AWS Certified DevOps Engineer Professional Master Cheat Sheet

SDLC

Deployment Types:

Single Target

- New app version is installed on target server
- Outage occurs during installation
- No secondary servers, testing is limited
- Rollback requires removing new version and installing previous version

All-at-once

- Deployment happens in one step
- Destination is multiple targets
- Requires orchestration tooling
- Shares negatives of Single Target
    - No ability to test
    - Outage
    - Less than ideal rollback

Minimum In-service

- Happens in multiple stages
- Deployment happens to as many targets as possible while maintaining minimum in-servive targets
- Allows automated testing
    - Deployment targets are assessed and tested prior to continuing

- No downtime

Rolling Deployment

- Deployment happens in multiple stages, number of targets per stage is user-defined
- Moving parts, orchestration, health checks required
- Overall applicable health is not necessarily maintained
- Can be least effecient deployment type based on time-taken
- Allows automated testing
    - Deployment targets are assessed and tested prior to continuing
- Generally no downtime, assuming number of targets will not impact application
- Can be paused, allowing limited multi-version testing

Blue Green Deployment

- Requires advanced orchestration tooling
- Costs more: maintain two environments at once
- Deployment is rapid, entire environment is deployed at once
- Cutover and migration is clean and controlled (DNS change)
- Rollback is equally clean (DNS regression)
- Health and performance of entire green environment can be tested prior to cutover
- Using advanced template systems (CloudFormation), entire process can be automated

Canary Deployment

- Similar to Blue Green, but blue stays incrementally active
- Can be done with Route53 - weighted Round Robin
- Easily allows for a/b testing

CodeCommit

Version control service to privately store and manage assets (such as documents, source code, and binary files) in the cloud

- Fully Managed –AWS CodeCommit eliminates the need to host, maintain, backup, and scale your own source control servers.
- Secure –AWS CodeCommit automatically encrypts your files in transit and at rest. AWS CodeCommit is integrated with AWS Identity and Access Management (IAM), allowing you to assign user-specific permissions to your repositories.
- Highly Available – AWS CodeCommit is built on highly scalable, redundant, and durable AWS services such as Amazon S3 and Amazon DynamoDB.
- Scalable - AWS CodeCommit allows you store any number of files and there are no repository size limits.
- Faster Development Lifecycle - AWS CodeCommit keeps your repositories close to your build, staging, and production environments in the AWS cloud. This allows you to increase the speed and frequency of your development lifecycle.

Creating Web Hooks with CodeCommit

In the Amazon Simple Notification Service (SNS) console, you can create a SNS topic with an HTTP endpoint and the desired URL for the webhook. From the AWS CodeCommit console, you can then configure that SNS topic to a repository event using triggers.

CodeBuild

Fully managed continuous integration service in the cloud. CodeBuild compiles source code, runs tests, and produces packages that are ready to deploy.

- Eliminates the need to provision, manage, and scale your own build servers
- Automatically scales up and down and processes multiple builds concurrently, so builds don't have to wait in a queue.
- Use prepackaged build environments or custom build environments to use your own build tools
- Pay by the minute

How it Works

As input, you must provide CodeBuild with a build project. A build project defines how CodeBuild runs a build. It includes information such as where to get the source code, the build environment to use, the build commands to run, and where to store the build output. A build environment represents a combination of operating system, programming language runtime, and tools that CodeBuild uses to run a build.

Steps:

1. CodeBuild uses the build project to create the build environment.
2. CodeBuild downloads the source code into the build environment and then uses the build specification (build spec), as defined in the build project or included directly in the source code. A build spec is a collection of build commands and related settings, in YAML format, that CodeBuild uses to run a build.
3. If there is any build output, the build environment uploads its output to an Amazon S3 bucket. The build environment can also perform tasks that you specify in the build spec (for example, sending build notifications to an Amazon SNS topic).
4. While the build is running, the build environment sends information to CodeBuild and Amazon CloudWatch Logs.
5. While the build is running, you can use the CodeBuild console, AWS CLI, or AWS SDKs, to get summarized build information from CodeBuild and detailed build information from Amazon CloudWatch Logs. If you use AWS CodePipeline to run builds, you can get limited build information from CodePipeline.

Sources

CodeBuild can connect to AWS CodeCommit, S3, GitHub, and GitHub Enterprise and Bitbucket to pull source code

Programming Frameworks

Java, Ruby, Python, Go, Node.js, Android, .NET Core, PHP, and Docker. Customize an environment by creating a Docker image and uploading it to the Amazon EC2 Container Registry or the Docker Hub registry

Jenkins Integration

CodeBuild Plugin for Jenkins can be used to integrate CodeBuild into Jenkins jobs

Security

- Specify a key stored in the AWS Key Management Service (AWS KMS) to encrypt your artifacts

- Runs build in fresh environments isolated from other users and discards each build environment upon completion. CodeBuild provides security and separation at the infrastructure and execution levels.

Buildspec.yml

A build spec is a collection of build commands and related settings, in YAML format, that CodeBuild uses to run a build. You can include a build spec as part of the source code or you can define a build spec when you create a build project.

By default, the build spec file must be named buildspec.yml and placed in the root of your source directory. You can override the default build spec file name and location

The build spec has the following syntax:

```
version: 0.2

run-as: Linux-user-name

env:
  variables:
    key: "value"
    key: "value"
  parameter-store:
    key: "value"
    key: "value"

phases:
  install:
    run-as: Linux-user-name
    commands:
      - command
      - command
    finally:
      - command
      - command
  pre_build:
    run-as: Linux-user-name
    commands:
      - command
      - command
    finally:
      - command
      - command
  build:
    run-as: Linux-user-name
    commands:
      - command
      - command
    finally:
      - command
      - command
  post_build:
    run-as: Linux-user-name
```

```
    commands:
      - command
      - command
    finally:
      - command
      - command
artifacts:
  files:
    - location
    - location
  name: artifact-name
  discard-paths: yes
  base-directory: location
  secondary-artifacts:
    artifactIdentifier:
      files:
        - location
        - location
      name: secondary-artifact-name
      discard-paths: yes
      base-directory: location
    artifactIdentifier:
      files:
        - location
        - location
      discard-paths: yes
      base-directory: location
cache:
  paths:
    - path
    - path
```

Build Environment Compute Types

Compute type	computeType	value Memory	vCPUs	Disk space	Operating system
build.general1.small	BUILD_GENERAL1_SMALL	3 GB	2	64 GB	Linux
build.general1.medium	BUILD_GENERAL1_MEDIUM	7 GB	4	128 GB	Linux, Windows
build.general1.large	BUILD_GENERAL1_LARGE	15 GB	8	128 GB	Linux, Windows

Docker Images Provided by CodeBuild

Platform	Programming language or framework	Image identifier	Definition
Ubuntu 18.04	(Standard image)	aws/codebuild/standard:1.0	ubuntu/standard/1.0
Windows Server Core 2016	(Base Image)	aws/codebuild	windows-base:1.0
Standard image of the Ubuntu 18.04 platform contains the following programming languages: • Ruby 2.x • Python 3.x • PHP 7.x • Node 10.x • Java 8 • Golang 1.x • .NET Core 2.x • Docker 18.x • Android 28.x			

CodeDeploy

Service that automates code deployments to any instance, including Amazon EC2 instances and instances running on-premises.

