CloudX Associate: AWS for Testers

IAM

TASK 2 – IAM

1. DEPLOY THE IAM APPLICATION

Deploy the **cloudxiam** CDK stack: deployment instructions

2. DEPLOYMENT VALIDATION

Create a **manual/automated deployment validation** test suite that covers the following requirements:

CXQA-IAM-01: 3 IAM policies are created according to the following requirements:

Policy	Actions Allowed	Resources	Effect
FullAccessPolicyEC2	ec2:*	All	Allow
FullAccessPolicyS3	s3:*	All	Allow
ReadAccessPolicyS3	s3:Describe*, s3:Get*, s3:List*	All	Allow

CXQA-IAM-02: 3 IAM roles are created according to the following requirements:

Role	Policies
FullAccessRoleEC2	FullAccessPolicyEC2
FullAccessRoleS3	FullAccessPolicyS3
ReadAccessRoleS3	ReadAccessPolicyS3

CXQA-IAM-03: 3 IAM users groups are created according to the following requirements:

Group	Policies
FullAccessGroupEC2	FullAccessPolicyEC2
FullAccessGroupS3	FullAccessPolicyS3
ReadAccessGroupS3	ReadAccessPolicyS3

CXQA-IAM-04: 3 IAM users are created according to the following requirements:

User	Group
FullAccessUserEC2	FullAccessGroupEC2
FullAccessUserS3	FullAccessGroupS3
ReadAccessUserS3	ReadAccessGroupS3

2.1 Testing Tools:

- AWS Console
- AWS CLI
- AWS SDK (for automated tests).

3. REGRESSION TESTING

- Deploy version 2 of the **cloudxiam** application deployment instructions.
- Execute deployment validation tests against new application version deployment.
- Create bug/root cause report for the found regression issues.

4. ENVIRONMENT CLEAN-UP

Delete the **cloudxiam** application stack and clean up the environment: clean-up instructions.

5. SUBMIT RESULTS

Upload the home task artifacts (screenshots, test cases/links to automated tests code in the git repository) to Learn Portal and change the task status to „Needs Review”.

IMPORTANT THINGS TO KEEP IN MIND

1. Once you create AWS Account, setup Multi-factor Authentication!
2. Do NOT share your account!
3. Do NOT commit your account Credentials to the Git repository!
4. Terminate/Remove (destroy) all created resources/services once you finish the module (or the learning for the day)!
5. Please Do NOT forget to delete NAT Gateway if you used it!
6. Do NOT keep the instances running if you don’t use it!
7. Carefully keep track of billing and working instances so you don't exceed limits!

AWS IAM THEORY

What is IAM?

AWS IAM stands for "Identity and Access Management" and it is a web service provided by Amazon Web Services (AWS) that helps to securely control access to AWS services and resources for users. IAM allows to manage users, groups, roles, and permissions, enabling to securely control who can access what resources within AWS environment. Below are some key features and concepts of AWS IAM:

- Users: Represent individuals or entities (such as applications) that interact with AWS resources. Each user has unique security credentials (username and password or access keys).
- Groups: A collection of IAM users. Groups make it easier to manage permissions for multiple users with similar roles.
- Roles: An IAM role is similar to a user, but it is not associated with a specific user identity. Instead, it is assumed by anyone who needs it for a set of permissions. Roles are commonly used to delegate access to AWS resources or services.
- Permissions: IAM allows to define fine-grained permissions to control access to AWS resources. Permissions are defined using policies, which are JSON documents that specify the actions allowed or denied on specific resources.
- Policies: IAM policies are JSON documents that define permissions. They can be attached to users, groups, or roles, and they specify what actions are allowed or denied on which resources.

What are the typical use case scenarios for IAM?

AWS IAM is used in various scenarios to manage access to AWS resources securely. Here are some typical use case scenarios for IAM:

- User Management: IAM allows to create and manage users who can access AWS account. You can create individual IAM users for each person or application that needs access to AWS resources. This enables to grant unique security credentials to each user and track their activities separately.
- Access Control: IAM enables to define fine-grained permissions to control access to AWS resources. You can create IAM policies that specify which actions are allowed or denied on specific resources for different users or groups. This allows you to implement the principle of least privilege, ensuring that users have only the permissions they need to perform their jobs.
- Role-Based Access Control (RBAC): IAM supports role-based access control, allowing to define roles with specific permissions and assign them to users or groups. Roles are commonly used to delegate access to AWS resources or services to applications running on EC2 instances, AWS Lambda functions, or other AWS services.
- Temporary Credentials: IAM supports the generation of temporary security credentials that are valid for a limited duration. These temporary credentials can be used by applications or users to access AWS resources securely without storing long-term access keys. Temporary credentials are commonly used in scenarios such as cross-account access or federated access.
- Multi-Factor Authentication (MFA): IAM supports multi-factor authentication, which adds an extra layer of security to user sign-ins. You can require users to present two or more forms of authentication (factors) before they can access AWS resources, reducing the risk of unauthorized access.

