JON BONSO AND KENNETH SAMONTE

AWS CERTIFIED DEVOPS ENGINEER PROFESSIONAL

Tutorials Dojo
Study Guide and Cheat Sheets

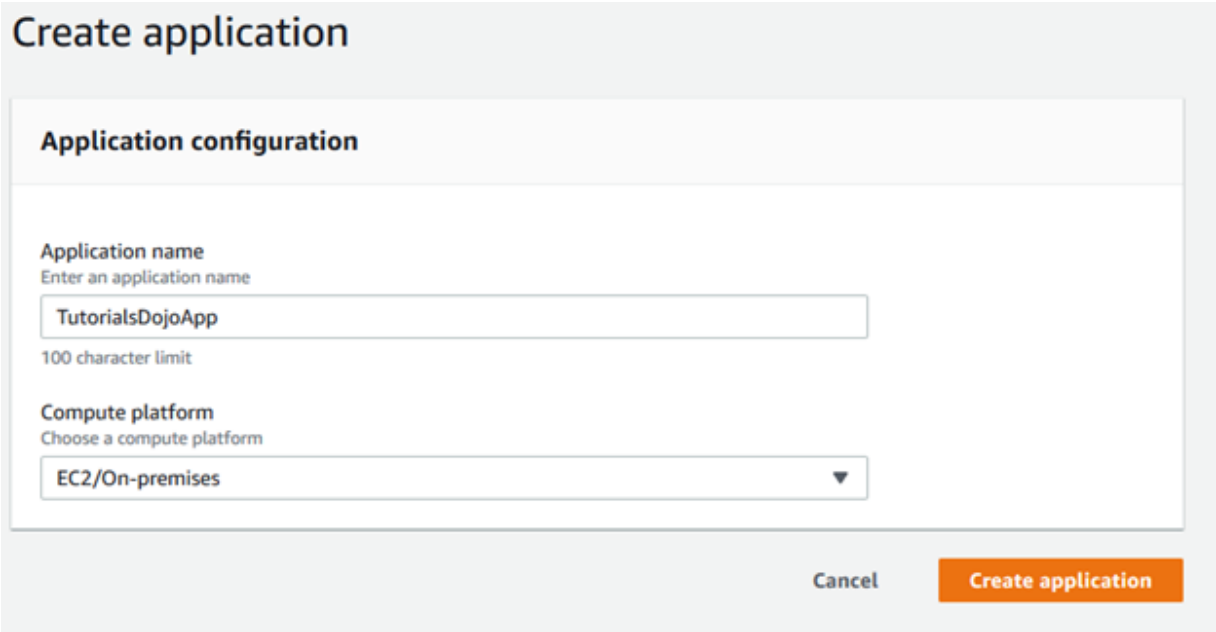
CodeDeploy with CloudWatch Logs, Metrics, and Alarms

AWS CodeDeploy allows you to automate software deployments to a variety of services such as EC2, AWS Fargate, AWS Lambda, and your on-premises servers.

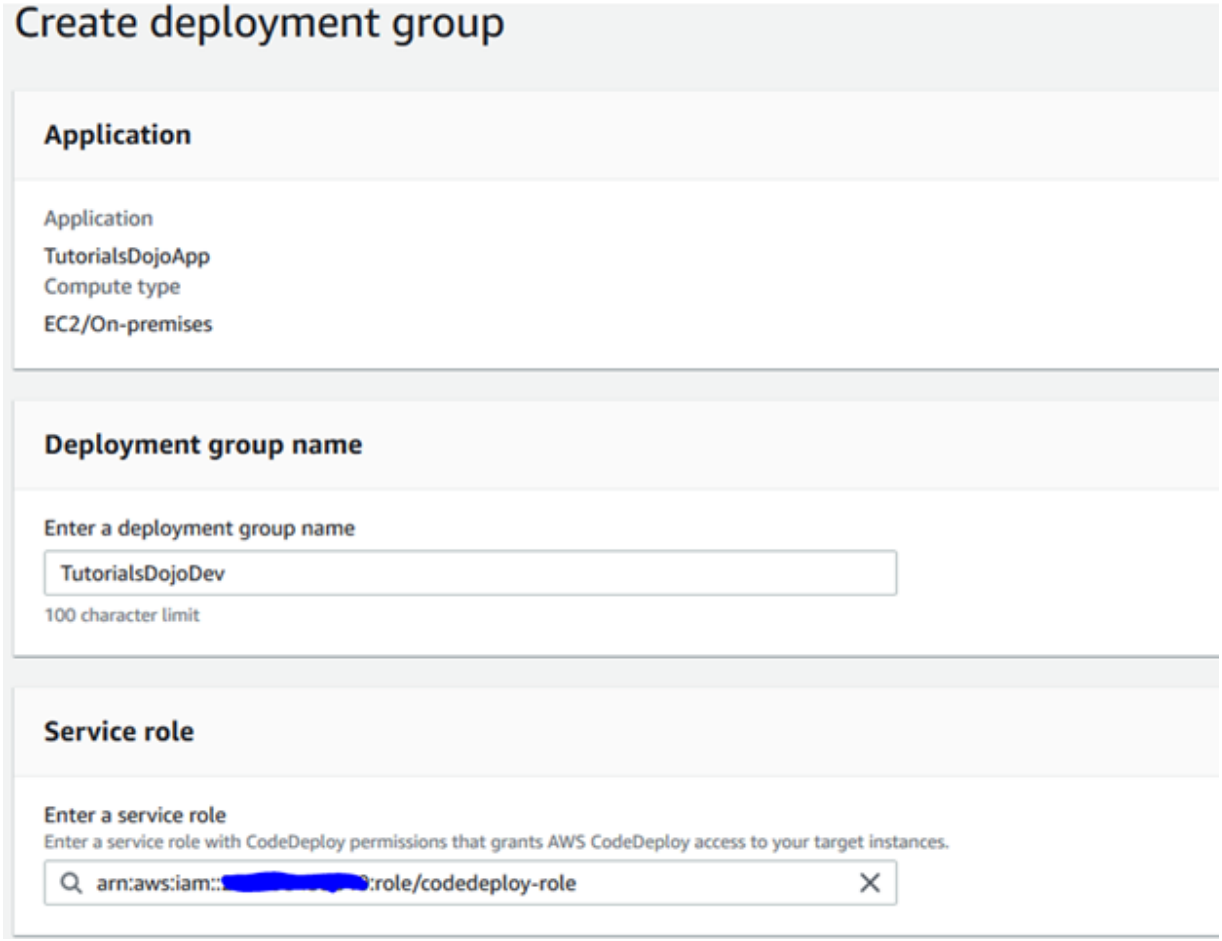
For example, after you have produced from AWS CodeBuild, you can have CodeDeploy fetch the artifact from the AWS S3 bucket and then deploy it to your AWS instances. Please note that the target instances need to have the AWS CodeDeploy agent installed on them for this to be successful and have proper Tags.