Supported Platforms/Deployment Types:

- EC2/On-Premises: In-Place or Blue/Green Deployments
    - Describes instances of physical servers that can be Amazon EC2 cloud instances, on-premises servers, or both. Applications created using the EC2/On-Premises compute platform can be composed of executable files, configuration files, images, and more.
    - Deployments that use the EC2/On-Premises compute platform manage the way in which traffic is directed to instances by using an in-place or blue/green deployment type.
- AWS Lambda: Canary, Linear, All-At-Once Deployments
    - Applications created using the AWS Lambda compute platform can manage the way in which traffic is directed to the updated Lambda function versions during a deployment by choosing a canary, linear, or all-at-once configuration.
- Amazon ECS: Blue/Green Deployment
    - Used to deploy an Amazon ECS containerized application as a task set.
    - CodeDeploy performs a blue/green deployment by installing an updated version of the containerized application as a new replacement task set. CodeDeploy reroutes production traffic from the original application, or task set, to the replacement task set. The original task set is terminated after a successful deployment.

CodeDeploy Components per Compute Platform

CodeDeploy Component	EC2/On-Premises	AWS Lambda	Amazon ECS
Deployment group	Deploys a revision to a set of instances.	Deploys a new version of a serverless Lambda function on a high-availability compute infrastructure.	Specifies the Amazon ECS service with the containerized application to deploy as a task set, a production and optional test listener used to serve traffic to the deployed application, when to reroute traffic and

CodeDeploy Component	EC2/On-Premises	AWS Lambda	Amazon ECS
			terminate the deployed application's original task set, and optional trigger, alarm, and rollback settings.
Deployment	Deploys a new revision that consists of an application and AppSpec file. The AppSpec specifies how to deploy the application to the instances in a deployment group.	Shifts production traffic from one version of a Lambda function to a new version of the same function. The AppSpec file specifies which Lambda function version to deploy.	Deploys an updated version of an Amazon ECS containerized application as a new, replacement task set. CodeDeploy reroutes production traffic from the task set with the original version to the new replacement task set with the updated version. When the deployment completes, the original task set is terminated.
Deployment configuration	Settings that determine the deployment speed and the minimum number of instances that must be healthy at any point during a deployment.	Settings that determine how traffic is shifted to the updated Lambda function versions.	Traffic is always shifted all at once. Custom deployment configuration settings cannot be specified for an Amazon ECS deployment.
Revision	A combination of an AppSpec file and application	An AppSpec file that specifies which Lambda function to	An AppSpec file that specifies:

CodeDeploy Component	EC2/On-Premises	AWS Lambda	Amazon ECS
	files, such as executables, configuration files, and so on.	deploy and Lambda fuctions that can run validation tests during deployment lifecycle event hooks.	• The Amazon ECS task definition for the Amazon ECS service with the containerized application to deploy. • The container where your updated application is deployed. • A port for the container where production traffic is rerouted. • Optional network configuration settings and Lambda fuctions that can run validation tests during deployment lifecycle event hooks.
Application	A collection of deployment groups and revisions. An EC2/On-Premises application uses the EC2/On-Premises	A collection of deployment groups and revisions. An application used for an AWS Lambda deployment uses the Amazon ECS	A collection of deployment groups and revisions. An application used for an Amazon ECS deployment uses the Amazon ECS compute platform.

CodeDeploy Component	EC2/On-Premises	AWS Lambda	Amazon ECS
	compute platform.	compute platform.	

App Spec File

The application specification file (AppSpec file) is a YAML-formatted or JSON-formatted file used by CodeDeploy to manage a deployment.

Note: the name of the AppSpec file for an EC2/On-Premises deployment must be appspec.yml. The name of the AppSpec file for an Amazon ECS or AWS Lambda deployment must be appspec.yaml.

AppSpec Files on ECS

Determine:

- Amazon ECS task definition file. This is specified with its ARN in the TaskDefinition instruction in the AppSpec file.
- The container and port in replacement task set where your Application Load Balancer or Network Load Balancer reroutes traffic during a deployment. This is specified with the LoadBalancerInfo instruction in the AppSpec file.
- Optional information about your Amazon ECS service, such the platform version on which it runs, its subnets, and its security groups.
- Optional Lambda functions to run during hooks that correspond with lifecycle events during an Amazon ECS deployment. For more information, see AppSpec 'hooks' Section for an Amazon ECS Deployment.

Example appspec.yaml (ECS)

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "arn:aws:ecs:us-east-1:111222333444:task-definition/my-
task-definition-family-name:1"
        LoadBalancerInfo:
          ContainerName: "SampleApplicationName"
          ContainerPort: 80
# Optional properties
        PlatformVersion: "LATEST"
        NetworkConfiguration:
          AwsvpcConfiguration:
```

```
            Subnets: ["subnet-1234abcd","subnet-5678abcd"]
            SecurityGroups: ["sg-12345678"]
            AssignPublicIp: "ENABLED"
Hooks:
  - BeforeInstall: "LambdaFunctionToValidateBeforeInstall"
  - AfterInstall: "LambdaFunctionToValidateAfterTraffic"
  - AfterAllowTestTraffic: "LambdaFunctionToValidateAfterTestTrafficStarts"
  - BeforeAllowTraffic: "LambdaFunctionToValidateBeforeAllowingProductionTraffic"
  - AfterAllowTraffic: "LambdaFunctionToValidateAfterAllowingProductionTraffic"
```

AppSpec Files on Lambda

Determine:

- Which Lambda function version to deploy.
- Which Lambda functions to use as validation tests.

An AppSpec file can be YAML-formatted or JSON-formatted. You can also enter the contents of an AppSpec file directly into CodeDeploy console when you create a deployment.

Example appspec.yaml (Lambda)

```
version: 0.0
Resources:
  - myLambdaFunction:
      Type: AWS::Lambda::Function
      Properties:
        Name: "myLambdaFunction"
        Alias: "myLambdaFunctionAlias"
        CurrentVersion: "1"
        TargetVersion: "2"
Hooks:
  - BeforeAllowTraffic: "LambdaFunctionToValidateBeforeTrafficShift"
  - AfterAllowTraffic: "LambdaFunctionToValidateAfterTrafficShift"
```

AppSpec Files on EC2/On-Premises

Determine:

- What it should install onto your instances from your application revision in Amazon S3 or GitHub.
- Which lifecycle event hooks to run in response to deployment lifecycle events.

Note: An AppSpec file must be a YAML-formatted file named appspec.yml and it must be placed in the root of the directory structure of an application's source code. Otherwise, deployments fail.

Steps:

- Complete AppSpec file, bundle it, along with the content to deploy, into an archive file (zip, tar, or compressed tar)
- After you have a bundled archive file (known in CodeDeploy as a revision), you upload it to an Amazon S3 bucket or Git repository.
- Use CodeDeploy to deploy the revision.
- The appspec.yml for an EC2/On-Premises compute platform deployment is saved in the root directory of your revision.

Example appspec.yml (EC2/On-Premises)

