CloudX Associate: AWS for Testers

EC2

Error! Unknown document property name.

TASK 3 – EC2

1. DEPLOY THE IAM APPLICATION

Deploy the **cloudxinfo** CDK stack: deployment instructions.

2. DEPLOYMENT VALIDATION

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

CXQA-EC2-01: 2 application instances should be deployed: public and private

CXQA-EC2-02: Each EC2 instance should have the following configuration:

- Instance type: t2.micro
- Instance tags: cloudx: qa
- Root block device size: 8 GB
- Instance OS: Amazon Linux 2
- The public instance should have public IP assigned
- The private instance should not have public IP assigned

CXQA-EC2-03: The security groups' configuration:

- The public instance should be accessible from the internet by SSH (port 22) and HTTP (port 80) only
- The private instance should be accessible only from the public instance by SSH and HTTP protocols only
- Both private and public instances should have access to the internet

2.1 Testing Tools:

- AWS Console
- AWS CLI
- AWS SDK (for automated tests).
- Application Swagger OpenAPI documentation
- Postman / CURL
- SSH client

3. APPLICATION FUNCTIONAL VALIDATION

Create a manual/automated test suite that covers the following requirements, both for public and private instances.

CXQA-EC2-04: Application API endpoint should respond with the correct instance information from EC2 metadata:

- aws region
- availability zone
- instance private IP

3.1 Testing Tools:

- Application Swagger OpenAPI documentation
- Postman / CURL
- SSH client

Execute test cases and verify that requirements are met.

4 REGRESSION TESTING

- Deploy **version 2** of the **cloudxinfo** application deployment instructions.
- Execute deployment and functional validation test suites against new application version deployment.
- Create bug/root cause report for the found regression issues.

5. ENVIRONMENT CLEAN-UP

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

6. 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

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

AWS EC2 THEORY

What hardware is offered by EC2?

- General Purpose Instances: These instances provide a balance of compute, memory, and networking resources, making them suitable for a wide range of applications. Examples include the t3, m5, and m6g instance families.
- Compute Optimized Instances: These instances are optimized for compute-intensive workloads that require high-performance processors. Examples include the c5 and c6g instance families.
- Memory Optimized Instances: These instances are designed for memory-intensive workloads, such as large-scale databases, in-memory caching, and real-time analytics. Examples include the r5 and r6g instance families.
- Storage Optimized Instances: These instances are optimized for high-performance storage, making them suitable for data-intensive workloads that require fast access to local storage. Examples include the i3 and i3en instance families.
- Accelerated Computing Instances: These instances are equipped with specialized hardware, such as GPUs or FPGAs, to accelerate specific types of workloads, such as machine learning, graphics rendering, or video encoding. Examples include the p3, g4, and f1 instance families.
- Burstable Performance Instances: These instances provide baseline performance with the ability to burst above the baseline when needed. They are suitable for workloads that have variable compute demands. Examples include the t3 and t4g instance families.
- Dedicated Hosts: These instances allow you to run EC2 instances on dedicated physical servers, providing additional isolation and compliance options.

What provisioning/billing options are available with EC2?

- On-Demand Instances: With On-Demand Instances, you pay for compute capacity by the hour or by the second, depending on the instance type and region. This option allows you to launch instances as needed without any long-term commitments or upfront payments. You only pay for the compute capacity you use, with no minimum commitments.
- Reserved Instances (RIs): Reserved Instances allow you to reserve compute capacity for a specific duration (1 year or 3 years) in exchange for a significant discount compared to On-Demand rates. RIs provide a lower cost per hour compared to On-Demand Instances, making them suitable for predictable workloads with steady-state usage. RIs offer three payment options: No Upfront, Partial Upfront, and All Upfront.
- Spot Instances: Spot Instances allow you to bid for unused EC2 capacity, potentially offering significant cost savings compared to On-Demand rates. You specify the maximum price you are willing to pay per hour, and if your bid exceeds the current Spot price, your instances will run. Spot Instances are ideal for workloads with flexible start and end times or those that can tolerate interruptions.
- Dedicated Hosts: With Dedicated Hosts, you have the option to run EC2 instances on dedicated physical servers, providing additional isolation and compliance options. Dedicated Hosts can be purchased On-Demand or as Reserved Instances, offering flexibility and control over your infrastructure.
- Savings Plans: Savings Plans provide flexible pricing options and significant discounts (up to 72%) compared to On-Demand rates. Savings Plans offer a commitment to a consistent amount of usage (measured in dollars per hour) for a 1-year or 3-year term. They provide savings across EC2 instance usage, regardless of instance family, size, region, or operating system.
- Capacity Reservations: Capacity Reservations allow you to reserve capacity in specific Availability Zones, ensuring that you have the compute capacity you need when you need it. Capacity Reservations can be purchased On-Demand or as Reserved Capacity, offering flexibility and control over your infrastructure.

What is EBS?

Amazon Elastic Block Store (Amazon EBS) is a block storage service provided by Amazon Web Services (AWS) that offers persistent block-level storage volumes for use with Amazon EC2 instances. EBS volumes are designed to be highly available and reliable, providing durable storage for data that requires frequent access or must persist beyond the life of an EC2 instance.