Here are the steps to create a deployment on AWS CodeDeploy, including a discussion on how it integrates with AWS CloudWatch.

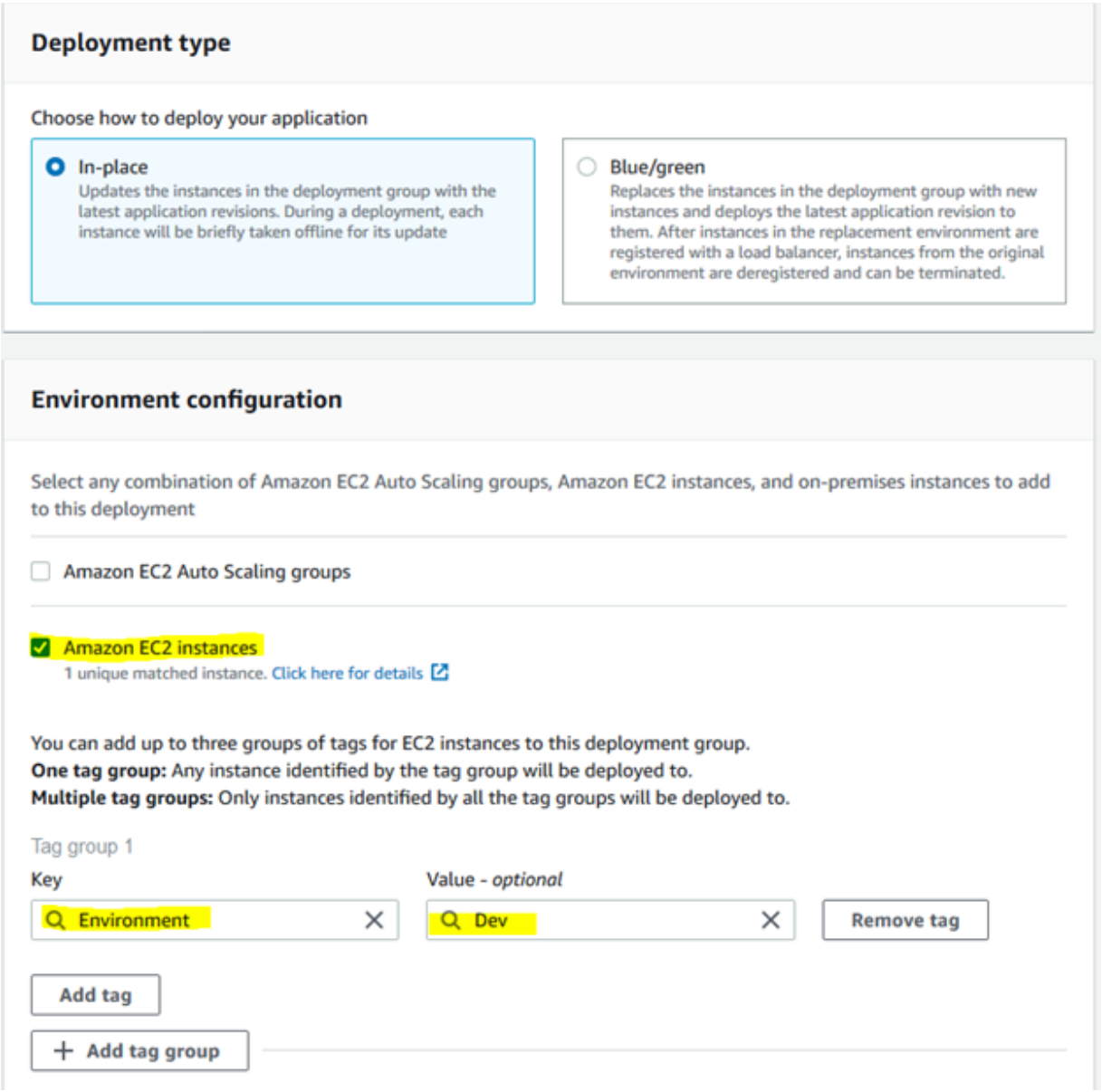
1. Go to AWS CodeDeploy > Applications and click “Create Application”. Input details of your application and which platform it is running.



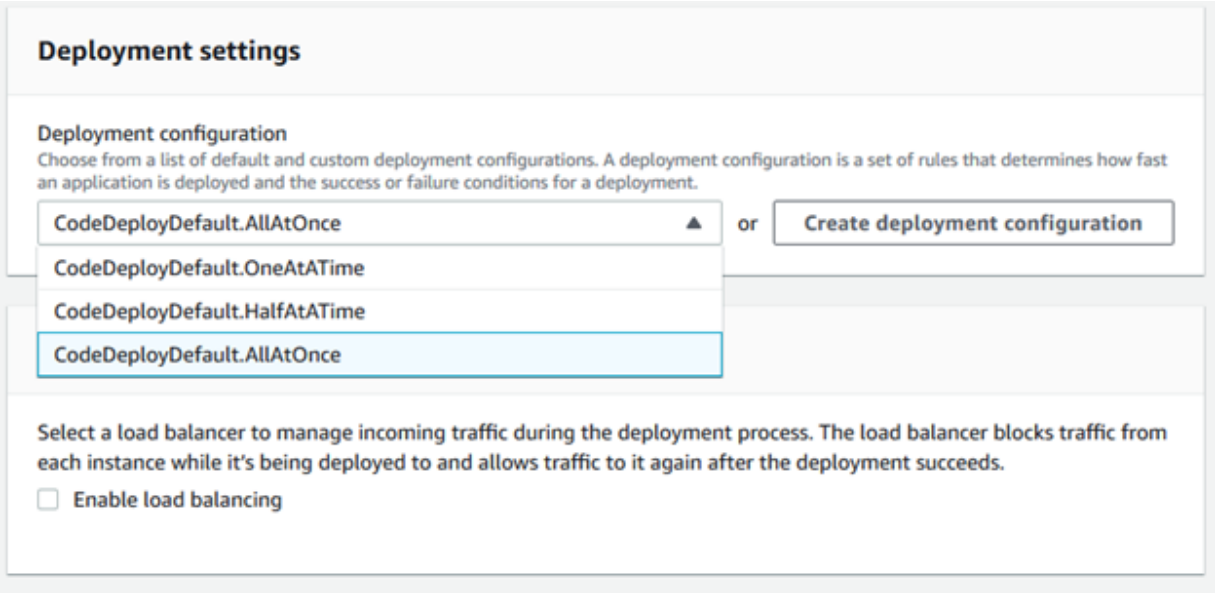
2. Select your application and create a Deployment Group. CodeDeploy needs to have an IAM permission to access your targets as well as read the AWS S3 bucket containing the artifact to be deployed.