What are user groups and security groups in AWS? What’s the difference?

User Groups:

- Definition: User groups in AWS IAM are collections of IAM users. Instead of assigning permissions directly to individual users, you can create user groups and attach IAM policies to them. Users within a group inherit the permissions granted to that group.

- **Purpose:** User groups help simplify permission management by allowing to assign permissions to multiple users simultaneously. Instead of updating permissions for each user individually, you can update the permissions for the group, and those changes automatically apply to all users within the group.
- **Use Cases:** User groups are commonly used to organize users based on their roles or responsibilities within an organization. For example, you might create user groups such as "Developers," "Administrators," or "Accounting Team" and assign appropriate permissions to each group.

Security Groups:

- **Definition:** Security groups in AWS are associated with EC2 instances and control inbound and outbound traffic at the instance level. Each security group acts as a virtual firewall, specifying which traffic is allowed to reach the associated instances.
- **Purpose:** Security groups provide network security by defining rules that control the traffic to and from EC2 instances. You can specify rules based on protocols, ports, and IP addresses to allow or deny traffic as needed.
- **Use Cases:** Security groups are used to control access to EC2 instances and other AWS resources. For example, you can create security groups to allow inbound SSH (port 22) access for administration purposes or to restrict access to specific IP ranges.

Key Differences:

- **Level of Operation:** User groups operate at the IAM level and are used to manage permissions for IAM users, while security groups operate at the EC2 instance level and are used to control network traffic.
- **Functionality:** User groups are used for managing permissions and access to AWS resources, while security groups are used for defining network access control rules for EC2 instances.
- **Scope:** User groups are applicable across the entire AWS account and can be used to manage permissions for various AWS services, whereas security groups are specific to individual EC2 instances and apply only to the traffic entering or leaving those instances.

What are IAM policies?

IAM policies in AWS are JSON documents that define permissions and access control rules. These policies are used to specify what actions are allowed or denied on AWS resources for IAM users, groups, or roles. IAM policies are a fundamental component of IAM and are used to enforce security best practices, control access to resources, and ensure compliance with organizational policies and regulatory requirements.
There are some key aspects of IAM policies:

- **JSON Format:** IAM policies are written in JSON format. They consist of one or more statements, each of which defines a set of permissions.
- **Statements:** Each IAM policy has one or more statements. Each statement consists of the following components:
  - *Effect*: Specifies whether the statement allows or denies access. It can have two possible values: "Allow" or "Deny".
  - *Action*: Specifies the AWS service actions that the policy allows or denies. Actions are represented as strings, such as "s3:GetObject" or "ec2:RunInstances".
  - *Resource*: Specifies the AWS resources to which the actions apply. Resources are represented as Amazon Resource Names (ARNs), such as "arn:aws:s3:::example-bucket/" or "".
- **Permissions:** IAM policies define permissions by specifying which actions are allowed or denied on which resources. You can create policies that grant broad permissions (e.g., full access to all AWS services) or narrow permissions (e.g., read-only access to specific S3 buckets).
- **Policy Attachments:** IAM policies can be attached to IAM users, groups, or roles. When a policy is attached to an identity (user, group, or role), the permissions defined in the policy are applied to that identity.

What is EC2 and S3?

Amazon EC2 (Elastic Compute Cloud):

- Definition: Amazon EC2 is a web service that provides resizable compute capacity in the cloud. It allows users to launch and manage virtual servers, known as instances, on-demand.
- Purpose: EC2 enables users to run applications, host websites, and perform various computing tasks in the cloud. Users can choose from a wide range of instance types with different CPU, memory, storage, and networking capacities to meet their specific requirements.
- Features: EC2 instances offer flexibility, scalability, and control over computing resources. Users can launch instances within minutes, scale up or down as needed, and pay only for the compute capacity they use.

Amazon S3 (Simple Storage Service):

- Definition: Amazon S3 is a scalable object storage service designed to store and retrieve any amount of data from anywhere on the web. It provides highly durable and available storage infrastructure for a wide variety of use cases.
- Purpose: S3 is used for storing and managing data, such as static website files, application data, backups, log files, and media content. It is commonly used as a data lake, content repository, or backup and archival solution.
- Features: S3 offers features such as high availability, durability, scalability, security, and low-latency access to data. It provides a simple and intuitive interface for uploading, downloading, and managing objects (files) stored in buckets (containers).

How to allow a user to programmatically access AWS resources?