Key features of Amazon EBS include:

- Persistent Storage: EBS volumes provide persistent block storage that persists independently from the EC2 instance lifecycle. This means that data stored on EBS volumes remains intact even if the associated EC2 instance is stopped, terminated, or fails.
- Elasticity: EBS volumes can be dynamically resized, allowing you to scale storage capacity up or down based on your application's needs without downtime. You can also create multiple volumes and attach them to a single EC2 instance to meet performance and capacity requirements.
- High Performance: EBS volumes offer low-latency, high-throughput performance, making them suitable for a wide range of workloads, including databases, file systems, and transactional applications. You can choose from different volume types optimized for different use cases, such as General Purpose SSD (gp2), Provisioned IOPS SSD (io1), Throughput Optimized HDD (st1), and Cold HDD (sc1).
- Snapshot and Backup: EBS volumes support point-in-time snapshots, allowing you to create backups of your data and restore volumes from snapshots as needed. Snapshots are incremental, meaning that only the changed blocks are stored, which helps reduce storage costs and backup times.
- Encryption: EBS volumes support encryption at rest, allowing you to encrypt data stored on volumes using AWS Key Management Service (KMS) keys. Encryption helps protect sensitive data and meet compliance requirements.
- Availability and Durability: EBS volumes are designed for high availability and durability, with data replicated within the same Availability Zone to protect against component failures. You can also create Multi-AZ volumes for additional redundancy and fault tolerance.

What types of volumes are offered by EC2?

Amazon Elastic Block Store (Amazon EBS) offers several types of volumes optimized for different performance, cost, and durability requirements. Each volume type is designed to meet specific use cases and workload demands. Here are the main types of EBS volumes offered by Amazon EC2:

General Purpose SSD (gp3 and gp2):

- *Use Cases*: General-purpose workloads, boot volumes, development and test environments, small to medium databases, low-latency interactive applications.
- *Performance*: Offers a balance of price and performance with low-latency and consistent IOPS performance.
- *Features:* Supports bursting for periods of peak usage (gp3 only), scalable performance with adjustable baseline IOPS (gp3 only), and up to 16 TiB volume size.

Provisioned IOPS SSD (io2 and io1):

- *Use Cases:* High-performance databases, I/O-intensive applications, mission-critical workloads that require consistent and predictable performance.
- *Performance:* Provides high and consistent IOPS performance with low-latency access to data.
- *Features:* Allows you to provision the desired number of IOPS (input/output operations per second), adjustable volume size, and up to 64 TiB volume size.

Throughput Optimized HDD (st1):

- *Use Cases:* Big data, data warehouses, log processing, data lakes, streaming workloads that require high throughput for large datasets.
- *Performance:* Optimized for streaming workloads with high throughput and low-cost storage.
- *Features:* Provides high throughput performance for sequential read and write operations, supports cost-effective storage for large datasets, and up to 64 TiB volume size.

Cold HDD (sc1):

- *Use Cases:* Infrequently accessed data, archival storage, backup and restore, low-cost storage for data that is accessed less frequently.
- *Performance:* Offers low-cost storage with the ability to store large volumes of data at a lower price point.
- *Features:* Provides cost-effective storage for infrequently accessed data, supports sequential read and write operations, and up to 16 TiB volume size.

What is the difference between AMI and snapshot in terms of EC2?

In Amazon EC2, both Amazon Machine Images (AMIs) and snapshots are used for creating and managing instances, but they serve different purposes:

Amazon Machine Image (AMI):

- An AMI is a template that contains the configuration (such as the operating system, application server, and applications) required to launch an EC2 instance.
- AMIs are used to launch new instances or replace existing instances with a known configuration.
- AMIs can be either public (provided by AWS or the community) or private (customized and maintained by you).
- AMIs can include one or more EBS snapshots as part of their configuration.
- When you launch an instance from an AMI, AWS provisions the necessary resources based on the AMI's configuration, including creating new EBS volumes based on the AMI's snapshot(s) if needed.
- You can create custom AMIs from existing instances, modify them as needed, and share them with other AWS accounts.

Snapshot:

- A snapshot is a point-in-time copy of an Amazon EBS volume. It contains the data stored on the volume at the time the snapshot was created.
- Snapshots are used for data backup, replication, and disaster recovery purposes. They allow you to create backups of your EBS volumes and restore them as needed.
- Snapshots are incremental, meaning that only the blocks that have changed since the last snapshot are stored. This helps reduce storage costs and backup times.
- You can create snapshots manually or automatically using scheduled snapshots or lifecycle policies.
- Snapshots can be used to create new EBS volumes or restore existing volumes to a previous state.

In summary, while both AMIs and snapshots are used for managing instances and data in Amazon EC2, AMIs are used to create instance templates with predefined configurations, while snapshots are used for backing up and restoring data stored on EBS volumes. AMIs can include snapshots as part of their configuration, but they also contain additional metadata and settings required to launch an instance.

What are regions and availability zones in AWS?

Region:

- A region is a physical location around the world where AWS has data centers (also known as Availability Zones). Each region is a separate geographic area and consists of multiple isolated and physically distinct data centers.
- AWS operates multiple regions globally to provide customers with options for deploying their applications and services closer to their end-users, complying with data residency requirements, and achieving redundancy and disaster recovery.
- Each AWS region is identified by a unique name (e.g., us-east-1, eu-west-1, ap-southeast-2) and is completely independent of other regions. Resources created in one region are isolated from resources in other regions.
- AWS services may vary by region, with some services available in all regions, while others are available only in specific regions.