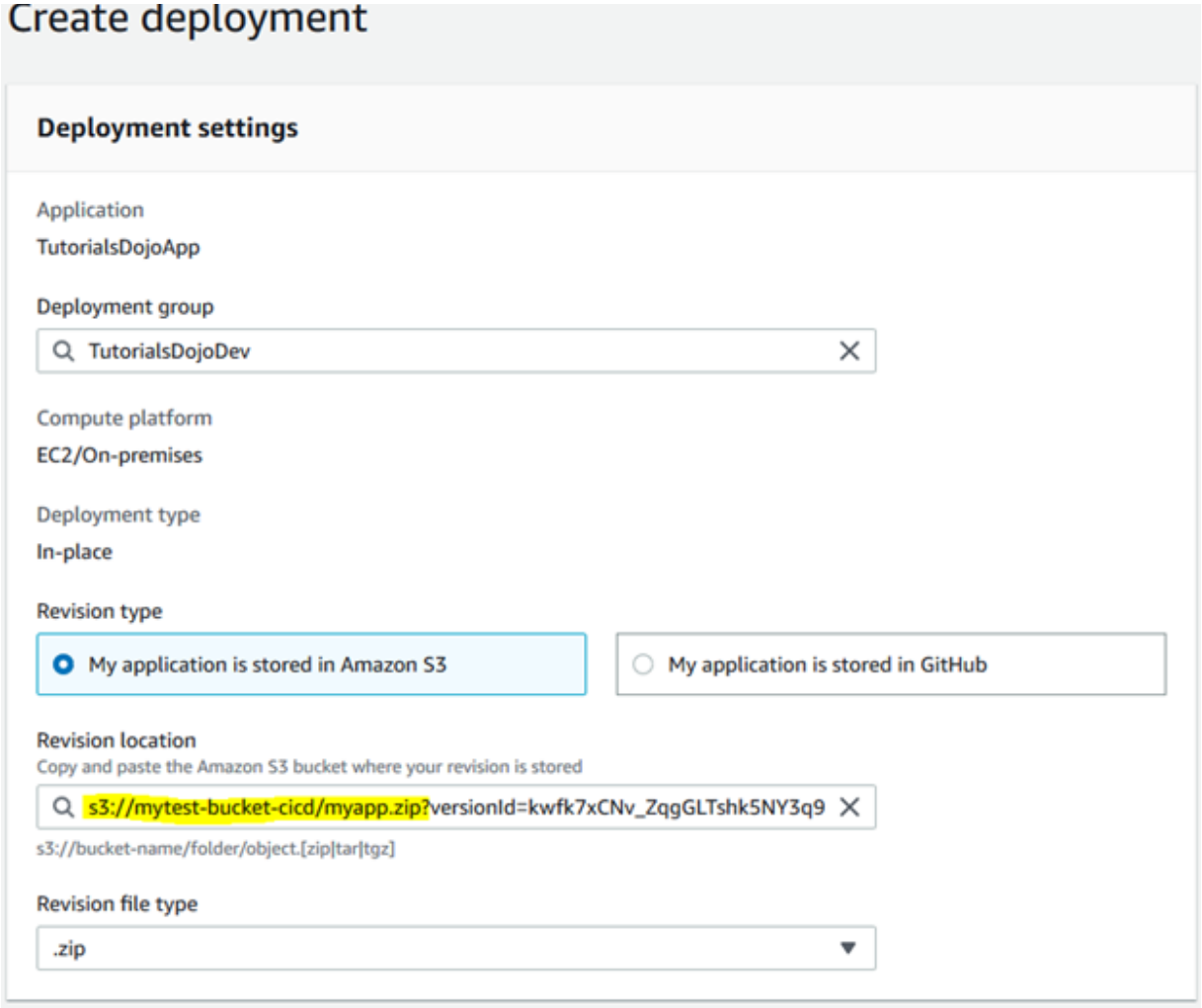
3. Select your deployment type. You can use In-place deployment if you already have the instances running. Specify the Tags of those EC2 instances, for example **Environment:Dev**. CodeDeploy will use this as an identifier for your target instances.



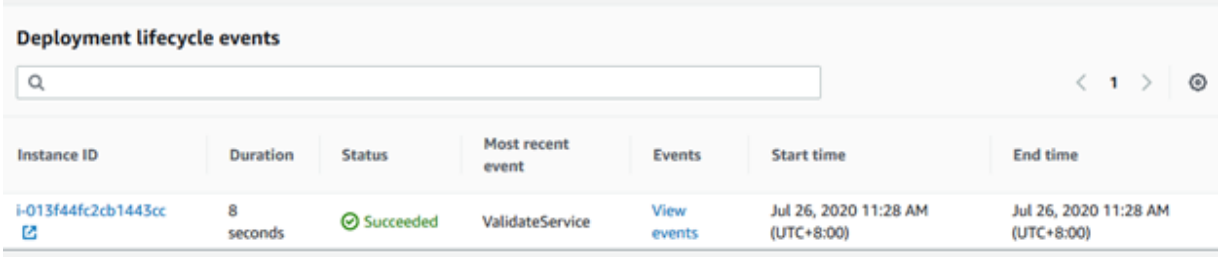
4. Select how you want the new code to be deployed, such as All-at-once, or one-at-a-time, etc. These deployment settings will be discussed on the succeeding sections.