```
version: 0.0
os: linux
files:
  - source: Config/config.txt
    destination: /webapps/Config
  - source: source
    destination: /webapps/myApp
hooks:
  BeforeInstall:
    - location: Scripts/UnzipResourceBundle.sh
    - location: Scripts/UnzipDataBundle.sh
  AfterInstall:
    - location: Scripts/RunResourceTests.sh
      timeout: 180
  ApplicationStart:
    - location: Scripts/RunFunctionalTests.sh
      timeout: 3600
  ValidateService:
    - location: Scripts/MonitorService.sh
      timeout: 3600
      runas: codedeployuser
```

Configuration Management and Infrastructure as Code

Cloudformation

Key Terms

1. Stack - manage related resources as a single unit called a stack. Create, update, and delete a collection of resources by creating, updating, and deleting stacks. All the resources in a stack are defined by the stack's AWS CloudFormation template.

2. Template - JSON or YAML formatted text file. AWS CloudFormation uses these templates as blueprints for building your AWS resources.
3. Stack Policy - IAM style policy statement which governs what can be changed and who can change it.

Anatomy of a Template

- Format Version (optional)
  - The AWS CloudFormation template version that the template conforms to. The template format version is not the same as the API or WSDL version.
- Description (optional)
  - A text string that describes the template. This section must always follow the template format version section.
- Metadata (optional)
  - Objects that provide additional information about the template.
- **Parameters** (optional)
  - Values to pass to your template at runtime (when you create or update a stack).
- **Mappings** (optional)
  - A mapping of keys and associated values that you can use to specify conditional parameter values, similar to a lookup table. You can match a key to a corresponding value by using the Fn::FindInMap intrinsic function in the Resources and Outputs sections.
- Conditions (optional)
  - Conditions that control whether certain resources are created or whether certain resource properties are assigned a value during stack creation or update. For example, you could conditionally create a resource that depends on whether the stack is for a production or test environment.
- Transform (optional)
  - For serverless applications (also referred to as Lambda-based applications), specifies the version of the AWS Serverless Application Model (AWS SAM) to use. When you specify a transform, you can use AWS SAM syntax to declare resources in your template. The model defines the syntax that you can use and how it is processed.
  - You can also use AWS::Include transforms to work with template snippets that are stored separately from the main AWS CloudFormation template. You can store your snippet files in an Amazon S3 bucket and then reuse the functions across multiple templates.
- **Resources** (required)
  - Specifies the stack resources and their properties, such as an Amazon Elastic Compute Cloud instance or an Amazon Simple Storage Service bucket. You can refer to resources in the Resources and Outputs sections of the template.
- **Outputs** (optional)

- Describes the values that are returned whenever you view your stack's properties. For example, you can declare an output for an S3 bucket name and then call the aws cloudformation describe-stacks AWS CLI command to view the name.

Intrinsic Functions

Fn::Base64

- Returns the Base64 representation of the input string. This function is typically used to pass encoded data to Amazon EC2 instances by way of the UserData property.

JSON

```
{ "Fn::Base64" : valueToEncode }
```

YAML

```
!Base64 valueToEncode
```

Fn::Cidr

- Returns an array of CIDR address blocks. The number of CIDR blocks returned is dependent on the count parameter.

JSON

```
{ "Fn::Cidr" : [ipBlock, count, cidrBits]}
```

YAML

```
!Cidr [ ipBlock, count, cidrBits ]
```

- **ipBlock**: The user-specified CIDR address block to be split into smaller CIDR blocks.
- **count**: The number of CIDRs to generate. Valid range is between 1 and 256.
- **cidrBits**: The number of subnet bits for the CIDR. For example, specifying a value "8" for this parameter will create a CIDR with a mask of "/24".

Fn::FindInMap

- Returns the value corresponding to keys in a two-level map that is declared in the Mappings section JSON

```
{ "Fn::FindInMap" : [ "MapName", "TopLevelKey", "SecondLevelKey"] }
```

YAML

```
!FindInMap [ MapName, TopLevelKey, SecondLevelKey ]
```

- **MapName**:
- The logical name of a mapping declared in the Mappings section that contains the keys and values.
- **TopLevelKey**: The top-level key name. Its value is a list of key-value pairs.
- **SecondLevelKey**: The second-level key name, which is set to one of the keys from the list assigned to TopLevelKey.

Fn::GetAtt

- Returns the value of an attribute from a resource in the template. JSON

```
{ "Fn::GetAtt" : [ "logicalNameOfResource", "attributeName" ] }
```

YAML

```
!GetAtt logicalNameOfResource.attributeName
```

Fn::GetAZs

- Returns an array that lists Availability Zones for a specified region. Prevents hard-coding a full list of Availability Zones for a specified region. JSON

```
{ "Fn::GetAZs" : "region" }
```

YAML

```
!GetAZs region
```

Fn::ImportValue

- Returns the value of an output exported by another stack. You typically use this function to create cross-stack references. Note:
- For each AWS account, Export names must be unique within a region.
- You can't create cross-stack references across regions.
- For outputs, the value of the Name property of an Export can't use Ref or GetAtt functions that depend on a resource.
- Similarly, the ImportValue function can't include Ref or GetAtt functions that depend on a resource.
- You can't delete a stack if another stack references one of its outputs.
- You can't modify or remove an output value that is referenced by another stack. JSON

```
{ "Fn::ImportValue" : sharedValueToImport }
```

YAML

```
!ImportValue sharedValueToImport
```

- Note: You can't use the short form of !ImportValue when it contains a !Sub.

Example:

```
Fn::ImportValue:
  !Sub "${NetworkStack}-SubnetID"
```

Fn::Join

- Appends a set of values into a single value, separated by the specified delimiter. If a delimiter is the empty string, the set of values are concatenated with no delimiter.
  JSON

```
{ "Fn::Join" : [ "delimiter", [ comma-delimited list of values ] ] }
"Fn::Join" : [ ":", [ "a", "b", "c" ] ]
```

YAML

```
!Join [ delimiter, [ comma-delimited list of values ] ]
!Join [ ":", [ a, b, c ] ]
!Join
  - ''
  - - 'arn:'
    - !Ref Partition
    - ':s3:::elasticbeanstalk-*-'
    - !Ref 'AWS::AccountId'
```

Fn::Select

- Returns a single object from a list of objects by index.

Note:

- Fn::Select does not check for null values or if the index is out of bounds of the array. Both conditions will result in a stack error, so you should be certain that the index you choose is valid, and that the list contains non-null values.

JSON