Availability Zone (AZ):

- An availability zone is a distinct location within a region that is engineered to be isolated from failures in other availability zones. Each availability zone typically consists of one or more data centers located nearby, but with independent power, cooling, and networking infrastructure.
- Availability zones are interconnected with high-speed, low-latency networking, allowing you to deploy redundant and highly available architectures across multiple zones within the same region.
- AWS customers can deploy their applications and services across multiple availability zones. By distributing workload across multiple zones, applications can continue to operate even if one or more availability zones experience disruptions or failures.
- Each AWS region contains multiple availability zones, usually three or more. The number of availability zones may vary by region and can be expanded over time to meet increasing demand.

In summary, AWS regions are isolated geographic areas with multiple availability zones, each of which is a separate data center with its own infrastructure and facilities.

How is it possible to install/configure software on a EC2 instance?

1. Launch an EC2 Instance:
First, you need to launch an EC2 instance with the desired operating system (e.g., Amazon Linux, Ubuntu, Windows Server) and instance type (e.g., t2.micro, m5.large).
You can do this through the AWS Management Console, AWS CLI, or AWS SDKs.
2. Connect to the EC2 Instance:
Once the EC2 instance is running, you need to connect to it using SSH (for Linux instances) or Remote Desktop Protocol (RDP) (for Windows instances).
For Linux instances, you typically use a terminal and SSH to connect:

```
ssh -i /path/to/key.pem ec2-user@public-ip-address
```

For Windows instances, you use an RDP client to connect using the public IP address or public DNS of the instance.
3. Install Required Software:
Install the software packages and dependencies required for your application or workload.
You can use package managers like yum (for Amazon Linux, CentOS) or apt (for Ubuntu) to install software packages.
For Windows instances, you can download and install software manually or use PowerShell scripts or package managers.
4. Test the Setup:
After installing and configuring the software, test that everything is working as expected.
Run your application or service and verify that it functions correctly.

What keys are created for each EC2 instance? What for?

When you launch an Amazon EC2 instance, two types of keys are created:
Key Pair (SSH Key Pair):

- A key pair, also known as an SSH key pair, is generated when you create an EC2 instance. It consists of a public key and a private key.
- The public key is stored on the EC2 instance, and the private key is downloaded to your local machine during instance creation.
- The key pair is used for SSH (Secure Shell) authentication, allowing you to securely connect to the EC2 instance using SSH.
- You use the private key to authenticate yourself when connecting to the instance via SSH. The public key is stored on the instance and is used to verify your identity.

AWS Identity and Access Management (IAM) Role (Optional):

- In addition to the key pair, you can also assign an IAM role to an EC2 instance when you launch it.
- An IAM role is an AWS identity with permissions attached to it. It allows the EC2 instance to perform actions on AWS services based on the permissions granted to the role.
- IAM roles are typically used for granting permissions to EC2 instances to access other AWS services, such as S3, DynamoDB, or RDS, without requiring access keys to be stored on the instance.
- These keys are created to ensure secure access to your EC2 instances and to provide permissions for accessing AWS resources. The key pair allows you to securely connect to your EC2 instances via SSH, while IAM roles provide permissions for accessing other AWS services from the instance. It's important to manage and secure these keys properly to prevent unauthorized access to your instances and AWS resources.

What happens to EC2 instances when they are stopped and started vs re-started?

Stopped and Started:

- When you stop an EC2 instance, its state changes to "stopped," and the instance is no longer running. However, the instance's metadata, configuration, and associated resources (such as EBS volumes and Elastic IP addresses) remain intact.
- When you start a stopped EC2 instance, it is launched again with the same configuration, including the same instance ID, private and public IP addresses, security groups, and attached storage volumes.
- Stopping and starting an EC2 instance involves a complete reboot of the instance, which may result in a new underlying physical host for the instance. As a result, the instance may be assigned a new internal IP address (if it's using DHCP) but retains its external (public) IP address and other configuration settings.
- The instance's EBS volumes maintain their data, and any data stored in the instance's instance store is lost when the instance is stopped.
- Stopping and starting an instance may take longer than simply restarting it, as the instance needs to be re-provisioned on a new underlying host.

Reboot (Re-start):

- When you reboot (re-start) an EC2 instance, it undergoes a soft reboot, where the operating system is restarted, but the underlying physical host and instance configuration remain the same.
- Rebooting an instance maintains its instance ID, private and public IP addresses, security groups, and attached storage volumes. It does not change the instance's metadata or configuration.
- The reboot process is faster than stopping and starting an instance because it doesn't involve re-provisioning the instance on a new host.
- Rebooting an instance does not affect the data stored on its EBS volumes or instance store (ephemeral storage).

In summary, stopping and starting an EC2 instance involves a complete shutdown and relaunch of the instance, potentially resulting in changes to the instance's internal IP address and longer downtime. Rebooting an instance, on the other hand, simply restarts the operating system without changing the instance's configuration or affecting its attached volumes.

What is the difference between IAM roles and EC2 (VPC) security groups?

IAM roles are used to grant temporary permissions to entities accessing AWS resources, while EC2 security groups are used to control traffic to and from EC2 instances at the network interface level. IAM roles are used for controlling access to AWS services and resources, while security groups are used for controlling network traffic to EC2 instances.

Is it possible to decrease the size of an existing EBS volume?