To allow a user to programmatically access AWS resources, you typically need to perform the following steps:

- Create an IAM User: First, you need to create an IAM user in your AWS account. An IAM user represents an individual or entity that interacts with AWS services. You can create a new IAM user through the AWS Management Console.
- Generate Access Keys: After creating the IAM user, you need to generate access keys for the user. Access keys consist of an access key ID and a secret access key, which are used to authenticate programmatic requests to AWS services. Access keys are required when making API calls using AWS SDKs or CLI.
- Assign Permissions: Next, you need to assign permissions to the IAM user. Permissions are defined using IAM policies, which specify what actions the user is allowed to perform on which AWS resources. You can attach managed policies or inline policies to the IAM user to grant the necessary permissions.
- Securely Distribute Access Keys: Once access keys are generated, securely distribute them to the user. Access keys should be kept confidential and should not be shared publicly. Users can use their access keys to authenticate API requests to AWS services programmatically.
- Programmatic Access: Finally, the user can use their access keys to authenticate API requests to AWS services programmatically. They can use AWS SDKs (such as AWS SDK for Java, Python, .NET, etc.) or the AWS CLI to interact with AWS services and perform various operations, such as creating, managing, or querying resources.

What is the allow/deny priority order when policies are configured on different levels (group, user, etc.)

When policies are configured at different levels (group, user, etc.) in AWS Identity and Access Management (IAM), the priority order for allow and deny permissions is determined by the following rules:

- Explicit Deny: An explicit deny always overrides any allow. If any policy attached to the user, group, or role explicitly denies a permission, that denial takes precedence over any allow from any other policies attached to the user, group, or role, as well as any implicit allows.

- Explicit Allow: If there is no explicit deny, the user, group, or role is allowed the action unless there is an explicit deny.
- Implicit Deny: If there is no explicit allow or deny, access to the action is implicitly denied. This means that if no policy grants permission to the action, it is implicitly denied.

What ways of managing permissions for a given user do you know?

In AWS IAM, there are several ways to manage permissions for a given user:

- Managed Policies:
  - *AWS Managed Policies:* AWS provides a set of pre-configured managed policies that define common sets of permissions for various AWS services. These policies are maintained and updated by AWS and cover a wide range of use cases.
  - *Customer Managed Policies:* You can create custom managed policies tailored to your specific requirements. These policies are stored independently of users, groups, or roles and can be attached to multiple identities.
- Inline Policies:
  - Inline policies are policies that are embedded directly into an IAM user, group, or role. They are defined within the identity itself and cannot be detached or shared.
  - Inline policies provide more granularity and control over permissions compared to managed policies because they are closely associated with specific identities.
- Group Policies:
  - IAM user groups allow you to group users with similar roles or permissions together. You can attach managed or inline policies to user groups, and all users in the group inherit the permissions defined in the group policies.
  - Group policies simplify permission management by allowing you to define permissions once at the group level instead of attaching policies to individual users.
- User Policies:
  - IAM user policies are policies that are directly attached to individual IAM users. These policies define permissions specific to an individual user.
  - User policies are useful when you need to grant permissions to a specific user that differ from the permissions granted by group policies.
- Role Policies:
  - IAM roles are used to delegate permissions to entities (such as applications or services) that need to access AWS resources. Roles have policies attached to them that define the permissions granted to the role.
  - Role policies specify what actions the role is allowed to perform on which AWS resources. These policies can be managed or inline.
- Permission Boundaries:

Permission boundaries allows sto control the maximum permissions that an IAM entity (user or role) can have. They act as a limiting factor for the permissions granted by policies attached to the entity.

What places can be used by AWS CLI to gather credentials and settings?

The AWS CLI can gather credentials and settings from various places to authenticate and configure access to AWS services. These places include:

AWS CLI Configuration Files:

The AWS CLI stores configuration settings and credentials in configuration files located in the user's home directory:

- ~/.aws/config: This file contains configuration settings such as the default region, output format, and named profiles.

- ~/.aws/credentials: This file contains AWS access keys (access key ID and secret access key) used for authentication.

Environment Variables:
The AWS CLI can use environment variables to provide credentials and configuration settings. The following environment variables are commonly used:

- AWS_ACCESS_KEY_ID: Specifies the AWS access key ID.
- AWS_SECRET_ACCESS_KEY: Specifies the AWS secret access key.
- AWS_SESSION_TOKEN: Specifies the session token for temporary security credentials.
- AWS_DEFAULT_REGION: Specifies the default AWS region.
- AWS_DEFAULT_OUTPUT: Specifies the default output format for CLI commands (e.g., json, text, table).

AWS IAM Role Assume Role Provider:
The AWS CLI can automatically retrieve temporary security credentials by assuming an IAM role. This functionality is commonly used in environments where EC2 instances or other AWS services need to access AWS resources securely.