```
{ "Fn::Select" : [ index, listOfObjects ] }
{ "Fn::Select" : [ "1", [ "apples", "grapes", "oranges", "mangoes" ] ] }
```

YAML

```
!Select [ index, listOfObjects ]
!Select [ "1", [ "apples", "grapes", "oranges", "mangoes" ] ]
```

Fn::Split

- Split a string into a list of string values so that you can select an element from the resulting string list. Specify the location of splits with a delimiter, such as , (a comma). After you split a string, use the Fn::Select function to pick a specific element. JSON

```
{ "Fn::Split" : [ "|" , "a|b|c" ] }
{ "Fn::Select" : [ "2", { "Fn::Split": [",", {"Fn::ImportValue":
"AccountSubnetIDs"}]}] }
```

YAML

```
!Split [ "|" , "a|b|c" ]
!Select [2, !Split [",", !ImportValue AccountSubnetIDs]]
```

Fn::Sub

- Substitutes variables in an input string with values that you specify JSON

```
{ "Fn::Sub" : [ String, { Var1Name: Var1Value, Var2Name: Var2Value } ] }
{ "Fn::Sub": [ "www.${Domain}", { "Domain": {"Ref" : "RootDomainName" }} ]}
```

YAML

```
!Sub
  - String
  - { Var1Name: Var1Value, Var2Name: Var2Value }
Name: !Sub
  - www.${Domain}
  - { Domain: !Ref RootDomainName }
```

Fn::Transform

- Specifies a macro to perform custom processing on part of a stack template. Macros enable you to perform custom processing on templates, from simple actions like find-and-replace operations to extensive transformations of entire templates.
- You can also use Fn::Transform to call the AWS::Include Transform transform, which is a macro hosted by AWS CloudFormation. JSON

```
{ "Fn::Transform" : { "Name" : macro name, "Parameters" : {key : value, ... } } }
```

YAML

```
!Transform { "Name" : macro name, "Parameters" : {key : value, ... } }
```

Ref

- Returns the value of the specified parameter or resource. JSON

```
{ "Ref" : "logicalName" }
```

YAML

```
!Ref logicalName
```

Condition Functions

- Use intrinsic functions to conditionally create stack resources. Conditions are evaluated based on input parameters that you declare when you create or update a stack.
- Define all conditions in the Conditions section of a template except for Fn::If

Condition Intrinsic Functions

- Fn::And
    - `Fn::And: [condition]`
    - `!And [condition]`
    - Examples:

```
"MyAndCondition": {
    "Fn::And": [
        {"Fn::Equals": ["sg-mysggroup", {"Ref": "ASecurityGroup"}]},
        {"Condition": "SomeOtherCondition"}
    ]
  }
MyAndCondition: !And
- !Equals ["sg-mysggroup", !Ref ASecurityGroup]
- !Condition SomeOtherCondition
```

- Fn::Equals
    - `"Fn::Equals" : ["value_1", "value_2"]`
    - `!Equals [value_1, value_2]`
    - Examples:

```
"UseProdCondition" : {
  "Fn::Equals": [
     {"Ref": "EnvironmentType"},
     "prod"
  ]
}
UseProdCondition:
  !Equals [!Ref EnvironmentType, prod]
```

- Fn::If
    - `"Fn::If": [condition_name, value_if_true, value_if_false]`
    - `!If [condition_name, value_if_true, value_if_false]`
    - Examples:

```
"SecurityGroups" : [{
  "Fn::If" : [
    "CreateNewSecurityGroup",
    {"Ref" : "NewSecurityGroup"},
    {"Ref" : "ExistingSecurityGroup"}
  ]
```

```
}]
SecurityGroups:
  - !If [CreateNewSecurityGroup, !Ref NewSecurityGroup, !Ref
ExistingSecurityGroup]
```

- Fn::Not
  - "Fn::Not": [{condition}]
  - !Not [condition]
  - Examples:

```
"MyNotCondition" : {
  "Fn::Not" : [{
     "Fn::Equals" : [
       {"Ref" : "EnvironmentType"},
       "prod"
     ]
  }]
}
MyNotCondition:
  !Not [!Equals [!Ref EnvironmentType, prod]]
```

- Fn::Or
  - "Fn::Or": [{condition}, {...}]
  - !Or [condition, ...]
  - Examples:

```
"MyOrCondition" : {
  "Fn::Or" : [
     {"Fn::Equals" : ["sg-mysggroup", {"Ref" : "ASecurityGroup"}]},
     {"Condition" : "SomeOtherCondition"}
  ]
}
MyOrCondition:
  !Or [!Equals [sg-mysggroup, !Ref ASecurityGroup], Condition:
SomeOtherCondition]
```

Wait Conditons/Creation Policy

CreationPolicy attribute prevents resource's status from reaching create complete until AWS CloudFormation receives a specified number of success signals or the timeout period is exceeded.

- Use the cfn-signal helper script or SignalResource API to send signal
- Only available for following resources
  - AWS::AutoScaling::AutoScalingGroup
  - AWS::EC2::Instance
  - AWS::CloudFormation::WaitCondition

Example:

In template:

```
CreationPolicy:
  AutoScalingCreationPolicy:
    MinSuccessfulInstancesPercent: Integer
  ResourceSignal:
    Count: Integer
    Timeout: String
MyInstance:
  Type: AWS::EC2::Instance
  Properties:
    UserData: !Base64
      'Fn::Join':
        - ''
        - - |
            #!/bin/bash -x
          - |
            # Install the files and packages from the metadata
          - '/opt/aws/bin/cfn-init -v '
          - '         --stack '
          - !Ref 'AWS::StackName'
          - '         --resource MyInstance '
          - '         --region '
          - !Ref 'AWS::Region'
          - |+

          - |
            # Signal the status from cfn-init
          - '/opt/aws/bin/cfn-signal -e $? '
          - '         --stack '
          - !Ref 'AWS::StackName'
          - '         --resource MyInstance '
          - '         --region '
          - !Ref 'AWS::Region'
          - |+
```

In most conditions, CreationPolicy is preferable to **WaitConditon**. Use a wait condition for situations like the following:

- To coordinate stack resource creation with configuration actions that are external to the stack creation
- To track the status of a configuration process

Example

```
WebServerGroup:
  Type: AWS::AutoScaling::AutoScalingGroup
  Properties:
    AvailabilityZones:
      Fn::GetAZs: ""
    LaunchConfigurationName:
      Ref: "LaunchConfig"
    MinSize: "1"
    MaxSize: "5"
```