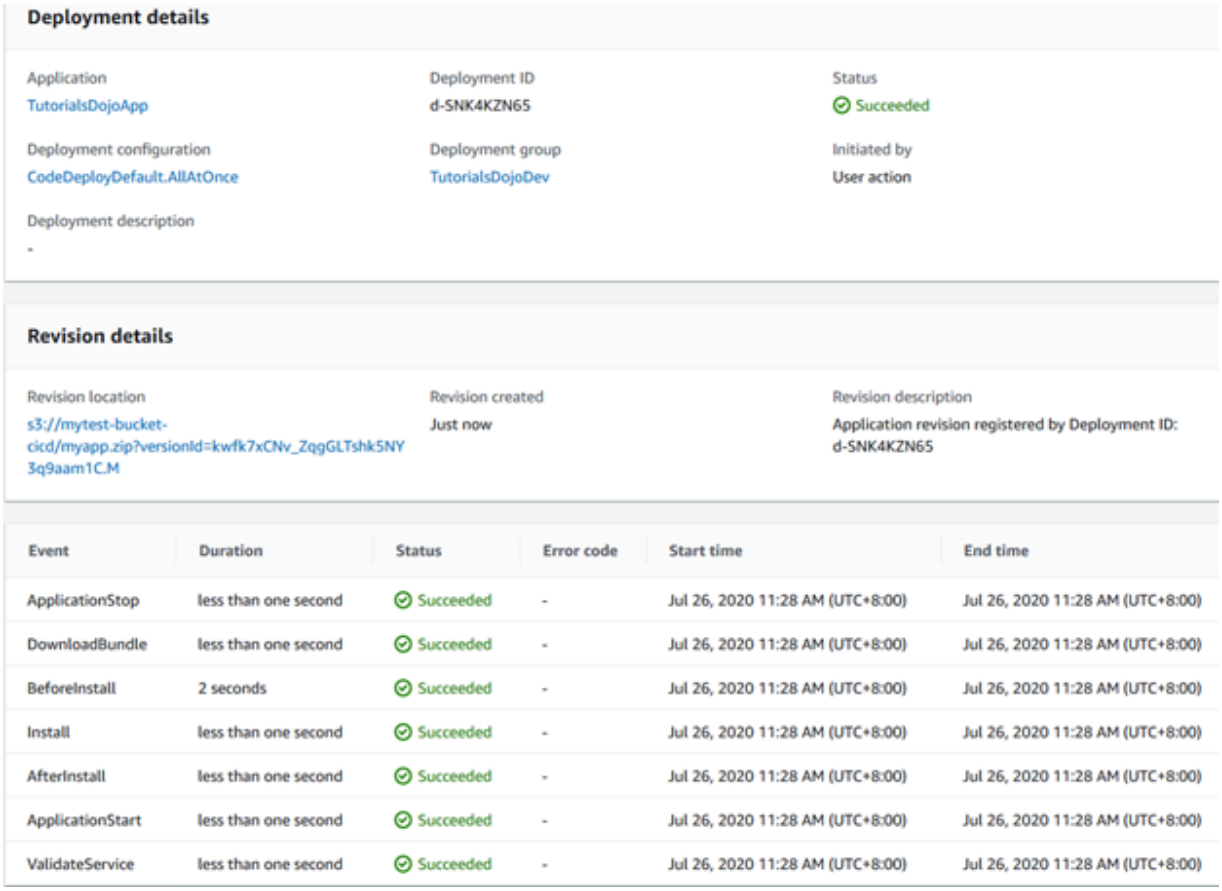
5. Now on your application Deployment group, create a Deployment. Input details for the artifact source. On the S3 bucket, you can specify the versionID of the artifact file.



6. After you click “Create deployment”, the deployment of the artifact on your EC2 instances will begin. You should see that the deployment will succeed.



7. You can click on “View events” to see the stages of the deployment process. Here you can view the deployment events as well as the status of the deployment lifecycle hooks you have defined. See the Lifecycle Event hooks section of this book for the details regarding each event.



Deployment details

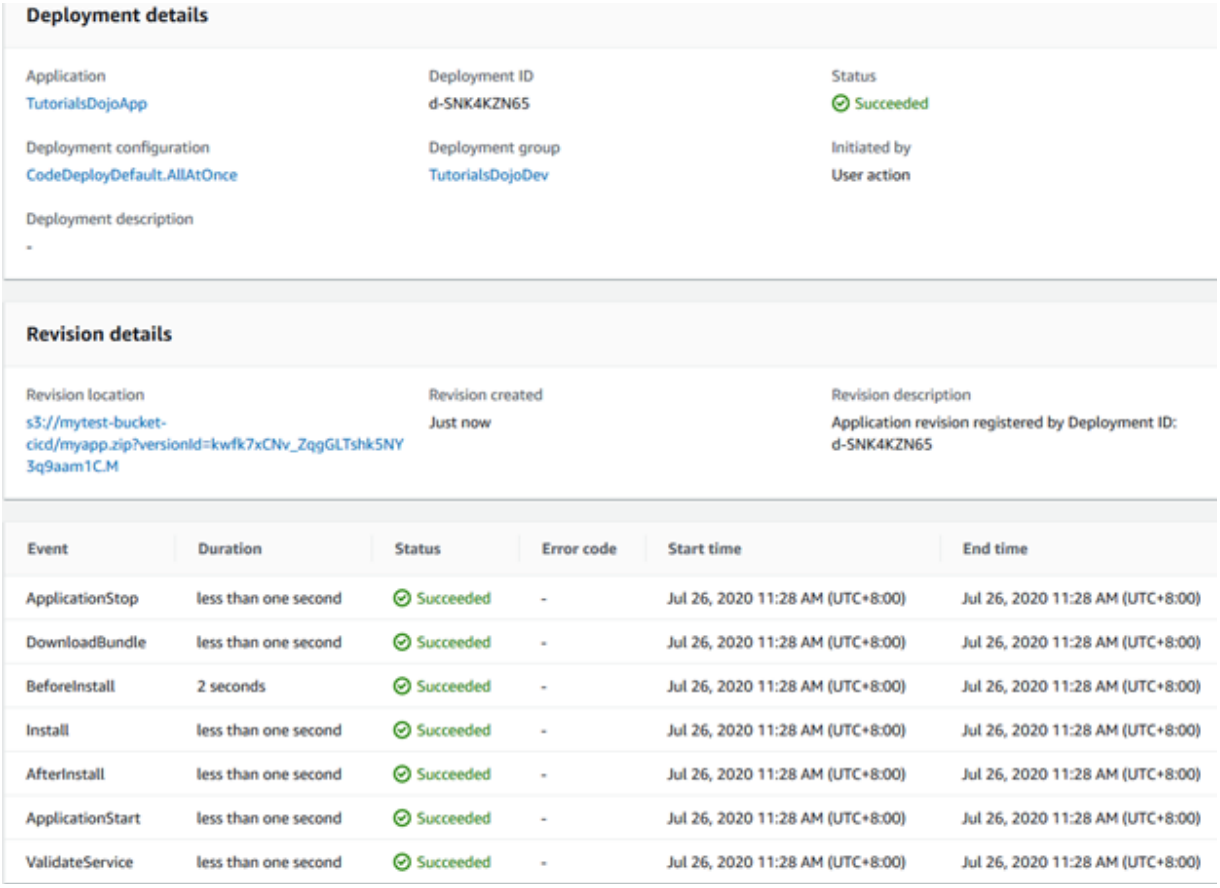
Application	Deployment ID	Status
TutorialsDojoApp	d-SNK4KZN65	Succeeded
Deployment configuration	**Deployment group**	**Initiated by**
CodeDeployDefault.AllAtOnce	TutorialsDojoDev	User action
Deployment description		
-		

Revision details

Revision location	Revision created	Revision description
s3://mytest-bucket-cicd/myapp.zip?versionId=kwfk7xCNv_ZqgGLTshk5NY3q9aam1C.M	Just now	Application revision registered by Deployment ID: d-SNK4KZN65

Event	Duration	Status	Error code	Start time	End time
ApplicationStop	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
DownloadBundle	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
BeforeInstall	2 seconds	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
Install	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
AfterInstall	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
ApplicationStart	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
ValidateService	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)

8. CodePipeline is integrated with AWS CloudWatch rule. You can create a CloudWatch Events rule that will detect CodeDeploy status changes, such as a Successful or Failed deployment. Then have it invoke a Lambda function to perform a custom action such as sending a notification on a Slack channel or setting an SNS topic to send an email to you about the status of your deployment.



