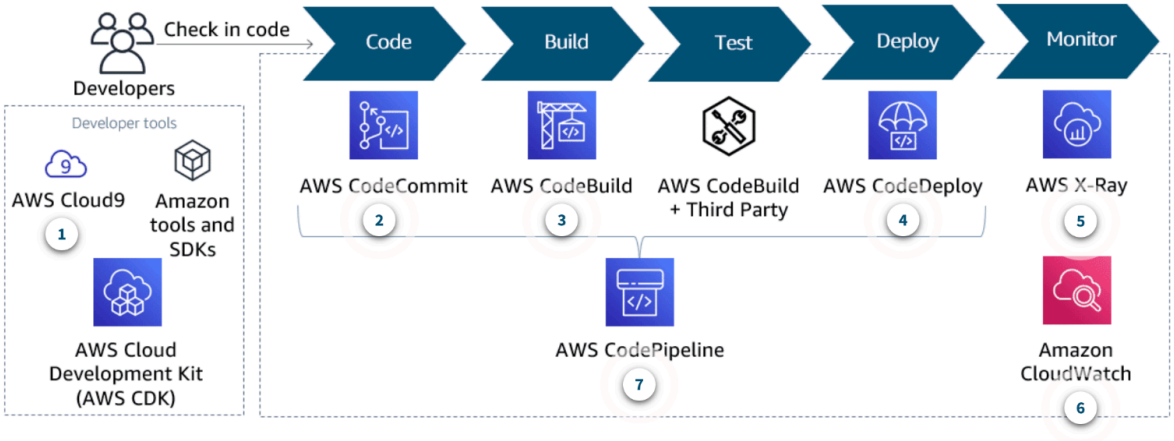
- Communication and collaboration
- Monitoring and observability
- Continuous Integration (CI)
- Continuous Delivery/Continuous Deployment (CD)
- MicroServices Architecture
- Infrastructure as Code (IaC)



Amazon transformation from traditional software development

Amazon transformed from traditional software development to DevOps. From monolithic architecture (silos) to microservices architecture (service-oriented) (cross-functional teams).

AWS DevOps Tools



AWS Cloud9: Write, run, and debug your code
AWS CodeCommit: Securely store and source control your code

AWS CodeBuild: Compile source code, run tests, and produce software packages
AWS CodeDeploy: Deployment service that automates software deployment to a variety of compute services, such as AWS EC2 and AWS Lambda
AWS X-Ray: Collects data and provides tools to gain insights
AWS CloudWatch: Monitor AWS resources and the applications you run on AWS in real time
AWS CodePipeline: Fully managed continuous delivery service. Automates the build, tests, and deploy phases

AWS CodePipeline

AWS CodePipeline is a continuous delivery service that enables you to model, visualize, and automate the steps required to release your software.

Why use?
With CodePipeline you can:

- Capture and visualize your pipeline, run it, view real-time status, and retry failed actions.
- Automate your release processes, eliminating human error, speed up delivery, and improve the quality of the release.
- Establish consistency in the release.
- Incorporate your source, build, and deploy tools.
- View pipeline history details.
- Integrate with third-party and AWS tools to build, test, and deploy your code when notified of a code change.

Monitoring
You can monitor your pipeline in a number of ways to assure its performance, reliability, and availability, and to find ways to improve it. You can monitor the pipeline directly from the AWS CodePipeline console, the command line interface (CLI), use Amazon EventBridge, or AWS CloudTrail.