Yes, it is possible to decrease the size of an existing Amazon EBS volume, but it involves several steps and considerations:

Backup Data: Before resizing the volume, it's crucial to back up any important data stored on the volume to avoid data loss in case of errors during the resizing process.
Verify Free Space: Ensure that there is enough free space on the volume to accommodate the new size. If the volume is nearly full, you may need to delete unnecessary files or resize partitions to free up space.
Resize Filesystem (if applicable): If the EBS volume contains a filesystem (such as ext4, NTFS, or XFS), you may need to resize the filesystem to match the new size of the volume. This step is necessary to utilize the full capacity of the resized volume.
Resize EBS Volume: Once the filesystem is resized (if applicable), you can resize the EBS volume using the AWS Management Console, AWS CLI, or SDKs. Keep in mind that you can only decrease the size of a volume if it is not the root volume of a running instance, and you cannot reduce the size below the amount of data currently stored on the volume.
Verify Changes: After resizing the volume, verify that the changes have taken effect and that the filesystem (if applicable) reflects the new size.
It's important to note that decreasing the size of an EBS volume carries some risks, and it's recommended to thoroughly review the documentation and consider the implications before proceeding. Additionally, always ensure that you have recent backups of your data before making any changes to your EBS volumes.

Is it possible to reuse a EBS volume for multiple instances?

No, it's not possible to attach an EBS volume to multiple instances simultaneously. Amazon EBS volumes are designed to be attached to a single EC2 instance at a time. When you attach an EBS volume to an instance, it becomes exclusive to that instance, and only that instance can access and use the volume.
However, you can detach an EBS volume from one instance and then attach it to another instance if needed. This process allows you to transfer the volume from one instance to another, effectively "reusing" the volume across multiple instances sequentially.

How is it possible to get such metadata as current region/AZ from within a running EC2 instance?

Retrieve Current Region:
You can obtain the current region using the region endpoint of the instance metadata service. This endpoint returns the AWS region in which the instance is running.
To retrieve the current region, make an HTTP request to the following URL from within the instance:

```
http://{ip}/latest/meta-data/placement/availability-zone
```

The response from this request contains the availability zone in which the instance is running. The format of the availability zone includes the region code (e.g., us-east-1a), allowing you to extract the region.
Retrieve Current Availability Zone (AZ):
Similarly, you can obtain the current availability zone using the availability-zone endpoint of the instance metadata service. This endpoint returns the availability zone in which the instance is running.
To retrieve the current availability zone, make an HTTP request to the following URL from within the instance:

```
http://{ip}/latest/meta-data/placement/availability-zone
```

The response from this request contains the availability zone in which the instance is running.
Here's an example of how you can retrieve the region and availability zone using the AWS Command Line Interface (CLI) from within the instance:

```
# Retrieve current region
aws  ec2  describe-availability-zones  --query  'AvailabilityZones[0].[RegionName]'  --
```

```
output text

# Retrieve current availability zone
curl -s http://169.254.169.254/latest/meta-data/placem.
```

What are the available elastic load balancer types? What is the key difference between them?

Classic Load Balancer (CLB):

- The original load balancer offering from AWS, now referred to as Classic Load Balancer, provides basic load balancing across multiple EC2 instances and operates at both the connection and application layers.
- Supports HTTP, HTTPS, TCP, and SSL/TLS protocols.
- Key Difference: Classic Load Balancer is the legacy load balancer type and has limited features compared to the newer load balancer types.

Application Load Balancer (ALB):

- Application Load Balancer operates at the application layer and is designed to route traffic to different targets based on content of the request (e.g., URL path, host, HTTP headers).
- Supports HTTP, HTTPS, and WebSocket protocols.
- Supports content-based routing, host-based routing, path-based routing, and integration with AWS services like AWS WAF, AWS Lambda, and Amazon ECS.
- Key Difference: ALB provides more advanced routing capabilities and is suitable for modern application architectures with microservices and container-based workloads.

Network Load Balancer (NLB):

- Network Load Balancer operates at the connection and transport layer (Layer 4) and is designed to handle high volumes of traffic with low latency and high throughput.
- Supports TCP, TLS, and UDP protocols.
- Distributes traffic across targets based on IP addresses and ports specified in the listener configuration.
- Key Difference: NLB provides ultra-high performance and is suitable for latency-sensitive, high-throughput applications, such as gaming, media streaming, and IoT workloads.
- In summary, the key difference between these ELB types lies in their functionality and capabilities. Classic Load Balancer is the legacy offering with basic features, while Application Load Balancer (ALB) and Network Load Balancer (NLB) offer more advanced routing, performance, and integration options suitable for modern application architectures and high-performance workloads.

What are the key events in EC2 instance lifecycle?

Instance Launch:

- The instance is launched from an Amazon Machine Image (AMI) using the AWS Management Console, CLI, SDKs, or API.
- During launch, you specify parameters such as instance type, AMI, network settings, storage options, security groups, and IAM role.

Instance Startup:

- Once launched, the instance initializes and boots up the selected operating system.
- Any user data scripts specified during instance launch are executed.
- The instance transitions from the pending state to the running state when it is ready to accept incoming connections.

Instance Running:

- The instance is in the running state and can accept incoming traffic and perform its designated tasks.
- Applications and services running on the instance are operational, and users can access the instance via SSH, RDP, or other protocols.

Instance Maintenance:

- Periodically, AWS performs maintenance activities on the underlying infrastructure, such as hardware maintenance, security updates, and hypervisor upgrades.
- During maintenance events, AWS may perform automatic instance reboots or migrations to ensure the health and reliability of the infrastructure.