IAM Instance Profile Credentials:
When running on an EC2 instance, the AWS CLI can automatically fetch credentials from the instance metadata service (IMDS). EC2 instances can be associated with an IAM role, and the CLI retrieves temporary security credentials from the instance metadata service.

Container Credentials Provider:
If running in an ECS task or Kubernetes pod, the AWS CLI can retrieve credentials from environment variables or the task/pod metadata service, similar to how it retrieves credentials on EC2 instances.

Shared Credentials File:
Apart from the default credentials file, the AWS CLI can use a shared credentials file located in a custom location specified by the AWS_SHARED_CREDENTIALS_FILE environment variable. This is useful in scenarios where multiple users share the same credentials file.

Configuration Files and Settings for AWS SDKs:
The AWS CLI can use the same configuration files (~/.aws/config and ~/.aws/credentials) used by AWS SDKs to retrieve settings and credentials.

What output formats does AWS CLI support?

The AWS Command Line Interface (CLI) supports several output formats for displaying command results. These formats allow users to customize the output to best suit their needs. The supported output formats include:

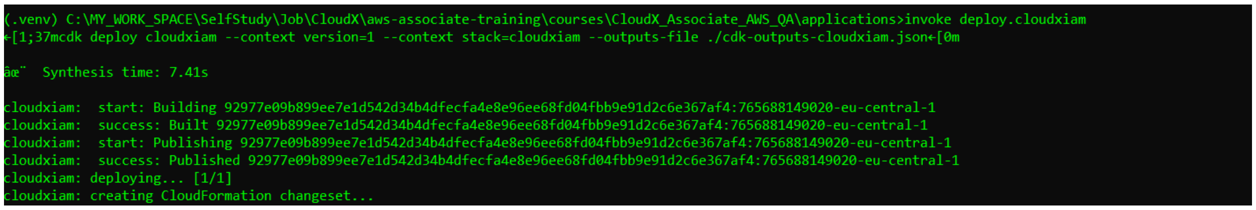
- json: Outputs the results in JSON format. JSON output is structured and easily parseable by scripts and programs.
- text: Outputs the results in a human-readable text format. Text output is suitable for reading directly in the terminal.
- table: Outputs the results in a tabular format. Table output provides a structured view of the data, making it easy to read and understand.
- yaml: Outputs the results in YAML (YAML Ain't Markup Language) format. YAML output is similar to JSON but offers a more concise and readable syntax.
- csv: Outputs the results in comma-separated values (CSV) format. CSV output is useful for exporting data to spreadsheets or other tools that support CSV format.

For example:

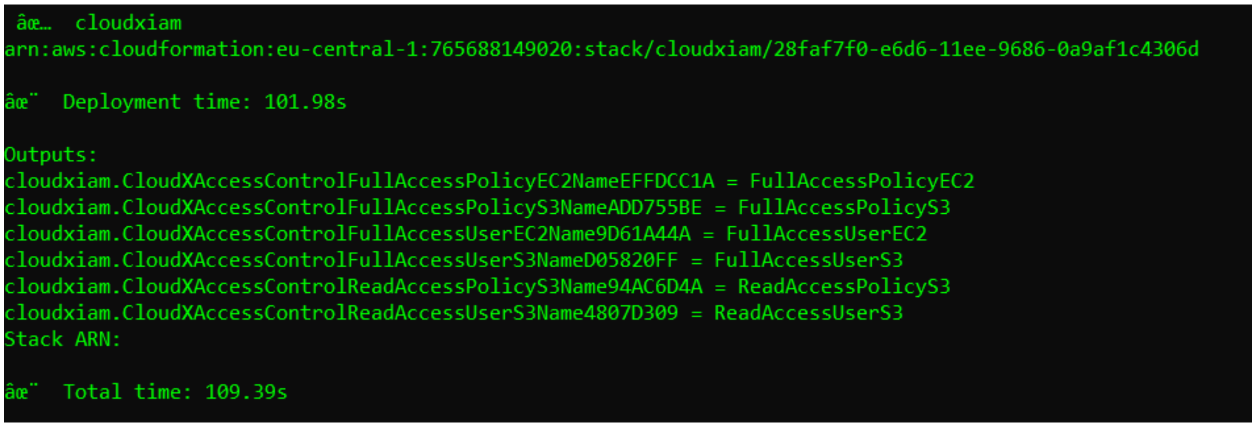
```
aws ec2 describe-instances --output json
aws s3 ls my-bucket --output table.
```