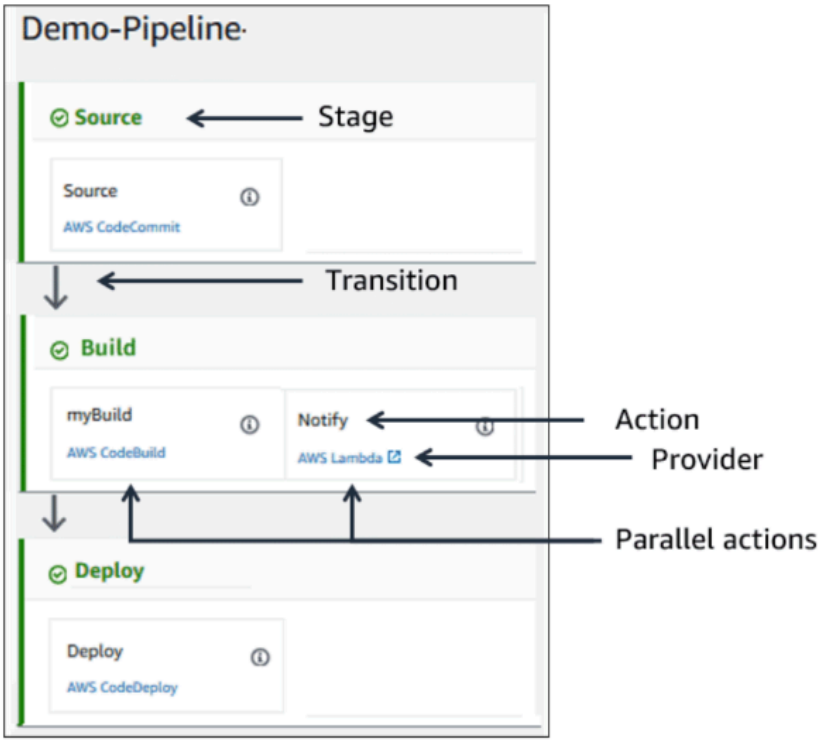
Security
Security is an important part of any pipeline. CodePipeline supports resource-level permissions, enabling you to specify which user can perform what action on the pipeline. Some users might have read-only access to the pipeline, while others might have access to a particular stage or action within a stage. For more information about security, see Security in AWS CodePipeline.

How it works

With CodePipeline, you model your release process with a number of stages and actions. CodePipeline breaks up your release workflow into a series of stages, such as code, build, and test. Each stage can have a number of actions that need to be performed.

- Actions are tasks that can run in sequence or in parallel to each other. Actions are associate with a service provider that runs the action, or they can require user intervention. Service providers can be AWS services (like CodeBuild, Amazon Simple Storage Service (Amazon S3), AWS Lambda, and AWS CloudFormation), or third-party services (like Jenkins and TeamCity).
- Action types include:
    - Source (where the source is stored)
    - Build (how to build the application)
    - Test how to test the application)
    - Deploy (how to deploy the application)
    - Approval (manual approval and notifications)
    - Invoke (Invoke a custom function).

Following is an example of a three-stage pipeline:



AWS CodeCommit

AWS CodeCommit is a fully managed source control service that hosts secure Git-based repositories. A repository is a fundamental version control object in CodeCommit, and it stores your project files and source code.

Why use?
With CodeCommit you can:

- Eliminate the administrative overhead of managing your own hardware and software needed to store your code. CodeCommit is fully managed, highly available, and has no limits in the type or size of files it can store.
- Collaborate with your team on code using Git commands they already know. CodeCommit is a secure Git-based repository that can handle large numbers of files, code branches, and lengthy revision histories.
- Improve your existing workflow by integrating CodeCommit with other AWS services, IDEs, and third-party software (for example, Jira).

Monitoring
You can create notifications and trigger actions based on events such as creation of a branch, or when a commit is made.

Security
From a security perspective, access to the repository is managed with policies and user accounts. A version of the repository is found in the cloud, and can be assessed through HTTPS or Secure Shell (SSH). The data is encrypted at rest and in transit. For more information about security, see Security in AWS CodeCommit.