Instance Stop/Start:

- The instance can be stopped and started manually by the user or programmatically using the AWS Management Console, CLI, or API.
- Stopping an instance halts the instance's running processes and preserves its data and configuration. Starting the instance resumes its operation from the stopped state.

Instance Termination:

- The instance is terminated either manually by the user or automatically under certain conditions (e.g., instance failure, termination of associated Auto Scaling group).
- Termination removes the instance permanently, including its associated data, public and private IP addresses, and any attached EBS volumes.
- It's important to backup any important data stored on the instance before termination to prevent data loss.

How does load balancing works? What are its core component?

Load balancers distribute incoming traffic across multiple servers or resources to ensure optimal resource utilization, high availability, and scalability. Here's an overview of how load balancing works and its core components:

Client Request:

The process begins when a client sends a request to access an application, website, or service hosted on the server infrastructure.

Load Balancer:

The load balancer sits between the clients and the backend servers or resources, acting as a reverse proxy. It receives incoming requests from clients and forwards them to the appropriate backend servers based on predefined routing algorithms.

Routing Algorithm:

- Load balancers use routing algorithms to determine how incoming requests are distributed across backend servers. Common routing algorithms include:
- Round Robin: Distributes requests evenly across backend servers in a cyclic manner.
- Least Connections: Routes requests to the server with the fewest active connections, aiming to balance the load more evenly.
- Weighted Round Robin: Assigns a weight to each server to reflect its capacity, allowing more powerful servers to handle a larger share of the load.

Health Checks:

Load balancers periodically perform health checks on backend servers to ensure they are available and healthy. If a server fails the health check (e.g., due to high CPU usage, network issues), the load balancer stops routing traffic to it until it recovers.

Backend Servers:

Backend servers are the destination of requests forwarded by the load balancer. They host the applications, websites, or services being accessed by clients.

Backend servers can be physical servers, virtual machines, containers, or serverless functions, depending on the architecture and requirements of the application.

Response:

Once a backend server processes a request and generates a response, it sends the response back to the load balancer. The load balancer then forwards the response to the client that initiated the request.

Scaling:

Load balancers facilitate horizontal scaling by enabling the addition or removal of backend servers dynamically based on demand. As traffic increases, additional servers can be added to handle the load, and as traffic decreases, servers can be removed to reduce costs.

How is it possible to grant a EC2 instance permissions to access certain AWS resources like S3?

To grant an Amazon EC2 instance permissions to access AWS resources like Amazon S3, you can use IAM roles. IAM roles are AWS identities with permissions attached to them, allowing you to delegate permissions to EC2 instances or other entities securely without needing to embed long-term credentials (such as access keys) directly into your application code or configuration files.
Here's how you can grant an EC2 instance permissions to access Amazon S3 using IAM roles:
Create an IAM Role:
Start by creating an IAM role with the necessary permissions to access Amazon S3. You can define the permissions using an IAM policy that specifies the actions, resources, and conditions allowed for accessing S3.
Include the required S3 permissions in the IAM policy, such as *s3:GetObject, s3:PutObject, s3:ListBucket*, etc., depending on the specific actions your application needs to perform.
Attach the IAM Role to the EC2 Instance:
Once you've created the IAM role, attach it to the EC2 instance that needs access to Amazon S3. You can do this during instance launch or attach the role to an existing instance.
When launching a new EC2 instance, specify the IAM role in the instance configuration settings. If attaching the role to an existing instance, you can do so through the AWS Management Console, CLI, or SDKs.
Access S3 from the EC2 Instance:
With the IAM role attached to the EC2 instance, the instance can now access Amazon S3 using the AWS SDKs, CLI, or APIs without needing access keys or other credentials explicitly configured.
The IAM role grants temporary security credentials to the EC2 instance, allowing it to authenticate and access S3 resources securely.

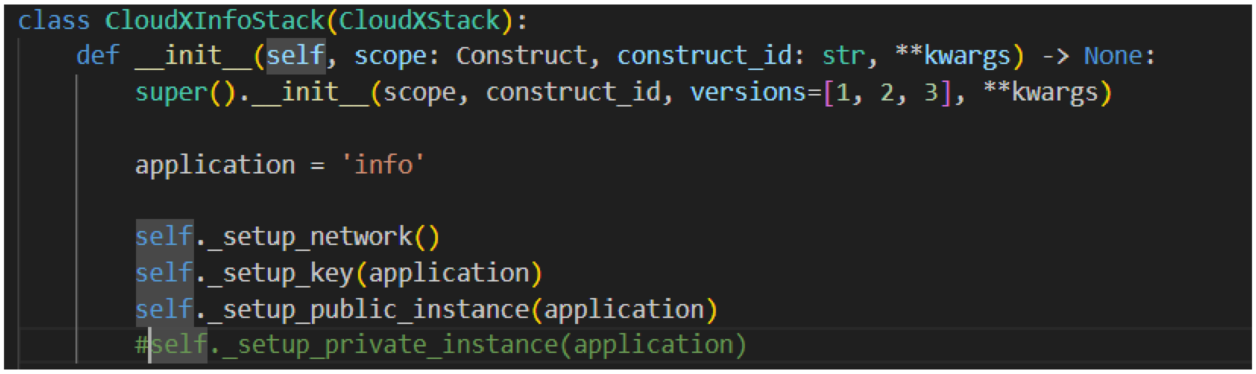
TASK IMPLEMENTATION

1. CLOUDXIAM DEPLOYMENT

Due to issue with launching 2 ec2 instances simultaneously the deployment was executed in 2 stages: 1st stage – deployment on the public instance, 2nd – deployment on the private instance.