```
    DesiredCapacity:
      Ref: "WebServerCapacity"
    LoadBalancerNames:
      -
        Ref: "ElasticLoadBalancer"
WaitHandle:
  Type: AWS::CloudFormation::WaitConditionHandle
WaitCondition:
  Type: AWS::CloudFormation::WaitCondition
  DependsOn: "WebServerGroup"
  Properties:
    Handle:
      Ref: "WaitHandle"
    Timeout: "300"
    Count:
      Ref: "WebServerCapacity"
```

Nested Stack

Stack is a resource which has following benefits

- Overcome limits of CloudFormation
- Split large number of resources over multiple templates
- Reuse common template patterns

Resource Deletion Policies

Three types of DeletionPolicy for each resource

- Delete (default)
- Retain
- Snapshot (only on a few services)

Stack Updates

Use Stack Policy to control actions Example:

```
{
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*"
    },
    {
      "Effect" : "Deny",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "LogicalResourceId/ProductionDatabase"
```

```
    }
  ]
}
```

- No stack policy = allow all updates
- Once a stack policy is applied, can't be deleted
- Once a policy is applied, all resources are protected by default
    - `Update:*` is denied

Updates can impact a resource in 4 ways, depending on resource

1. No interruption
2. Some interruption
3. Replacement
4. Deletion

Custom Resources

Create resources outside of the available AWS resources. Involves 3 parties:

1. Template developer
    - Creates a template that includes a custom resource type. The template developer specifies the service token and any input data in the template.
2. custom resource provider
    - Owns the custom resource and determines how to handle and respond to requests from AWS CloudFormation. The custom resource provider must provide a service token that the template developer uses.
3. AWS CloudFormation
    - During a stack operation, sends a request to a service token that is specified in the template, and then waits for a response before proceeding with the stack operation.

Steps of creating a custom resource

1. Template developer defines a custom resource in his or her template, which includes a service token and any input data parameters.
    - input data might be required; service token is always required.
    - service token specifies where AWS CloudFormation sends requests to, such as to an Amazon SNS topic ARN or to an AWS Lambda function ARN
2. During create, update, or delete a custom resource, AWS CloudFormation sends a request to the specified service token.
    - service token must be in the same region in which you are creating the stack.
    - CloudFormation includes information such as the request type and a pre-signed Amazon Simple Storage Service URL, where the custom resource sends responses

3. Custom resource provider processes the AWS CloudFormation request and returns a response of SUCCESS or FAILED to the pre-signed URL.
    - Response is in a JSON-formatted file
    - Custom resource provider can also include name-value pairs that the template developer can access
4. After getting a SUCCESS response, AWS CloudFormation proceeds with the stack operation. If a FAILED response or no response is returned, the operation fails.

CloudFormation Best Practices

- Organize Your Stacks By Lifecycle and Ownership
    - Two common frameworks:
        a. **multi-layered architecture**
        - Layered architecture organizes stacks into multiple horizontal layers that build on top of one another, where each layer has a dependency on the layer directly below it. Layers can have multiple stacks grouped by owner/lifecycle.

        b. **service-oriented architecture (SOA)**

        - A service-oriented architecture organizes problems into manageable parts, each with a clearly defined purpose representing a self-contained unit of functionality. You can map these services to a stack, with its own lifecycle and owners.
- Use Cross-Stack References to Export Shared Resources
    - Use cross-stack references to export resources from a stack so that other stacks can use them. Stacks can use the exported resources by calling them using Fn::ImportValue
- Use IAM to Control Access
    - Manage users permissions for viewing stack templates, creating stacks, or deleting stacks
    - Separate permissions between a user and the AWS CloudFormation service using a service role. AWS CloudFormation uses the service role's policy to make calls instead of the user's policy.
- Verify Quotas for All Resource Types
    - for example: you can only launch 200 AWS CloudFormation stacks per region in your AWS account
- Reuse Templates to Replicate Stacks in Multiple Environments
- Use Nested Stacks to Reuse Common Template Patterns
- Do Not Embed Credentials in Your Templates
- Use AWS-Specific Parameter Types
    - For example, you can specify a parameter as type AWS::EC2::KeyPair::KeyName, which takes an existing key pair name that is in your AWS account and in the region where you are creating the stack

- Use Parameter Constraints
- Use AWS::CloudFormation::Init to Deploy Software Applications on Amazon EC2 Instances
    - Describe the configurations that you want rather than scripting procedural steps. Update configurations without recreating instances.
- Use the Latest Helper Scripts
    - Inlcude following command in userdata: `yum install -y aws-cfn-bootstrap`
- Validate Templates Before Using Them
    - check for valid json or yaml
- Manage All Stack Resources Through AWS CloudFormation
    - do not update resources outside of stack template
- Create Change Sets Before Updating Your Stacks
    - See how proposed changes to a stack might impact your running resources before you implement them.
- Use Stack Policies
    - prevent unintentional disruptions by protecting important resources
- Use Code Reviews and Revision Controls to Manage Your Templates
- Update Your Amazon EC2 Linux Instances Regularly
    - `yum update`

Elastic Beanstalk

Upload an application and Elastic Beanstalk automatically handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring

- Useful for developers who only want to write code and not maintain infrastructure
- supports many languages as well as Docker, allowing extensibility for languages not natively supported

ebextentions

.ebextensions allow advanced environment customization with YAML or JSON configuration files

- Placed in a folder called `.ebextensions`
- Used for achieving automation in creation Elastic Beanstalk environment

AWS Config

Enables you to assess, audit, and evaluate the configurations of your AWS resources Allows for:

- Continuous monitoring
  - Get SNS notifications on changes
- Continuous assessment
  - keep resources in line with policies
- Troubleshooting
  - compare points in time
  - integrates directly with cloudtrail
- Compliance monitoring
  - view compliance status across entire enterprise in AWS
- Change management
  - Track resource relationships
  - Review resource dependencies

At a high level:

- Define rules that are used to check compliance of resources in your account
- View compliance in console or receive warnings via SNS

ECS

Highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances.

- Highly scalable
- Fast
- Manage docker containers on a cluster
- Supports API calls
- Supports scheduling

Lambda Step Functions

- Service that allows you to orchestrate your lambda functions
- Reliable way to step through functions in a particular order
- Give graphical view of application components

Use cases:

- Data processing: consolidate data from multiple databases into unified reports, refine and reduce large data sets into useful formats, or coordinate multi-step analytics and machine learning workflows

- DevOps and IT automation: build tools for continuous integration and continuous deployment, or create event-driven applications that automatically respond to changes in infrastructure
- E-commerce: automate mission-critical business processes, such as order fulfillment and inventory tracking