TASK IMPLEMENTATION

1. CLOUDXIAM DEPLOYMENT



```
(.venv) C:\MY_WORK_SPACE\SelfStudy\Job\CloudX\aws-associate-training\courses\CloudX_Associate_AWS_QA\applications>invoke deploy.cloudxiam
←[1;37mcdk deploy cloudxiam --context version=1 --context stack=cloudxiam --outputs-file ./cdk-outputs-cloudxiam.json←[0m

âœ¨  Synthesis time: 7.41s

cloudxiam:  start: Building 92977e09b899ee7e1d542d34b4dfecfa4e8e96ee68fd04fbb9e91d2c6e367af4:765688149020-eu-central-1
cloudxiam:  success: Built 92977e09b899ee7e1d542d34b4dfecfa4e8e96ee68fd04fbb9e91d2c6e367af4:765688149020-eu-central-1
cloudxiam:  start: Publishing 92977e09b899ee7e1d542d34b4dfecfa4e8e96ee68fd04fbb9e91d2c6e367af4:765688149020-eu-central-1
cloudxiam:  success: Published 92977e09b899ee7e1d542d34b4dfecfa4e8e96ee68fd04fbb9e91d2c6e367af4:765688149020-eu-central-1
cloudxiam: deploying... [1/1]
cloudxiam: creating CloudFormation changeset...
```



```
 âœ…  cloudxiam
arn:aws:cloudformation:eu-central-1:765688149020:stack/cloudxiam/28faf7f0-e6d6-11ee-9686-0a9af1c4306d

âœ¨  Deployment time: 101.98s

Outputs:
cloudxiam.CloudXAccessControlFullAccessPolicyEC2NameEFFDCC1A = FullAccessPolicyEC2
cloudxiam.CloudXAccessControlFullAccessPolicyS3NameADD755BE = FullAccessPolicyS3
cloudxiam.CloudXAccessControlFullAccessUserEC2Name9D61A44A = FullAccessUserEC2
cloudxiam.CloudXAccessControlFullAccessUserS3NameD05820FF = FullAccessUserS3
cloudxiam.CloudXAccessControlReadAccessPolicyS3Name94AC6D4A = ReadAccessPolicyS3
cloudxiam.CloudXAccessControlReadAccessUserS3Name4807D309 = ReadAccessUserS3
Stack ARN:

âœ¨  Total time: 109.39s
```

Pic 1 – cluodxiam is successfully deployed

2. DEPLOYMENT VALIDATION

2.1 TAF

For deployment validation a Test Automation Framework in .NET stack was created.

Link to repo with TAF code: https://github.com/Chubaka2612/CloudX

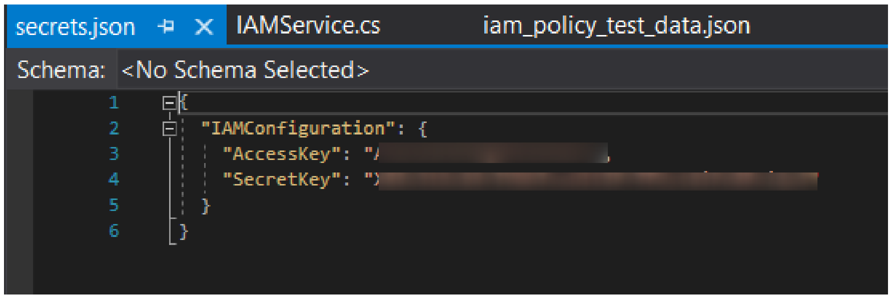
Currently the TAF solution consist of 3 projects:

- CloudX.Auto.Core – contains common classes, i.e. ConfigurationManager, AssertHelper, Common extensions and utils class, different classes for test cases management.
- CloudX.Auto.AWS.Core – included IAMServer class(wrapper on `IAmazonIdentityManagementService`) which allows programmatically interacts with AWS resources.
- CloudX.Auto.Tests – project for test suites and test cases. Currently contains the following test suite:
  - IAM
    - Policy – cover CXQA-IAM-01 scenario
    - Role - CXQA-IAM-02 scenario
    - Group - CXQA-IAM-03 scenario
    - User - CXQA-IAM-04 scenario

And includes 21 tests in total.

TDT approach is used to cover all required scenarios. For each scenario a separate .json file with data is created.

AccessKey and SecretKey are stored in UserSecrets and are not shared publicly.