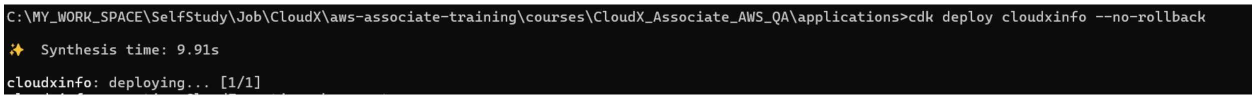
PART 1: PUBLIC INSTANCE

Deployment on the public instance



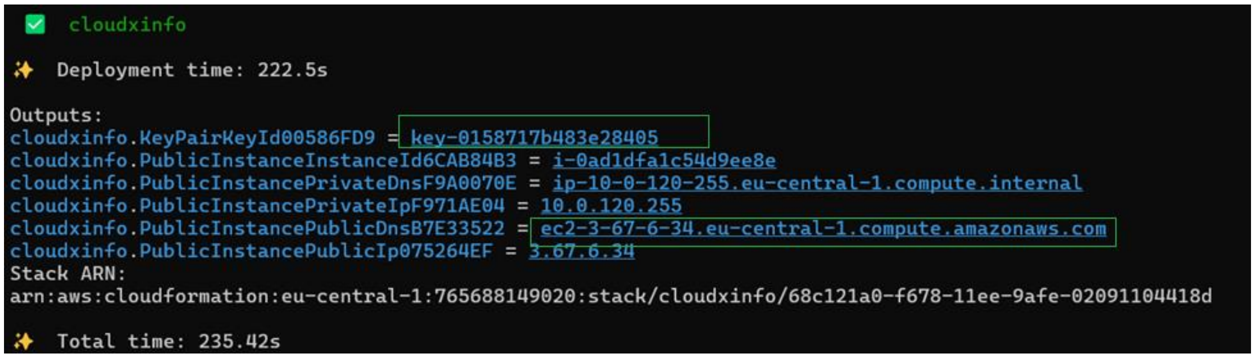
```
class CloudXInfoStack(CloudXStack):
    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, versions=[1, 2, 3], **kwargs)

        application = 'info'

        self._setup_network()
        self._setup_key(application)
        self._setup_public_instance(application)
        #self._setup_private_instance(application)
```

Pic 1 – disable setup of private instance



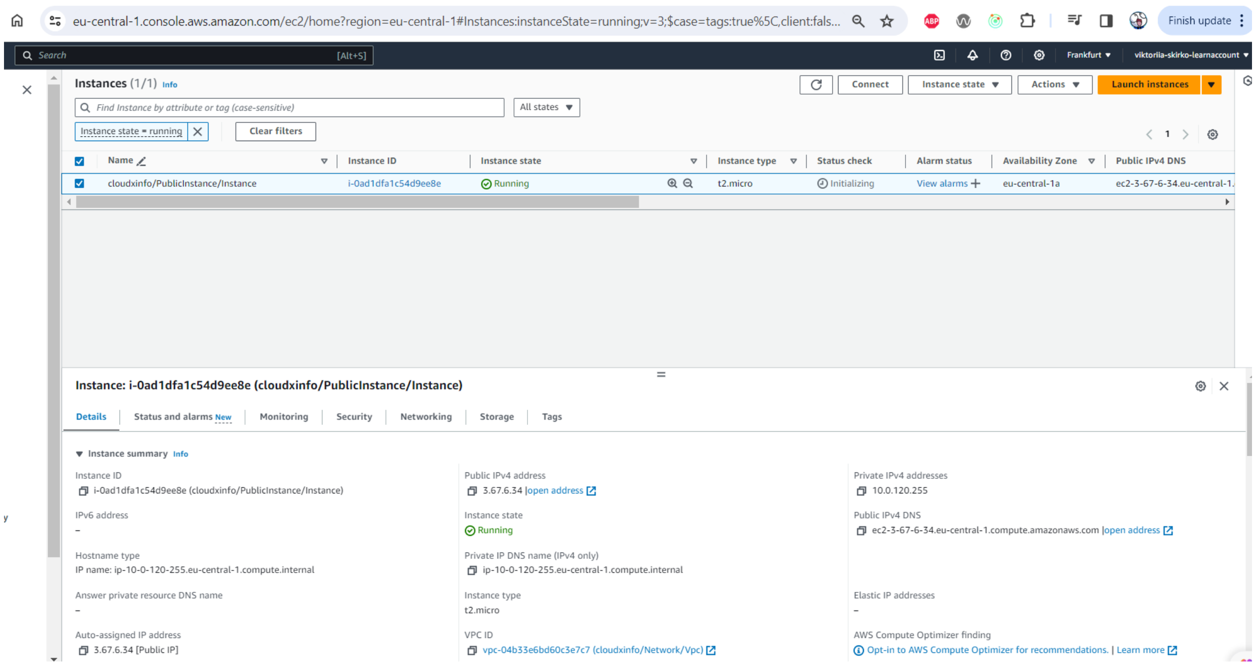
```
C:\MY_WORK_SPACE\SelfStudy\Job\CloudX\aws-associate-training\courses\CloudX_Associate_AWS_QA\applications>cdk deploy cloudxinfo --no-rollback

✨  Synthesis time: 9.91s

cloudxinfo: deploying... [1/1]
```