- Web applications: implement robust user registration processes and sign-on authentication

OpsWorks

There are three OpsWorks services

1. AWS OpsWorks for Chef Automate
    - Fully managed service configuration management service that hosts Chef
2. AWS Opsworks for Puppet Enterprise
    - Fully managed service configuration management service that hosts Puppet
3. AWS OpsWorks Stacks
    - Application and server management service
    - Allows application deployment in stacks
    - Configuration using Chef Solo
    - This is the service that will be in exam

What OpsWorks does:

- Manage application on AWS and on-premises
- Design layers that perform different functions
- Supports autoscaling and scheduled scaling
- Implements chef solo for configuration

Chef:

OpsWorks Stacks/Chef is a declarative state engine

- State what you want to happen, service handles how

Opsworks stacks/Chef has resources it can use:

- pacakages to install
- services to control
- configuraton files to update

Chef Recipes

- Use Chef recipes to automate everything Chef can do, such as creating directories and users, configuring databases, etc

Berkshelf

- Manage external cookbooks to safely include them in your stacks

Monitoring and Logging

CloudWatch

- Metric gathering
- Monitoring / Alerting
- Graphing
- Logging

Retention Period (good to know for exam)

- Data points < 60 seconds are available for 3 hours (high resolution)
- Data points = 60 seconds are available for 15 days
- Data points = 300 seconds (5 minutes) are available for 63 days
- Data points = 3600 (1 hour) are available for 445 days (15 months)

Note: shorter term data points are averaged to the next tier at the end of their lifecycle. e.g. a 1 second data point becomes a 1 minute average after 3 hours

CloudWatch Metrics

Concepts:

- **Metrics**: A metric represents a time-ordered set of data points that are published to CloudWatch.
    - Metrics are uniquely defined by a name, a namespace, and zero or more dimensions. Each data point in a metric has a time stamp, and (optionally) a unit of measure.
    - Metrics exist only in the region in which they are created. Metrics cannot be deleted, but they automatically expire after 15 months if no new data is published to them.

- **Namespace**: A namespace is a container for CloudWatch metrics. AWS namespaces use the following naming convention: AWS/service.
- **Dimension**: A dimension is a name/value pair that is part of the identity of a metric. You can assign up to 10 dimensions to a metric.
    - For example, you can get statistics for a specific EC2 instance by specifying the InstanceId dimension when you search for metrics.

Cloudwatch Events

Concepts:

- **Events**: An event indicates a change in your AWS environment. AWS resources can generate events when their state changes.
    - e.g. Amazon EC2 generates an event when the state of an EC2 instance changes from pending to running
- **Targets**: A target processes events.
    - Targets can include:
        - Amazon EC2 instances
        - AWS Lambda functions
        - Kinesis streams
        - Amazon ECS tasks
        - Step Functions state machines
        - Amazon SNS topics
        - Amazon SQS queues
        - built-in targets
    - A target receives events in JSON format.
- **Rules**: A rule matches incoming events and routes them to targets for processing.
    - A single rule can route to multiple targets, all of which are processed in parallel.
    - Rules are not processed in a particular order.
    - A rule can customize the JSON sent to the target, by passing only certain parts or by overwriting it with a constant.

CloudWatch Logs

Centralize the logs from all of your systems, applications, and AWS services that you use, in a single, highly scalable service.

- view logs

- search/query them for specific error codes or patterns
- filter them based on specific fields
- archive them securely for future analysis
- visualize log data in dashboards

Concepts:

- **Log Events**: A log event is a record of some activity recorded by the application or resource being monitored.
    - contains two properties: the timestamp of when the event occurred, and the raw event message. Event messages must be UTF-8 encoded.
- **Log Streams**: A log stream is a sequence of log events that share the same source.
    - intended to represent the sequence of events coming from the application instance or resource being monitored.
- **Log Groups**: Log groups define groups of log streams that share the same retention, monitoring, and access control settings.
    - Each log stream has to belong to one log group.

X-Ray

- Collects data about requests that your application serves
- Provides tools you can use to view, filter, and gain insights into that data
- Identify issues and opportunities for optimization

Policies and Standards Automation

Service Catalog

Allows IT administrators to create, manage, and distribute catalogs of approved products to end users, who can then access the products they need in a personalized portal

- Has an API that provides programmatic contol over all end-user actions as an alternative to AWS Console
- Allows you to create custom interfaces and apps

- Allows administrators to create and distribute application stacks called **products**
- Products can be grouped into folders called **portfolios**
- Users can then launch and manage products without requiring access to AWS services/console
- Users can only see products they are supposed to see

Trusted Advisor

Service that provides real time guidance to ensure resources are provisioned and managed correctly, following AWS best practices

Categories:

- Cost optimization
- Performance
    - e.g. under-utilizing resources
- Security
- Fault tolerance
- Service Limits

Systems Manager

- Collecting software inventory
- Applying OS patches
- Creating system images
- Configuring operation systems
- Manage hybrid cloud systems from a single interface (AWS, on-prem)
- Reducing costs

Features

- Run command
    - remote management of servers
- State manager
    - manage all your servers' configuration
    - e.g. firewall settings
    - e.g. anti-virus definitions
- Inventory

    - info on installed applications
- Maintentance Window
- Patch Manager
    - e.g. select and deploy software across all EC2 instances
- Automation
    - automate repetitive IT tasks
- Parameter Store

Organizations

Policy-based management for multiple AWS accounts.

Accounts can be split several ways within business: Environment (dev, qa, prod), Projects, or Business Units (Sales, support, dev, etc)

AWS Organizations allows you to:

- Programmatically create new accounts
- Create and maintain a group of accounts
- Set policies on those groups
- Set single payer, cost tracking

Overrides IAM policies with an organization policy

Secrets Manager

Service to help protect secrets needed to access applications, service, and IT resources

- Encrypts secrets at rest using your own encrpytion keys stored in KMS
- Secrets can be database credentials, passwords, 3rd part API keys, any text, etc.
- Store and control access to them with Secrets Manager Console/CLI/API/SDK
- Hardcoded credentials in code is replaced with API call
- Secrets can be rotated automatically according to your own schedule

Macie

Security service that uses machine learning to automatically discover, classify, and protect sensitive data in AWS

Managed service:

- Can recognize any Personally Identifiable Information (PII)
- Provides dashboard showing how data is accessed/moved
- Monitors data access for anomalies
- Generates detailed alerts when it detects risk of unauthorized access or accidental data leaks
- Currently only protects S3, but more is planned

Certificate Manager

Easily provision, manage, and deploy SSL/TLS certificates

- Centrally manage certificates in AWS
- Audit the use of each certificate in CloudTrail logs
- Private Certificate Authority