Deployment details

Application	Deployment ID	Status
TutorialsDojoApp	d-SNK4KZN65	Succeeded
Deployment configuration	**Deployment group**	**Initiated by**
CodeDeployDefault.AllAtOnce	TutorialsDojoDev	User action

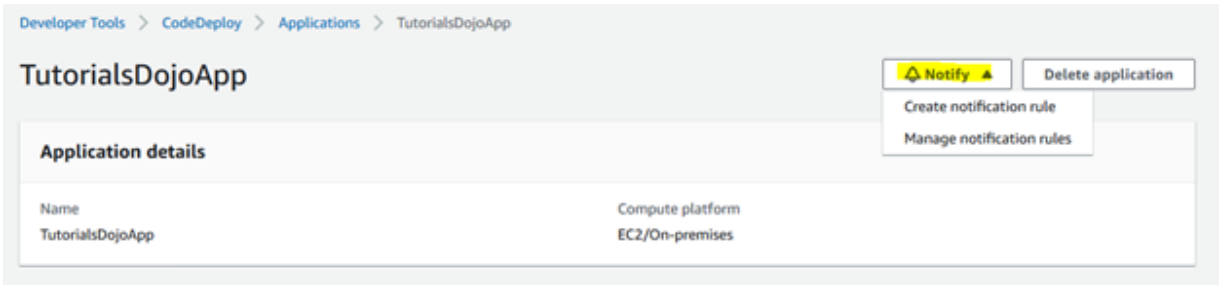
Deployment description
-

Revision details

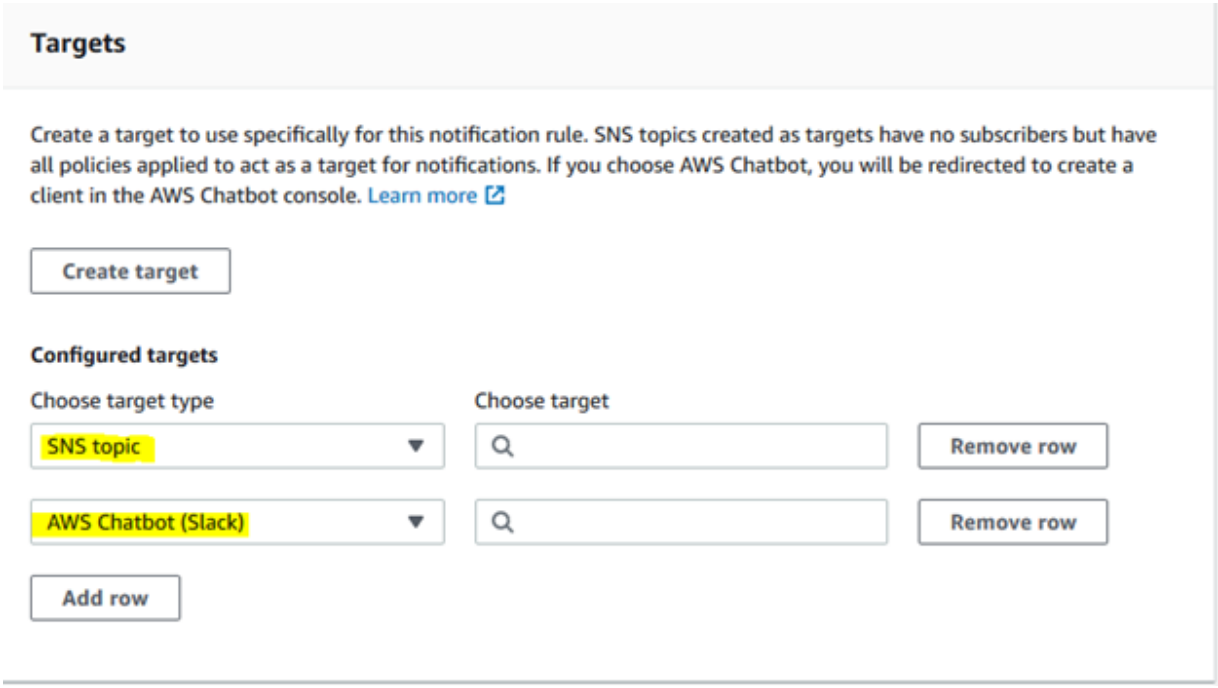
Revision location	Revision created	Revision description
s3://mytest-bucket-cicd/myapp.zip?versionId=kwfk7xCNv_ZqgGLTshk5NY3q9aam1C.M	Just now	Application revision registered by Deployment ID: d-SNK4KZN65

Event	Duration	Status	Error code	Start time	End time
ApplicationStop	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
DownloadBundle	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
BeforeInstall	2 seconds	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
Install	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
AfterInstall	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
ApplicationStart	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)
ValidateService	less than one second	Succeeded	-	Jul 26, 2020 11:28 AM (UTC+8:00)	Jul 26, 2020 11:28 AM (UTC+8:00)

9. CodeDeploy also has built-in notification triggers. Click “Notify” on your application.



10. For events happening on your deployment, you can have targets much like AWS CloudWatch Events, however, this is limited to only an SNS Topic or AWS chatbot to Slack.



DevOps Exam Notes:

AWS CodePipeline is integrated with AWS CloudWatch Events. You can create a CloudWatch Events rule that will detect CodeDeploy status changes, such as a Successful or Failed deployment. With the rule targets, you invoke a Lambda function to perform a custom action i.e. set an SNS topic to send an email to you about the status of your deployment.

CodeDeploy also has built-in notification triggers to notify you of your deployment status, however, this is limited to only an SNS Topic or AWS Chatbot to Slack.

CodeDeploy Supports ECS and Lambda Deployments

Aside from EC2 instances, AWS CodeDeploy also supports deployment for ECS instances and Lambda deployments. The general steps for deployment are still the same - create an **Application**, create a **deployment group** for your instances, and create a **deployment** for your application.

However, for ECS, the artifact is a Docker image which can be stored from AWS ECR or from DockerHub. For Lambda deployments, the artifact will come from a zip file from an S3 bucket. Be sure to have the proper filename of your Lambda handler file for successful code deployments.

Deployments for ECS also support deployment configuration such as all-at-once, one-at-a-time, half-of-the-time. These deployment configurations will be discussed on a separate section. Lambda on the other hand supports a percentage of traffic shifting such as linear or canary deployment. This is also discussed in a separate section.