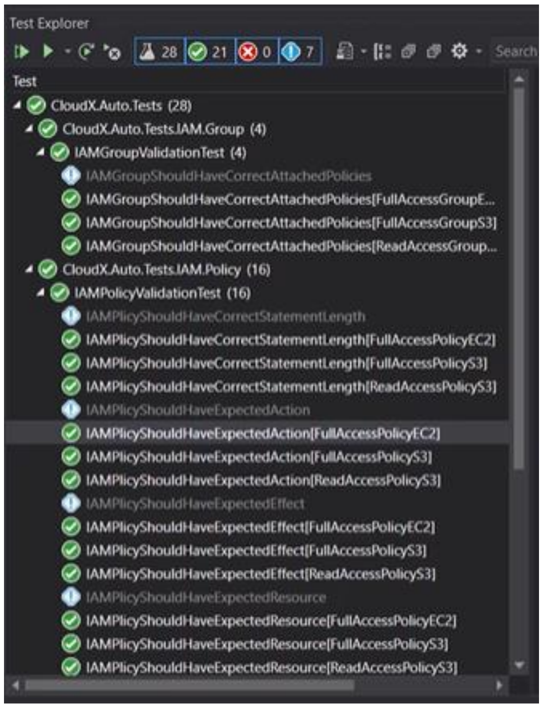
```
{
  "IAMConfiguration": {
    "AccessKey": "[redacted]",
    "SecretKey": "[redacted]"
  }
}
```

Used tools:

1. NUnit – test runner
2. AWS SDK for .NET – API for AWS resource interaction
3. Log3Net – logging library

2.1 Test execution

For cloudxiam v=1 the are no issues found. Results are below

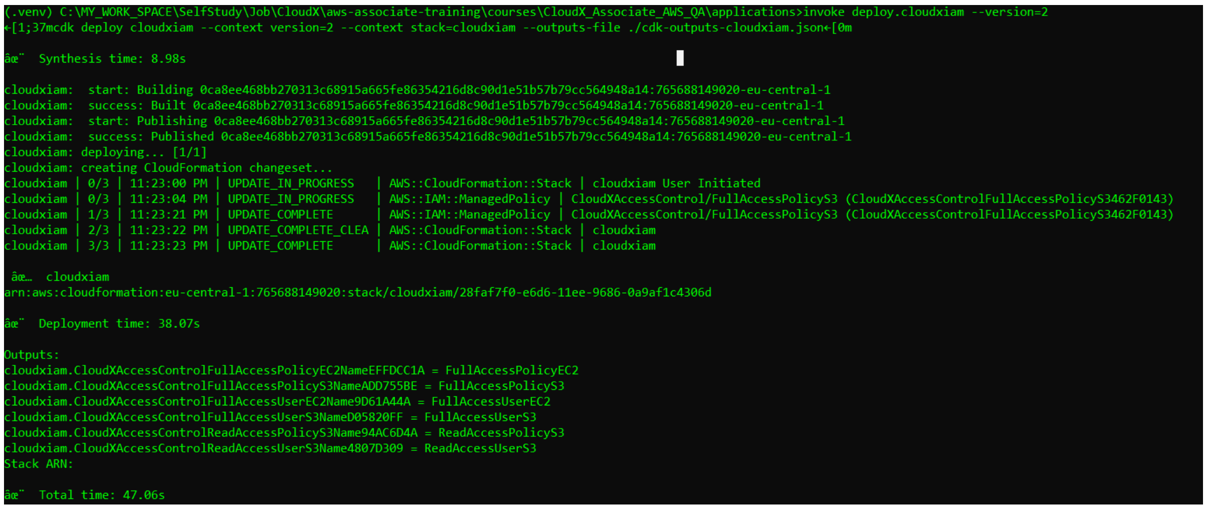


```
CloudX.Auto.Tests (28)
  CloudX.Auto.Tests.IAM.Group (4)
    IAMGroupValidationTest (4)
      IAMGroupShouldHaveCorrectAttachedPolicies
      IAMGroupShouldHaveCorrectAttachedPolicies(FullAccessGroupE...
      IAMGroupShouldHaveCorrectAttachedPolicies(FullAccessGroupS3)
      IAMGroupShouldHaveCorrectAttachedPolicies(ReadAccessGroup...
  CloudX.Auto.Tests.IAM.Policy (16)
    IAMPolicyValidationTest (16)
      IAMPlicyShouldHaveCorrectStatementLength
      IAMPlicyShouldHaveCorrectStatementLength(FullAccessPolicyEC2)
      IAMPlicyShouldHaveCorrectStatementLength(FullAccessPolicyS3)
      IAMPlicyShouldHaveCorrectStatementLength(ReadAccessPolicyS3)
      IAMPlicyShouldHaveExpectedAction
      IAMPlicyShouldHaveExpectedAction(FullAccessPolicyEC2)
      IAMPlicyShouldHaveExpectedAction(FullAccessPolicyS3)
      IAMPlicyShouldHaveExpectedAction(ReadAccessPolicyS3)
      IAMPlicyShouldHaveExpectedEffect
      IAMPlicyShouldHaveExpectedEffect(FullAccessPolicyEC2)
      IAMPlicyShouldHaveExpectedEffect(FullAccessPolicyS3)
      IAMPlicyShouldHaveExpectedEffect(ReadAccessPolicyS3)
      IAMPlicyShouldHaveExpectedResource
      IAMPlicyShouldHaveExpectedResource(FullAccessPolicyEC2)
      IAMPlicyShouldHaveExpectedResource(FullAccessPolicyS3)
      IAMPlicyShouldHaveExpectedResource(ReadAccessPolicyS3)
```