- AWS integration
- Import 3rd party certificates from other CAs

Incident and Event Response

GuardDuty

A threat detection service that continuously monitors for malicious or unauthorized behavior

It does this by monitoring for:

- Unusual API calls
- Unauthorized deployments
- Compromised instances

It uses:

- Threat intelligence feeds

- Machine Learning
- CloudWatch Events

How it works:

1. Enable service
2. Analyze continuously
3. Detect Threats using machine learning
4. Act

- Detailed analysis is available in console
- Can integrate logging, alerting, or trigger lambda

Amazon Inspector

Automated service that assess your applications for vulnerabilities and produces a security findings report. Mainly based around protecting EC2 instances.

- Identify security issuesu
- API driven
- Reduces risk by warning you of risk before it's a problem
- Leverage expertise: experts researching potential security issues
- Define and Enforce standards

How it works:

- Network assessments (no agent needed)
    - network configuration check (see if ports are reachable from outside VPC)
- Host assessment (Inspector Agent on EC2)
    - Can be automatically installed via Systems Manager run command
- Can run weekly or once

Amazon Kinesis

Collect, process, and analyze video and data streams in real time

Made of 4 services

- Kinesis Data Analytics

- Kinesis Data Firehose
- Kinesis Data Streams
- Kinesis Video Streams

Kinesis Data Analytics

- Analyze streaming data
- Respond in real time
- Query data using SQL
- completely managed service
- pay as you go (for what you use)

Kinesis Firehose

- Deliver streaming data
- No applications to manage
- Just configure data producer to send data to Firehose
    - Firehose send data to:
        - S3
        - Redshift
        - Amazon ElasticSearch
        - Splunk
    - Accepts records in chunks up to 1000 kb
- Data can be transformed

Kinesis Data Streams

- Collect streaming data
- Massively scalable
- Capture GBs per second
- data is available in milliseconds
- durable, streamed across 3 data centers
- data stored for 7 days
- Elastic, dynamically scale

Kinesis Video Streams

- Collect streaming video

- Handle ingestion from millions of devices
- Enables live and on-demand playback
- Can take advantage of Amazon Rekognition and Machine Learning Frameworks
- Access data through APIs
- Build real-time, video-enabled applications

Kinesis Overview

- Kinesis Data Analytics: Analyze streaming data using SQL
- Kinesis Firehose: Deliver streaming data to another AWS service, e.g. S3
- Kinesis Data Streams: Collect streaming data and do things with it (e.g. create dashboard)
- Kinesis Video Streams: Collect streaming video data and do things with it

Exam hint: Kinesis will be an answer if it has to do with large amounts of data

High Availability, Fault Tolerance, & Disaster Recovery

Single Sign-On

Cloud service that makes it easy to centrally manage single sign on access to multiple AWS accounts and business applications

Features

- Integrates with AWS Organizations
- SSO access to cloud applications
- Create/Manage users and groups
- Use existing corporate identities (Active Directory)

Relevance

- High availability
- Fault tolerant

CloudFront

A fast content delivery network (CDN) that securely delivers data, video, applications, and APIs to customers globally

Steps

1. Specify origin servers, like an Amazon S3 bucket or your own HTTP server
2. An origin server stores the original, definitive version of your objects, either in an Amazon S3 bucket or an HTTP server, such as a web server
3. Adobe Media Server RTMP protocol is always an Amazon S3 bucket (streaming video)
4. Upload files to your origin servers: web pages, images, and media files, or anything that can be served over HTTP or a supported version of Adobe RTMP (Adobe Flash Media Server)
5. With S3, you can make the objects in your bucket publicly readable or keep objects private
6. Create a CloudFront distribution, which tells CloudFront which origin servers to get your files from when users request the files through your web site or application. Specify details such as whether you want CloudFront to log all requests and whether you want the distribution to be enabled as soon as it's created.
7. CloudFront assigns a domain name to your new distribution that you can see in the CloudFront console or via API.
8. CloudFront sends your distribution's configuration (but not your content) to all of its edge locations

AutoScaling

Scale EC2 instance capacity automatically according to defined conditions

- High availability
- Better fault tolerance
- Cost savings

AutoScaling Lifecycle

- Starts when AutoScaling group launches an instance
- Ends when you terminate an instance
- Ends when AutoScaling group takes the instance out of service and terminates it

Overview

- Starts to scale out
- **Pending state**
    - Hook: EC2_INSTANCE_LAUNCHING
        - Pending:Wait
        - Pending:Proceed
- In **Service State**
- If instance fails or taken out of service, moves to **Terminating State**
    - Hook: EC2_INSTANCE_TERMINATING
        - Terminating:Wait
        - Terminating:Proceed
- Optional: **Standby state**
    - Managed by autoscaling group, but don't receive traffic
    - EnteringStandby
    - Standby
- When **Standby state** is put back into service, it enters **Pending State**
- Optional: Detach Instance
    - Detaching state
    - Detached state
    - Can be stand-alone EC2 or re-attached to another autoscaling group

How Lifecycles Work

1. Autoscaling responds to scale out event by launching an instance
2. Autoscaling puts the instance into **Pending:Wait** state
3. Autoscaling sends a message to notification target defined for the hook, along with information and token
4. Waits until you tell it or timeout ends
5. You can perform custom action, install software, etc.
6. By default, instance will wait 1 hour and change state to **Pending:Proceed**, then enter **InService** state

Notes on Lifecycle Hooks

- You can change heartbeat timeout or define it when you create the hook in the CLI with `heartbeat-timeout` parameter
- You can call the `complete-lifecycle-action` to tell autoscaling group to proceed
- You can call the `record-lifecycle-action-heartbeat` command to add more time to timeout
- 48 hours is the maximum amount you can keep server in wait state

Cooldowns

- Cooldowns ensure that autoscaling group does not launch or terminate more instances than needed
- Cooldowns start when an instance enter InService state, so if instance is left in Pending:Wait, autoscaling will wait before adding any other servers

Abandon or Continue

- At conclusion of lifecycle hook, an instance can result in two states: `abandon` or `continue`
- Abandon will cause autoscaling group to terminate instance and, if necessary, launch a new one
- Continue will put instance into service

Spot Instances

- can use lifecycle hooks with spot instances
- does NOT stop an instance from terminating due to change in Spot Price
- When spot instance terminates, you must complete lifecycle action

Route 53

Highly available and scalable cloud Domain Name System (DNS) service

Features

- Highly Available
- Interface directly with EC2 and S3
- Fault Tolerant
    - multiple routing types (e.g. latency based routing, weighted round robin)

Routing Policies

- Failover routing policy
  - for active-passive failure
- Geolocation routing policy
  - route based on location of users
- Geoproximity routing policy
  - route traffic based on location of resources and users
- Latency routing policy
  - route traffic to region with best latency
- Multivalue answer routing policy
  - route randomly to up to 8 destinations
- Weighted routing policy
  - route to different resources using a percentage split
  - good for **A/B Testing**

#RDS Amazon Relational Database Service - create, run, scale relational DBs in the cloud

Benefits

- Fast
- Cost effecient