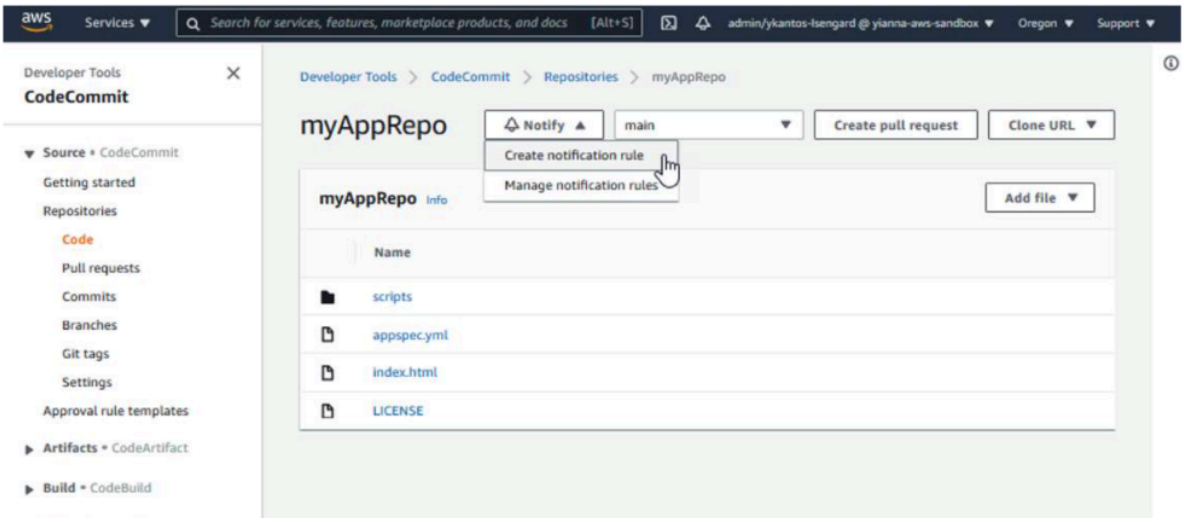
How it works

In CodeCommit, you create a repository. Users clone it to their environments, creating their own copy of the repository. After making changes, they push those changes back to the CodeCommit repository.

The CodeCommit repository is managed with Git source control. Git commands (such as git add, git push, git branch) can be used to work and collaborate on the code.

If it is part of a pipeline, CodeCommit can start the pipeline when a new code change is made on the configured CodeCommit repository and branch.

Below is a screen-capture of a CodeCommit repository named myAppRepo, and the files it contains. You are looking at the main branch. Notifications can be created and sent, based on what is happening in your repository. Note the Developer Tools panel which can be used to access other services, like CodeBuild, CodeDeploy, and CodePipeline.



CodeBuild

AWS CodeBuild is a fully managed build service that automatically compiles source code, runs tests, and produces software packages.

Why use?
With CodeBuild you can:

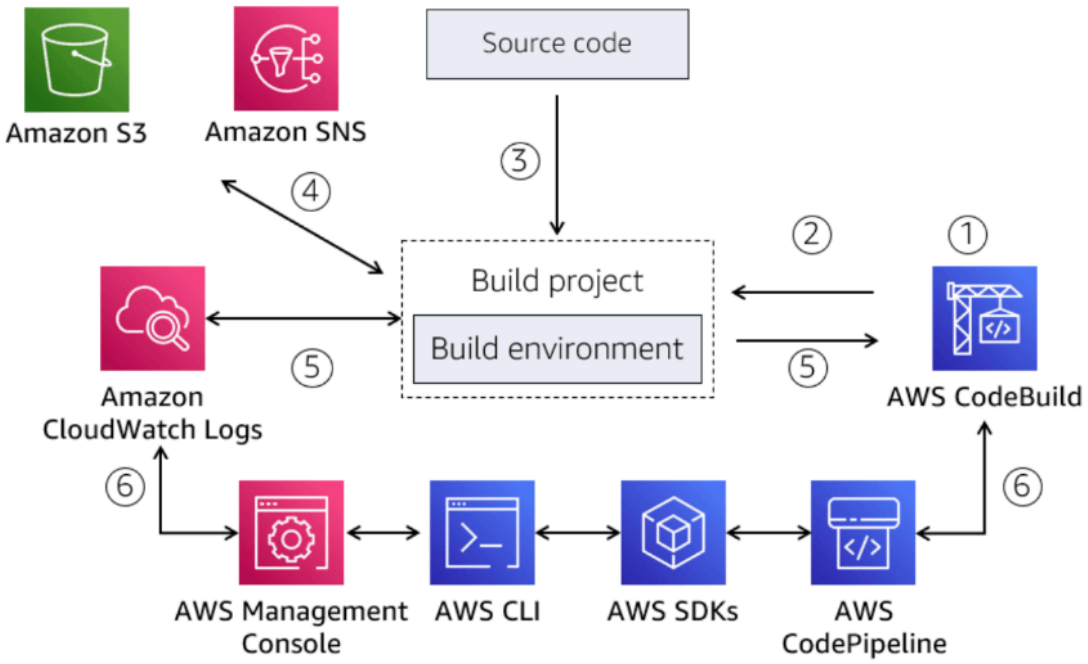
- Eliminate the need to set up, patch, update, and manage your own build servers, since CodeBuild is fully managed.
- Automatically compile source code, run tests, and produce build artifacts.
- Specify build commands to run at each phase of the build.
- Process multiple builds concurrently, for example, developers can continuously build and test their code, catch errors early, and correct them early.
- Leverage out of the box preconfigured build environments (such as .NET Core, Java, Ruby, Python, Go, NodeJS, Android and Docker). Build environments contain the operating system, programming language runtime, and build tools (such as Apache Maven, Gradle). You can also provide custom build environments suited to your needs by means of Docker images.
- Pull source code from CodeCommit, Amazon S3, GitHub, GitHub Enterprise, and Bitbucket.
- Integrate CodeBuild with Jenkins to simplify the build process.