Sources:
https://docs.aws.amazon.com/codedeploy/latest/userguide/monitoring-create-alarms.html
https://docs.aws.amazon.com/codedeploy/latest/userguide/monitoring.html
https://docs.aws.amazon.com/codedeploy/latest/userguide/deployments-create-console-ecs.html
https://docs.aws.amazon.com/codedeploy/latest/userguide/deployments-create-console-lambda.html

CloudFormation Template for ECS, Auto Scaling and ALB

Amazon Elastic Container Service (ECS) allows you to manage and run Docker containers on clusters of EC2 instances. You can also configure your ECS to use Fargate launch type which eliminates the need to manage EC2 instances.

With CloudFormation, you can define your ECS clusters and tasks definitions to easily deploy your containers. For high availability of your Docker containers, ECS clusters are usually configured with an auto scaling group behind an application load balancer. These resources can also be declared on your CloudFormation template.

DevOps Exam Notes:

Going on to the exam, be sure to remember the syntax needed to declare your ECS cluster, Auto Scaling group, and application load balancer. The **AWS::ECS::Service** resource creates an ECS cluster and the **AWS::ECS::TaskDefinition** resource creates a task definition for your container. The **AWS::ElasticLoadBalancingV2::LoadBalance**r resource creates an application load balancer and the **AWS::AutoScaling::AutoScalingGroup** resource creates an EC2 auto scaling group.

AWS provides an example template which you can use to deploy a web application in an Amazon ECS container with auto scaling and application load balancer. Here's a snippet of the template with the core resources:

```
{
  "AWSTemplateFormatVersion":"2010-09-09",
  "Resources":{
      "ECSCluster":{
      "Type":"AWS::ECS::Cluster"

      },

……

      "taskdefinition":{
      "Type":"AWS::ECS::TaskDefinition",

      "Properties":{

……

      "ECSALB":{
      "Type":"AWS::ElasticLoadBalancingV2::LoadBalancer",
```

```
        "Properties":{

……

        "ECSAutoScalingGroup":{
        "Type":"AWS::AutoScaling::AutoScalingGroup",
        "Properties":{
        "VPCZoneIdentifier":{
              "Ref":"SubnetId"

        },
```

Sources:
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/quickref-ecs.html
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ecs-service.html
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ecs-service-loadbalancers.html

Amazon Elastic Container Service (ECS)

- A container management service to run, stop, and manage Docker containers on a cluster.
- ECS can be used to create a consistent deployment and build experience, manage, and scale batch and **Extract-Transform-Load** (ETL) workloads, and build sophisticated application architectures on a microservices model.

Features

- You can create ECS clusters within a new or existing VPC.
- After a cluster is up and running, you can define task definitions and services that specify which Docker container images to run across your clusters.

Components

- Containers and Images
  - Your application components must be architected to run in **containers** — containing everything that your software application needs to run: code, runtime, system tools, system libraries, etc.
  - Containers are created from a read-only template called an **image**.
- Task Components
  - **Task definitions** specify various parameters for your application. It is a text file, in JSON format, that describes one or more containers, up to a maximum of ten, that form your application.
  - Task definitions are split into separate parts:
    - Task family - the name of the task, and each family can have multiple revisions.
    - IAM task role - specifies the permissions that containers in the task should have.
    - Network mode - determines how the networking is configured for your containers.
    - Container definitions - specify which image to use, how much CPU and memory the container are allocated, and many more options.
- Tasks and Scheduling
  - A **task** is the instantiation of a task definition within a cluster. After you have created a task definition for your application, you can specify the number of tasks that will run on your cluster.
    - Each task that uses the Fargate launch type has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.
  - You can upload a new version of your application task definition, and the ECS scheduler automatically starts new containers using the updated image and stop containers running the previous version.
- Clusters
  - When you run tasks using ECS, you place them in a **cluster**, which is a logical grouping of resources.
  - Clusters can contain tasks using both the Fargate and EC2 launch types.

  - When using the Fargate launch type with tasks within your cluster, ECS manages your cluster resources.
  - Enabling managed Amazon ECS cluster auto scaling allows ECS to manage the scale-in and scale-out actions of the Auto Scaling group.
- Services
  - ECS allows you to run and maintain a specified number of instances of a task definition simultaneously in a cluster.
  - In addition to maintaining the desired count of tasks in your service, you can optionally run your service behind a load balancer.
  - There are two deployment strategies in ECS:
    - **Rolling Update**
      - This involves the service scheduler replacing the current running version of the container with the latest version.

    Blue/Green Deployment with AWS CodeDeploy
      - This deployment type allows you to verify a new deployment of a service before sending production traffic to it.
      - The service must be configured to use either an Application Load Balancer or Network Load Balancer.
- Container Agent (AWS ECS Agent)
  - The **container agent** runs on each infrastructure resource within an ECS cluster.
  - It sends information about the resource's current running tasks and resource utilization to ECS, and starts and stops tasks whenever it receives a request from ECS.
  - Container agent is only supported on Amazon EC2 instances.

AWS Fargate

- You can use Fargate with ECS to run containers without having to manage servers or clusters of EC2 instances.
- You no longer have to provision, configure, or scale clusters of virtual machines to run containers.
- Fargate only supports container images hosted on Elastic Container Registry (ECR) or Docker Hub.

Task Definitions for Fargate Launch Type

- Fargate task definitions require that the network mode is set to *awsvpc*. The *awsvpc* network mode provides each task with its own elastic network interface.
- Fargate task definitions only support the *awslogs* log driver for the log configuration. This configures your Fargate tasks to send log information to Amazon CloudWatch Logs.
- Task storage is **ephemeral**. After a Fargate task stops, the storage is deleted.

Monitoring

- You can configure your container instances to send log information to CloudWatch Logs. This enables you to view different logs from your container instances in one convenient location.
- With CloudWatch Alarms, watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods.
- Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs.

AWS CodeDeploy

- A fully managed deployment service that automates software deployments to a variety of compute services such as Amazon EC2, AWS Fargate, AWS Lambda, and your on-premises servers.
- **Concepts**
  - An Application is a name that uniquely identifies the application you want to deploy. CodeDeploy uses this name, which functions as a container, to ensure the correct combination of revision, deployment configuration, and deployment group are referenced during a deployment.
  - Compute platform is the platform on which CodeDeploy deploys an application (EC2, ECS, Lambda, On-premises servers).
  - Deployment configuration is a set of deployment rules and deployment success and failure conditions used by CodeDeploy during a deployment.
  - Deployment group contains individually tagged instances, Amazon EC2 instances in Amazon EC2 Auto Scaling groups, or both.
    1. In an Amazon ECS deployment, a deployment group specifies the Amazon ECS service, load balancer, optional test listener, and two target groups. It also specifies when to reroute traffic to the replacement task set and when to terminate the original task set and ECS application after a successful deployment.
    2. In an AWS Lambda deployment, a deployment group defines a set of CodeDeploy configurations for future deployments of an AWS Lambda function.
    3. In an EC2/On-Premises deployment, a deployment group is a set of individual instances targeted for a deployment.
       - In an in-place deployment, the instances in the deployment group are updated with the latest application revision.
       - In a blue/green deployment, traffic is rerouted from one set of instances to another by deregistering the original instances from a load balancer and registering a replacement set of instances that typically has the latest application revision already installed.
  - A deployment goes through a set of predefined phases called deployment lifecycle events. A deployment lifecycle event gives you an opportunity to run code as part of the deployment.
    1. ApplicationStop
    2. DownloadBundle
    3. BeforeInstall
    4. Install
    5. AfterInstall
    6. ApplicationStart
    7. ValidateService
  - Features
    - CodeDeploy protects your application from downtime during deployments through rolling updates and deployment health tracking.
    - AWS CodeDeploy tracks and stores the recent history of your deployments.