Pic 2 – start deploying of cloudxinfo stack



```
✅  cloudxinfo

✨  Deployment time: 222.5s

Outputs:
cloudxinfo.KeyPairKeyId00586FD9 = key-0158717b483e28405
cloudxinfo.PublicInstanceInstanceId6CAB84B3 = i-0ad1dfa1c54d9ee8e
cloudxinfo.PublicInstancePrivateDnsF9A0070E = ip-10-0-120-255.eu-central-1.compute.internal
cloudxinfo.PublicInstancePrivateIpF971AE04 = 10.0.120.255
cloudxinfo.PublicInstancePublicDnsB7E33522 = ec2-3-67-6-34.eu-central-1.compute.amazonaws.com
cloudxinfo.PublicInstancePublicIp075264EF = 3.67.6.34
Stack ARN:
arn:aws:cloudformation:eu-central-1:765688149020:stack/cloudxinfo/68c121a0-f678-11ee-9afe-02091104418d

✨  Total time: 235.42s
```

Pic 3 – cloudxinfo was deployed on public instance (no outcomes for private instance)



eu-central-1.console.aws.amazon.com/ec2/home?region=eu-central-1#Instances:instanceState=running;v=3;$case=tags:true%5C,client:fals...

Frankfurt | viktoriia-skirko-learnaccount

Instances (1/1)

Filter: Instance state = running

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
cloudxinfo/PublicInstance/Instance	i-0ad1dfa1c54d9ee8e	Running	t2.micro	Initializing	View alarms	eu-central-1a	ec2-3-67-6-34.eu-central-1.

Instance: i-0ad1dfa1c54d9ee8e (cloudxinfo/PublicInstance/Instance)

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

Instance summary

Instance ID: i-0ad1dfa1c54d9ee8e (cloudxinfo/PublicInstance/Instance)	Public IPv4 address: 3.67.6.34 |open address	Private IPv4 addresses: 10.0.120.255
IPv6 address: –	Instance state: Running	Public IPv4 DNS: ec2-3-67-6-34.eu-central-1.compute.amazonaws.com |open address
Hostname type: IP name: ip-10-0-120-255.eu-central-1.compute.internal	Private IP DNS name (IPv4 only): ip-10-0-120-255.eu-central-1.compute.internal	
Answer private resource DNS name: –	Instance type: t2.micro	Elastic IP addresses: –
Auto-assigned IP address: 3.67.6.34 [Public IP]	VPC ID: vpc-04b33e6bd60c3e7c7 (cloudxinfo/Network/Vpc)	AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. | Learn more

Pic 4 – public instance is running

2. DEPLOYMENT VALIDATION

2.1 TAF

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

Link to repo with task implemented: https://github.com/Chubaka2612/CloudX/compare/master...task3_ec2

The TAF solution was extended:

- CloudX.Auto.AWS.Core – add EC2Server class (wrapper on AmazonEC2Client) which allows programmatically interacts with EC2 resources.
- CloudX.Auto.Tests – The following test suite was added:
  - EC2
    - Deployment – covers CXQA-EC2-01, CXQA-EC2-02, CXQA-EC2-03 scenarios;
    - Functionality - covers CXQA-EC2-04 scenario.

In total includes 8 tests in total.

TDT approach is used to cover all required scenarios.

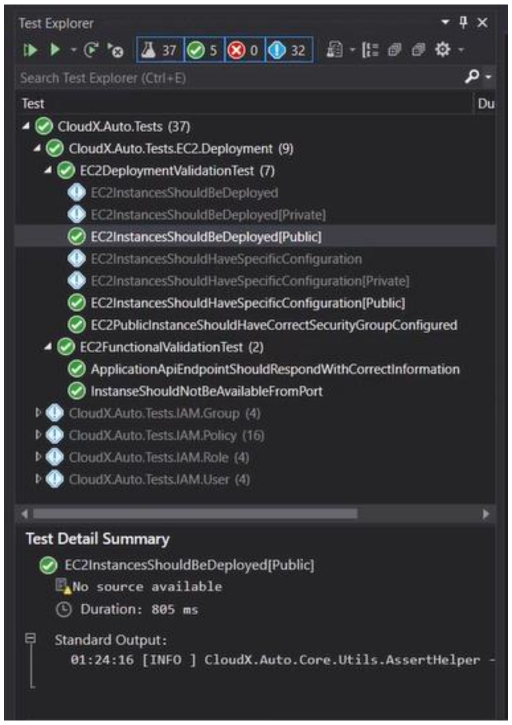
Used tools:

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

4. RestSharp – library for API testing

2.1 [Public Instance] Deployment and Functional test execution

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



```
Test Explorer
37  5  0  32
Search Test Explorer (Ctrl+E)
Test                                                              Du
◢ CloudX.Auto.Tests (37)
  ◢ CloudX.Auto.Tests.EC2.Deployment (9)
    ◢ EC2DeploymentValidationTest (7)
        EC2InstancesShouldBeDeployed
        EC2InstancesShouldBeDeployed[Private]
        EC2InstancesShouldBeDeployed[Public]
        EC2InstancesShouldHaveSpecificConfiguration
        EC2InstancesShouldHaveSpecificConfiguration[Private]
        EC2InstancesShouldHaveSpecificConfiguration[Public]
        EC2PublicInstanceShouldHaveCorrectSecurityGroupConfigured
    ◢ EC2FunctionalValidationTest (2)
        ApplicationApiEndpointShouldRespondWithCorrectInformation
        InstanseShouldNotBeAvailableFromPort
  ▷ CloudX.Auto.Tests.IAM.Group (4)
  ▷ CloudX.Auto.Tests.IAM.Policy (16)
  ▷ CloudX.Auto.Tests.IAM.Role (4)
  ▷ CloudX.Auto.Tests.IAM.User (4)

Test Detail Summary
  EC2InstancesShouldBeDeployed[Public]
    No source available
    Duration: 805 ms

  Standard Output:
    01:24:16 [INFO ] CloudX.Auto.Core.Utils.AssertHelper -
```