Link to demo

2.1 Regression test execution

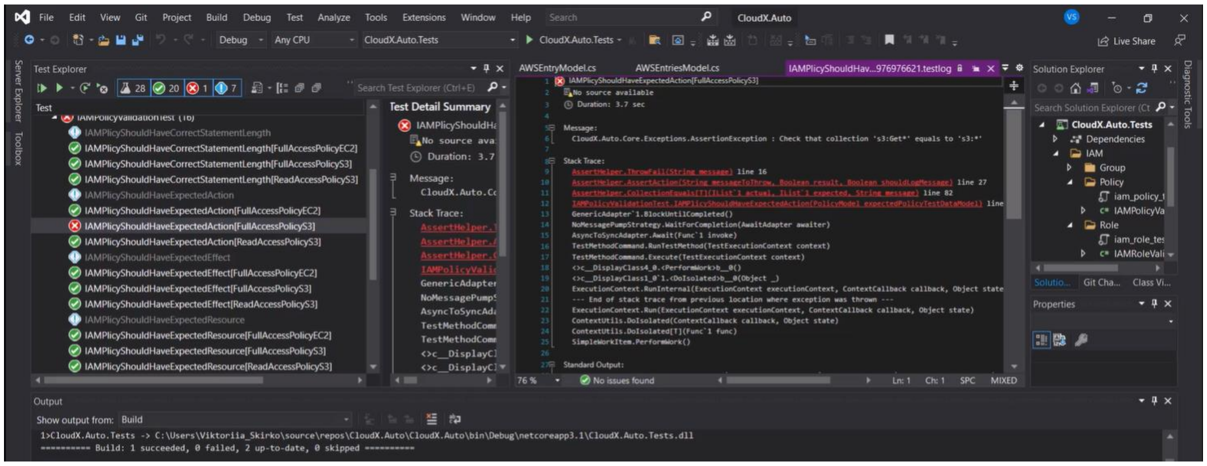
App cloudxiam v=2 is deployed



```
(.venv) C:\MY_WORK_SPACE\SelfStudy\Job\CloudX\aws-associate-training\courses\CloudX_Associate_AWS_QA\applications>invoke deploy.cloudxiam --version=2
←[1;37mcdk deploy cloudxiam --context version=2 --context stack=cloudxiam --outputs-file ./cdk-outputs-cloudxiam.json←[0m

âœ¨  Synthesis time: 8.98s

cloudxiam:  start: Building 0ca8ee468bb270313c68915a665fe86354216d8c90d1e51b57b79cc564948a14:765688149020-eu-central-1
cloudxiam:  success: Built 0ca8ee468bb270313c68915a665fe86354216d8c90d1e51b57b79cc564948a14:765688149020-eu-central-1
cloudxiam:  start: Publishing 0ca8ee468bb270313c68915a665fe86354216d8c90d1e51b57b79cc564948a14:765688149020-eu-central-1
cloudxiam:  success: Published 0ca8ee468bb270313c68915a665fe86354216d8c90d1e51b57b79cc564948a14:765688149020-eu-central-1
cloudxiam: deploying... [1/1]
cloudxiam: creating CloudFormation changeset...
cloudxiam | 0/3 | 11:23:00 PM | UPDATE_IN_PROGRESS   | AWS::CloudFormation::Stack | cloudxiam User Initiated
cloudxiam | 0/3 | 11:23:04 PM | UPDATE_IN_PROGRESS   | AWS::IAM::ManagedPolicy | CloudXAccessControl/FullAccessPolicyS3 (CloudXAccessControlFullAccessPolicyS3462F0143)
cloudxiam | 1/3 | 11:23:21 PM | UPDATE_COMPLETE      | AWS::IAM::ManagedPolicy | CloudXAccessControl/FullAccessPolicyS3 (CloudXAccessControlFullAccessPolicyS3462F0143)
cloudxiam | 2/3 | 11:23:22 PM | UPDATE_COMPLETE_CLEA | AWS::CloudFormation::Stack | cloudxiam
cloudxiam | 3/3 | 11:23:23 PM | UPDATE_COMPLETE      | AWS::CloudFormation::Stack | cloudxiam

 âœ…  cloudxiam
arn:aws:cloudformation:eu-central-1:765688149020:stack/cloudxiam/28faf7f0-e6d6-11ee-9686-0a9af1c4306d

âœ¨  Deployment time: 38.07s

Outputs:
cloudxiam.CloudXAccessControlFullAccessPolicyEC2NameEFFDCC1A = FullAccessPolicyEC2
cloudxiam.CloudXAccessControlFullAccessPolicyS3NameADD755BE = FullAccessPolicyS3
cloudxiam.CloudXAccessControlFullAccessUserEC2Name9D61A44A = FullAccessUserEC2
cloudxiam.CloudXAccessControlFullAccessUserS3NameD05820FF = FullAccessUserS3
cloudxiam.CloudXAccessControlReadAccessPolicyS3Name94AC6D4A = ReadAccessPolicyS3
cloudxiam.CloudXAccessControlReadAccessUserS3Name4807D309 = ReadAccessUserS3
Stack ARN:

âœ¨  Total time: 47.06s
```