- CodeDeploy is platform and language agnostic.
- CodeDeploy uses a file and command-based install model, which enables it to deploy any application and reuse existing setup code. The same setup code can be used to consistently deploy and test updates across your environment release stages for your servers or containers.
- CodeDeploy integrates with Amazon Auto Scaling, which allows you to scale EC2 capacity according to conditions you define such as traffic spikes. Notifications are then sent to AWS CodeDeploy to initiate an application deployment onto new instances before they are placed behind an Elastic Load Balancing load balancer.
- When using AWS CodeDeploy with on-premises servers, make sure that they can connect to AWS public endpoints.
- AWS CodeDeploy offers two types of deployments:
  - With in-place deployments, the application on each instance in the deployment group is stopped, the latest application revision is installed, and the new version of the application is started and validated. Only deployments that use the EC2/On-Premises compute platform can use in-place deployments.
  - With blue/green deployments, once the new version of your application is tested and declared ready, CodeDeploy can shift the traffic from your old version (blue) to your new version (green) according to your specifications.
- Deployment groups are used to match configurations to specific environments, such as a staging or production environments. An application can be deployed to multiple deployment groups.
- You can integrate AWS CodeDeploy with your continuous integration and deployment systems by calling the public APIs using the AWS CLI or AWS SDKs.

- Application Specification Files
  - The AppSpec file is a YAML-formatted or JSON-formatted file that is used to manage each deployment as a series of lifecycle event hooks.
  - For ECS Compute platform, the file specifies
    - The name of the ECS service and the container name and port used to direct traffic to the new task set.
    - The functions to be used as validation tests.
  - For Lambda compute platform, the file specifies
    - The AWS Lambda function version to deploy.
    - The functions to be used as validation tests.
  - For EC2/On-Premises compute platform, the file is always written in YAML and is used to
    - Map the source files in your application revision to their destinations on the instance.
    - Specify custom permissions for deployed files.
    - Specify scripts to be run on each instance at various stages of the deployment process.
- Deployments

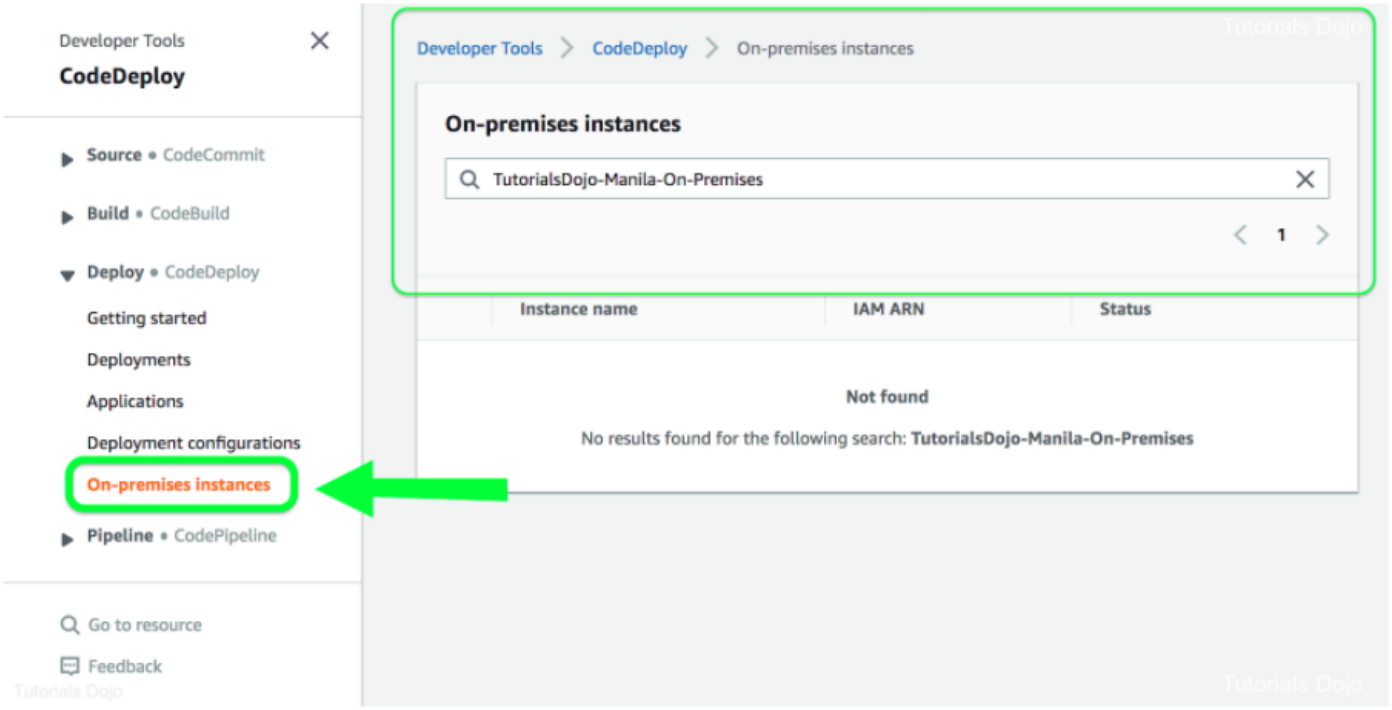
- You can use the CodeDeploy console or the create-deployment command to deploy the function revision specified in the AppSpec file to the deployment group.
- You can use the CodeDeploy console or the stop-deployment command to stop a deployment. When you attempt to stop the deployment, one of three things happens:
  - The deployment stops, and the operation returns a status of SUCCEEDED.
  - The deployment does not immediately stop, and the operation returns a status of pending. After the pending operation is complete, subsequent calls to stop the deployment return a status of SUCCEEDED.
  - The deployment cannot stop, and the operation returns an error.
- With Lambda functions and EC2 instances, CodeDeploy implements rollbacks by redeploying, as a new deployment, a previously deployed revision.
- With ECS services, CodeDeploy implements rollbacks by rerouting traffic from the replacement task set to the original task set.
- The CodeDeploy agent is a software package that, when installed and configured on an EC2/on-premises instance, makes it possible for that instance to be used in CodeDeploy deployments. The agent is not required for deployments that use the Amazon ECS or AWS Lambda.
- CodeDeploy monitors the health status of the instances in a deployment group. For the overall deployment to succeed, CodeDeploy must be able to deploy to each instance in the deployment and deployment to at least one instance must succeed.
- You can specify a minimum number of healthy instances as a number of instances or as a percentage of the total number of instances required for the deployment to be successful.
- CodeDeploy assigns two health status values to each instance:
  - Revision health - based on the application revision currently installed on the instance. Values include Current, Old and Unknown.
  - Instance health - based on whether deployments to an instance have been successful. Values include Healthy and Unhealthy.