Monitoring
You can monitor the build through the CodeBuild console, CloudWatch Logs and a number of other ways.

Security
Build artifacts are encrypted and you control access to your build projects through resource-level permissions in AWS Identity and Access Management (IAM) policies. For more information about security, see Security in AWS CodeBuild.

How it works
CodeBuild uses information provided by a build project to create a build environment (a provisioned Docker container created fresh on every build) where it then runs the build. Here is an overview of the steps that CodeBuild takes during the build.

1. CodeBuild uses a build project to run a build. A build project contains information needed to run the build, such as source repository location, runtime environment, build command, and where to store the build output. To access CodeBuild, use either the AWS Management Console, AWS Command Line Interface (AWS CLI), AWS SDKs, or CodePipeline.
2. CodeBuild uses the build project to create a build environment.
3. CodeBuild downloads the code and uses a build specification (buildspec) file to run a build. A buildspec is a collection of build commands that will run at each phase of the build (like installing tool packages, running tests, or packaging your code) and related settings.
4. Any build output is uploaded to an Amazon S3 bucket and able to configure notifications.
5. Build output is streamed to the service console, and CloudWatch Logs.
6. Monitor the progress of the build through CloudWatch or other services.


Amazon S3
Amazon SNS
Source code
3
4
Build project
Build environment
2
1
5
5
Amazon
CloudWatch Logs
AWS CodeBuild
6
6
AWS Management
Console
AWS CLI
AWS SDKs
AWS
CodePipeline

CodeDeploy

AWS CodeDeploy is a fully managed service that automates your software deployments, allowing you to deploy reliably and rapidly. It automates code deployment to a variety of compute services, including Amazon Elastic Compute Cloud (Amazon EC2), AWS Fargate, AWS Lambda, or on-premises servers.

Why use?
With CodeDeploy you can:

- Deploy server, serverless, or container applications.
- Automate deployments and eliminate the need for manual, error-prone operations. With CodeDeploy you can reliably and rapidly release new features and updates.
- Deploy on a variety of compute platforms including: AWS Lambda, Amazon ECS, Amazon EC2, or on-premises. You can even configure CodeDeploy to deploy to an Amazon EC2 Auto Scaling group, which will prepare the environment before traffic is sent to it.
- Concurrently deploy to one or multiple instances as the service scales to fit your needs.
- Minimize production downtime for your application by specifying if an update will be applied on an existing instance, or a newly provisioned environment that will replace the previous environment. You can also control how to handle the traffic-shifts from older to new versions. For example, if your application needs at least 50% of the instances in a deployment group to be up and serving traffic, you can specify that in your deployment configuration so that a deployment does not cause downtime.
- Automatically (or through user intervention) stop an unsuccessful deployment and roll back your deployment to a previous version.

Monitoring
AWS provides various tools that you can use to monitor CodeDeploy, including CloudWatch alarms, the CodeDeploy console and more.

Security
You can configure CodeDeploy to meet your security and compliance objectives. For more information on security, see Security in AWS CodeDeploy.

How it works
To automate the deployment to the appropriate compute resources, CodeDeploy needs to know which files to copy, what scripts to run, and where to deploy.