- Resizable
- Secure
- Highly Available
- Minimal administration

Supported DB engines

- MySQL
- MariaDB
- Microsoft SQL Server
- PostgreSQL
- Oracle
- Amazon Aurora

Managemed Administration - AWS - Provision infrastructure - install DB software - automatic backups - automatic patching - synchronous data replication - automatic failover - You - Settings - Schema - Performance tuning

Scaling - Vertical: - Change instance type - discounts for reserved instances - storage can be scaled live except MSSQL - Horizontal - Read replicas - Read-only copies synched with master db - can be in different regions - can be promoted for disaster recovery - can create a replica of a replica

Aurora

MySql and Postgres compatible DB built for the cloud

Benefits

- fast and reliable
- simple
- cost effective
- 5x throughput of mysql on same hardware
- compatibly with mysql 5.7
- fault tolerant and self healing
- disk failures repaired in background
- detects crashes and restarts
- no crash recovery or cache rebuilding required
- automatic failover to one of up to 15 read replicas
- storage autoscaling

Backups

- Automatic, continuous, or incremental
- point-in-time restore within second
- up to 35 day retention period
- no impact on db performance

Snapshots

- user-initiated snapshots stored in s3
- kept until explicitly deleted
- incremental

Failure and Fault Tolerance

- maintains 6 copies of data across 3 AZs
- point in time/snapshot restore

- data divided into 10 gb segments across many disks
- transparently handles loss
- can lose 2 copies without affecting ability to write
- can lose 3 copies without affecting ability to read
- self-healing

Replicas

- Amazon Aurora replicas
    - share underlying volume with primary instance
    - updates made by primary are visible to all replicas
    - up to 15
    - low performance impact on memory
    - replica can be failover target
- MySQL read replicas
    - Primary instance data is replayed on replica as transactions
    - up to 5
    - high performance impact on primary
    - replica can be target, but may have minutes of data loss

Security

- Must be in VPC
- SSL secures data in transit
- can encrypt data with KMS
- encrypted storage at storage/backup/snapshots/replicas
- NOTE: you can't encrypt an unencrypted database

DynamoDB

Fully managed NoSQL database that supports key-value and document data structures

Details

- no servers
- no storage limitations
- fully resilient, highly available
- performance scales in a linear way

- fully intregrated with IAM

Concepts

- Collection of tables, highest level of structure
- specify performance requirements on table
- WCUs - write capacity units - number of 1 kb writes per second
- RCUs - read capacity units - number of 4 kb reads per second

DynamoDB Tables

- contain items
- items contain attributes (including partition key, which must be unique)
- attributes can also include sort key

Attribute Types

- string
- number
- binary
- boolean
- null
- Document (lists, maps)
- Set (array)

Streams

- Optional feature capturing data modification events
- Data events appear in the stream in near real time and in order
- Each event is represented by a stream record when
    - new item is added
    - item is update
    - item is deleted
- Can trigger lambda when a certain event appears in the stream

Overview

This guide contains all the resources used, recommendations and tips for how to study for the AWS - DevOps Engineer Professional Exam.

1. Re:Invent Videos

The following Re:Invent videos cover a large chunk of the material that is in the exam. We recommend watching them at 1.5x the speed. 1.5x is fast enough to help get through the videos quickly yet slow enough to be able to comprehend the material.

Moving to DevOps the Amazon Way

Serverless Architecture and Best Practices

I didn't know Amazon API Gateway did that

Using DevOps, Microservices, & Serverless to Accelerate Innovation

Become a Serverless Black Belt: Optimizing Your Serverless Appli

Monitor All Your Things: Amazon CloudWatch in Action with BBC

2. Whitepapers

These are to all the whitepapers I read, took notes of and reviewed closer to the exam. It is tough to actually read all of these as it is a lot of material. You may skim through them but you must be really attentive/focused as you do so. There is a lot of little details in these whitepapers; details that are crucial to passing the exam.

AWS DevOps

Running containerized microservices on AWS

Microservices on AWS

Infrastructure as Code

Practicing CI/CD on AWS

Jenkins on AWS

Import Windows Server to Amazon EC2

AWS Blue Green Deployments

AWS Development and Test on AWS

3. AWS Free course

We highly recommend going through this amazing, short, FREE course created by AWS:

Exam Readiness: AWS Certified DevOps Engineer – Professional

This course assumes that you have strong knowledge of the exam material. This course does not cover all the exam material. Instead, this course is about how to approach the different questions in the exam. It breaks down the thought process of those who answer these very challenging questions correctly. We recommend going through this course after you have gone through the above mentioned videos and whitepapers.

4. Other supporting material

Exam Guide

Sample Question

- Practice Exam: If you have previously taken an AWS exam, you have the benefit of taking a free practice exam. Go to your AWS Certification account > Benefits > Practice Exam and follow the prompts to set it up. This practice exam is vital. It contains a lot of material that is close to the material in the actual exam.

5. Terminology

As you are studying for the exam, you will come across a lot of different AWS terminology and supporting information. Make a point of noting and researching the ones that you don't necessarily know and looking them up.

Here is a list of the ones that we have come across.

- Serverless Application Model (SAM)
- Caching on API Gateway
- Canary deployments on API Gateway
- Cloudformation Custom Resources
- ebextensions configuration files

- Customizing server for ElasticBeanstalk
- DynamoDB TTL
- S3 lifecycle policies with Tags
- CodeDeploy on on-premise instances
- Using SSH Keys in codecommit
- Trusted Advisor - Service limits
- Config - All the rules
- Codebuild Artifacts
- Different cloudformation functions
- cfn-init, cfn-signal, cfn-metadata and cfn-hup files for cloudformation
- Cloudformation stack policies
- Autoscaling lifecycle with hooks
- EC2 instance user data

6. Services seen in the exam

- Codecommit
- COdebuild
- Codedeploy
- Codepipeline
- Cloudformation
- Opsworks
- Lambda
- API Gateway
- X-Ray
- Config
- Trusted Advisor
- Cloudwatch Events
- Cloudwatch Logs
- Elasticache
- ElasticBeanstalk
- ECR
- ECS
- DynamoDB with DAX
- RDS
- EC2

- Route53
- Cloudfront
- Auto Scaling and Load balancers (Heavy emphasis on this)
- Certificate Manager
- Systems Manager
- Service Catalog
- Kinesis Datastreams vs Firehose
- GuardDuty
- AWS Server Migration
- AWS SSO
- Cost Optimization over all

7: Exam Strategy

- Keep track of time
- Read both the question and answer in full
- Identify the key words in the question and make sure to satisfy ALL requirements of the question. Two answers may be very similar but a detail in the question will set them apart.
- Spot the distractor/silly answers. Move on quickly from these.
- Watch out for mental exhaustion. If you feel this, close your eyes and take 5 deep breaths.
- The questions will not be super clear. It's ok to make some assumptions.
- An answer that says "Don't do anything", is not the right answer.
- Avoid answers that have you doing manual commands/process.
- Some answers would not be the best way to do things but could still be right.
- Focus on simplest, most technically correct answers

8: Final Note

Request: If you study for this exam and pass, please add your experience about SkillCertPro to help your colleagues. Good luck in your exam!!