- Blue/Green Deployments
  - EC2/On-Premises compute platform
    - You must have one or more Amazon EC2 instances with identifying Amazon EC2 tags or an Amazon EC2 Auto Scaling group.
    - Each Amazon EC2 instance must have the correct IAM instance profile attached.
    - The CodeDeploy agent must be installed and running on each instance.
    - During replacement, you can either
      - use the Amazon EC2 Auto Scaling group you specify as a template for the replacement environment; or
      - specify the instances to be counted as your replacement using EC2 instance tags, EC2 Auto Scaling group names, or both.

  - AWS Lambda platform

        - You must choose one of the following deployment configuration types to specify how traffic is shifted from the original Lambda function version to the new version:
            - Canary: Traffic is shifted in two increments. You can choose from predefined canary options that specify the percentage of traffic shifted to your updated Lambda function version in the first increment and the interval, in minutes, before the remaining traffic is shifted in the second increment.
            - Linear: Traffic is shifted in equal increments with an equal number of minutes between each increment. You can choose from predefined linear options that specify the percentage of traffic shifted in each increment and the number of minutes between each increment.
            - All-at-once: All traffic is shifted from the original Lambda function to the updated Lambda function version all at once.

        - With Amazon ECS, production traffic shifts from your ECS service's original task set to a replacement task set all at once.
    - Advantages of using Blue/Green Deployments vs In-Place Deployments
        - An application can be installed and tested in the new replacement environment and deployed to production simply by rerouting traffic.
        - If you're using the EC2/On-Premises compute platform, switching back to the most recent version of an application is faster and more reliable. Traffic can just be routed back to the original instances as long as they have not been terminated. With an in-place deployment, versions must be rolled back by redeploying the previous version of the application.
        - If you're using the EC2/On-Premises compute platform, new instances are provisioned and contain the most up-to-date server configurations.
        - If you're using the AWS Lambda compute platform, you control how traffic is shifted from your original AWS Lambda function version to your new AWS Lambda function version.
- With AWS CodeDeploy, you can also deploy your applications to your on-premises data centers. Your on-premises instances will have a prefix of “mi-xxxxxxxxx”.


Developer Tools
CodeDeploy
Source • CodeCommit
Build • CodeBuild
Deploy • CodeDeploy
Getting started
Deployments
Applications
Deployment configurations
On-premises instances
Pipeline • CodePipeline
Go to resource
Feedback
Developer Tools > CodeDeploy > On-premises instances
On-premises instances
TutorialsDojo-Manila-On-Premises
1
Instance name
IAM ARN
Status
Not found
No results found for the following search: TutorialsDojo-Manila-On-Premises

AWS CodePipeline

- A fully managed **continuous delivery service** that helps you automate your release pipelines for application and infrastructure updates.
- You can easily integrate AWS CodePipeline with third-party services such as GitHub or with your own custom plugin.
- **Concepts**
    - A **pipeline** defines your release process workflow, and describes how a new code change progresses through your release process.
    - A pipeline comprises a series of **stages** (e.g., build, test, and deploy), which act as logical divisions in your workflow. Each stage is made up of a sequence of actions, which are tasks such as building code or deploying to test environments.
        - Pipelines must have **at least two stages**. The first stage of a pipeline is required to be a source stage, and the pipeline is required to additionally have at least one other stage that is a build or deployment stage.
    - Define your pipeline structure through a **declarative JSON** document that specifies your release workflow and its stages and actions. These documents enable you to update existing pipelines as well as provide starting templates for creating new pipelines.
    - A **revision** is a change made to the source location defined for your pipeline. It can include source code, build output, configuration, or data. A pipeline can have multiple revisions flowing through it at the same time.
    - A **stage** is a group of one or more actions. A pipeline can have two or more stages.
    - An **action** is a task performed on a revision. Pipeline actions occur in a specified order, in serial or in parallel, as determined in the configuration of the stage.
        - You can add actions to your pipeline that are in an AWS Region different from your pipeline.
        - There are six types of actions
            - Source
            - Build
            - Test
            - Deploy
            - Approval
            - Invoke
    - When an action runs, it acts upon a file or set of files called **artifacts**. These artifacts can be worked upon by later actions in the pipeline. You have an artifact store which is an S3 bucket in the same AWS Region as the pipeline to store items for all pipelines in that Region associated with your account.
    - The stages in a pipeline are connected by **transitions.** Transitions can be disabled or enabled between stages. If all transitions are enabled, the pipeline runs continuously.

    - An **approval action** prevents a pipeline from transitioning to the next action until permission is granted. This is useful when you are performing code reviews before code is deployed to the next stage.
- **Features**
    - AWS CodePipeline provides you with a graphical user interface to create, configure, and manage your pipeline and its various stages and actions.
    - A pipeline starts automatically (default) when a change is made in the source location, or when you manually start the pipeline. You can also set up a rule in CloudWatch to automatically start a pipeline when events you specify occur.
    - You can model your build, test, and deployment actions to run **in parallel** in order to increase your workflow speeds.
    - AWS CodePipeline can pull source code for your pipeline directly from AWS CodeCommit, GitHub, Amazon ECR, or Amazon S3.
    - It can run builds and unit tests in AWS CodeBuild.
    - It can deploy your changes using AWS CodeDeploy, AWS Elastic Beanstalk, Amazon ECS, AWS Fargate, Amazon S3, AWS Service Catalog, AWS CloudFormation, and/or AWS OpsWorks Stacks.
    - You can use the CodePipeline Jenkins plugin to easily register your existing build servers as a custom action.
    - When you use the console to create or edit a pipeline that has a GitHub source, CodePipeline creates a **webhook**. A webhook is an HTTP notification that detects events in another tool, such as a GitHub repository, and connects those external events to a pipeline. CodePipeline deletes your webhook when you delete your pipeline.
- As a best practice, when you use a Jenkins build provider for your pipeline's build or test action, install Jenkins on an Amazon EC2 instance and configure a separate EC2 instance profile. Make sure the instance profile grants Jenkins only the AWS permissions required to perform tasks for your project, such as retrieving files from Amazon S3.