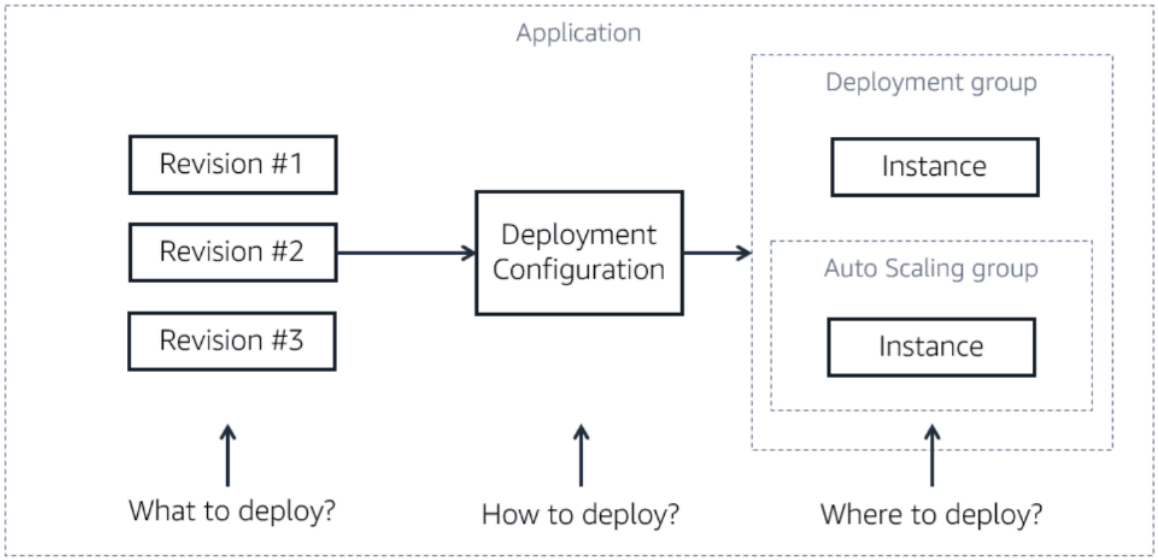
The concept of an *application* is used by CodeDeploy to ensure it knows what to deploy (correct revision of code), where to deploy (deployment group), and how to deploy (deployment configuration).

- **Code**
    - Identify the correct version (revision) of the code.
    - With the code, you provide an application specification file (AppSpec file) which is used to manage each deployment. During deployment, CodeDeploy looks for your AppSpec file in the root directory of the application's source.
    - The AppSpec file specifies where to copy the code and how to get it running. For example, it tells CodeDeploy how to stop the application if it is already running, how to install the code, what command to run before and after the code is installed, and how to get the application running again.

- **Deployment group**
  - A *deployment group* specifies the deployment targeted environment. The information it contains is specific to the target compute platform: AWS Lambda, Amazon ECS, Amazon EC2, or on-premises. For example, Amazon ECS lets you specify the Amazon ECS service, load balancer and more. For Amazon EC2, it is a logical group of deployment target instances or physical environments.
  - A CodeDeploy application can have one or more deployment groups.
  - Security needs to be assigned so the environment can communicate with CodeDeploy.
  - The CodeDeploy agent is needed if you are deploying to Amazon EC2 or an on-premises compute platform. It is installed and configured on the target instances. It accepts and executes requests on behalf of CodeDeploy.

- **Deployment configuration**
  - A *deployment configuration* is a set of deployment rules and deployment success and failure conditions used by AWS CodeDeploy during a deployment. For an Amazon EC2 compute platform, it specifies the number or percentage of instances that must remain available during deployment. It also specifies if an instance in the deployment group is briefly taken offline and updated with the latest code revision, or if a new instance replaces the instances in the deployment group.

The diagram below is an example of a deployment to an Amazon EC2 compute environment.



Create and Control a CI/CD Pipeline

The application being deployed is a simple web page. This release pipeline will automate deploying a working application to multiple regions.

In general, the CI/CD pipeline is deployed using AWS CodePipeline. This pipeline uses CodeCommit, CodeDeploy, and AWS Lambda.