Pic 2 – cloudxiam v=2 is deployed

Tests were executed and 1 issues was found.



```
Test
  IAMPolicyValidationTest (16)
    IAMPlicyShouldHaveCorrectStatementLength
    IAMPlicyShouldHaveCorrectStatementLength[FullAccessPolicyEC2]
    IAMPlicyShouldHaveCorrectStatementLength[FullAccessPolicyS3]
    IAMPlicyShouldHaveCorrectStatementLength[ReadAccessPolicyS3]
    IAMPlicyShouldHaveExpectedAction
    IAMPlicyShouldHaveExpectedAction[FullAccessPolicyEC2]
    IAMPlicyShouldHaveExpectedAction[FullAccessPolicyS3]
    IAMPlicyShouldHaveExpectedAction[ReadAccessPolicyS3]
    IAMPlicyShouldHaveExpectedEffect
    IAMPlicyShouldHaveExpectedEffect[FullAccessPolicyEC2]
    IAMPlicyShouldHaveExpectedEffect[FullAccessPolicyS3]
    IAMPlicyShouldHaveExpectedEffect[ReadAccessPolicyS3]
    IAMPlicyShouldHaveExpectedResource
    IAMPlicyShouldHaveExpectedResource[FullAccessPolicyEC2]
    IAMPlicyShouldHaveExpectedResource[FullAccessPolicyS3]
    IAMPlicyShouldHaveExpectedResource[ReadAccessPolicyS3]
```

```
IAMPlicyShouldHaveExpectedAction[FullAccessPolicyS3]
 No source available
 Duration: 3.7 sec

Message:
  CloudX.Auto.Core.Exceptions.AssertionException : Check that collection 's3:Get*' equals to 's3:*'

Stack Trace:
  AssertHelper.ThrowFail(String message) line 16
  AssertHelper.AssertAction(String messageToThrow, Boolean result, Boolean shouldLogMessage) line 27
  AssertHelper.CollectionEquals[T](IList`1 actual, IList`1 expected, String message) line 82
  IAMPolicyValidationTest.IAMPlicyShouldHaveExpectedAction(PolicyModel expectedPolicyTestDataModel) line
  GenericAdapter`1.BlockUntilCompleted()
  NoMessagePumpStrategy.WaitForCompletion(AwaitAdapter awaiter)
  AsyncToSyncAdapter.Await(Func`1 invoke)
  TestMethodCommand.RunTestMethod(TestExecutionContext context)
  TestMethodCommand.Execute(TestExecutionContext context)
  <>c__DisplayClass4_0.<PerformWork>b__0()
  <>c__DisplayClass1_0`1.<DoIsolated>b__0(Object _)
  ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object state
  --- End of stack trace from previous location where exception was thrown ---
  ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state)
  ContextUtils.DoIsolated(ContextCallback callback, Object state)
  ContextUtils.DoIsolated[T](Func`1 func)
  SimpleWorkItem.PerformWork()

Standard Output:
```

```
Output
Show output from: Build
1>CloudX.Auto.Tests -> C:\Users\Viktoriia_Skirko\source\repos\CloudX.Auto\CloudX.Auto\bin\Debug\netcoreapp3.1\CloudX.Auto.Tests.dll
========== Build: 1 succeeded, 0 failed, 2 up-to-date, 0 skipped ==========
```

Link to demo

2.3 Bug report

Summary: Policy ‘FullAccessPolicyS3’ has incorrect ‘Actions Allowed’ value

Type:	Bug
Priority:	Major
Affects Version:	2
Component:	IAM
Labels:	Regression

Steps to reproduce:

1. Obtain policy ‘FullAccessPolicyS3’ (use any convenient way: CLI, IAM/Policies console)
2. Observe ‘Statement.Action’ value.

Actual results: Statement.Action = s3:Get*

Expected results: Statement.Action = s3:*