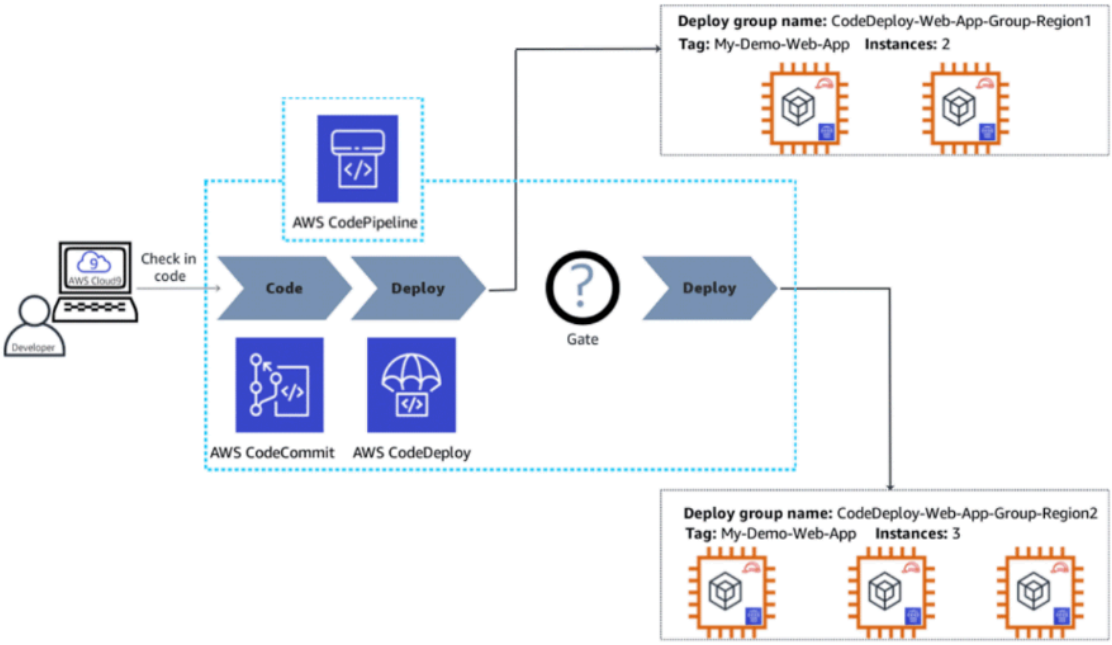
Demo high-level overview

These steps have been completed before the demo:

- The infrastructure required for the application to run was provisioned in two AWS Regions.
- Along with the application source, an appspec.yml file and supporting scripts have been created and provided with the code.
- A Lambda function has been created. When invoked, it checks some text on a webpage.

During the demo, the pipeline will be created in the following steps:

1. The demo starts with a quick review of the provisioned infrastructure and the template file that was used to create them.
2. AWS CodeCommit is configured to hold the code, and is the continuous integration service that starts the pipeline with every code change.
3. AWS CodeDeploy is configured with specifics about each deployment Region. CodeDeploy is the continuous deployment service that installs the application on the infrastructure.
4. AWS CodePipeline uses the configured services to create pipelines. First, a two stage pipeline is created that automatically deploys to Region 1. Then, we build on the pipeline and add two new stages. This release pipeline will deploy the simple web application to two distinct AWS Regions. The pipeline still automatically deploys code changes to Region 1, but requires a manual approval before deploy to Region 2. Finally, a Lambda function is used speed up the pipeline while maintaining control. It replaces the manual approval gate with automation.



Services referenced in the demo

The following services are depicted in the demo:

- AWS Cloud9 is used as an IDE to develop and make any code changes.
- AWS CodeCommit is used to create a source control repository to host the demo code.
- AWS CodeDeploy is used to deploy the demo application package to Amazon EC2 servers across multiple AWS Regions (first to us-west-2 and then to us-east-1).
- AWS CodePipeline is used to orchestrate the phases of the CI/CD pipeline. The pipeline uses Amazon CloudWatch Events to automatically start the pipeline when source changes are submitted to the repository.
- AWS CloudFormation was used prior to the demo, to create the infrastructure and supporting resources for the demo application environment such as Amazon EC2 servers, security group to control the traffic to those instances, deploy scripts, and Deployment groups to host the web application in multiple Regions.
- Amazon EC2 uses Amazon Linux instances to hosts the webpage. Amazon EC2 is a web service that provides secure, resizable compute capacity in the cloud.