Pic 5 – test results execution for **public** instance only

Link to test execution on public instance: demo

2.2 [Public instance] SSH connect to instance

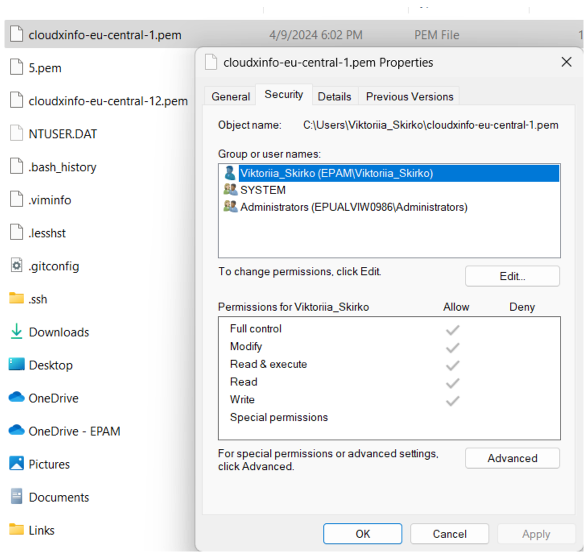
1. Execute

```
aws ssm get-parameter --name "/ec2/keypair/key-0158717b483e28405" --with-decryption -
-query "Parameter.Value" --output text >cloudxinfo-eu-central-1.pem
```

2. Copy and paste content from the file 'cloudxinfo-eu-central-1.pem' to another file with the sane name (otherwise get error all the time 'Load key "cloudxinfo-eu-central-1.pem": invalid format ec2-user@ec2-3-67-6-34.eu-central-1.compute.amazonaws.com: Permission denied (publickey,gssapi-keyex,gssapi-with-mic).'

3. Reduce permission on the file:

```
$username = $env:USERNAME
icacls cloudxinfo-eu-central-1.pem /inheritance:r /grant:r "$($username):(R)"
icacls cloudxinfo-eu-central-1.pem /grant:r "$($username):(F,WDAC,RA,X)"
```



cloudxinfo-eu-central-1.pem 4/9/2024 6:02 PM PEM File
5.pem
cloudxinfo-eu-central-12.pem
NTUSER.DAT
.bash_history
.viminfo
.lesshst
.gitconfig
.ssh
Downloads
Desktop
OneDrive
OneDrive - EPAM
Pictures
Documents
Links

cloudxinfo-eu-central-1.pem Properties

General | Security | Details | Previous Versions

Object name: C:\Users\Viktoriia_Skirko\cloudxinfo-eu-central-1.pem

Group or user names:
Viktoriia_Skirko (EPAM\Viktoriia_Skirko)
SYSTEM
Administrators (EPUALVIW0986\Administrators)

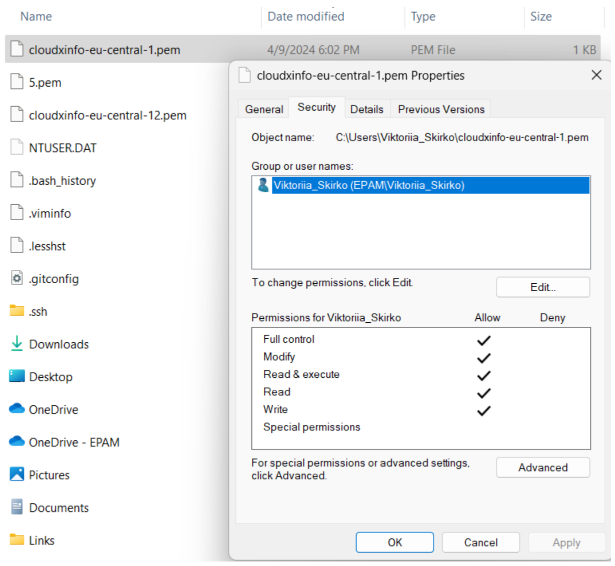
To change permissions, click Edit. Edit...

Permissions for Viktoriia_Skirko	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced. Advanced

OK Cancel Apply

Pic 6 – init permission on .pem file



Name	Date modified	Type	Size
cloudxinfo-eu-central-1.pem	4/9/2024 6:02 PM	PEM File	1 KB
5.pem			
cloudxinfo-eu-central-12.pem			
NTUSER.DAT			
.bash_history			
.viminfo			
.lesshst			
.gitconfig			
.ssh			
Downloads			
Desktop			
OneDrive			
OneDrive - EPAM			
Pictures			
Documents			
Links			

cloudxinfo-eu-central-1.pem Properties

General | Security | Details | Previous Versions

Object name: C:\Users\Viktoriia_Skirko\cloudxinfo-eu-central-1.pem

Group or user names:
Viktoriia_Skirko (EPAM\Viktoriia_Skirko)

To change permissions, click Edit. Edit...

Permissions for Viktoriia_Skirko	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

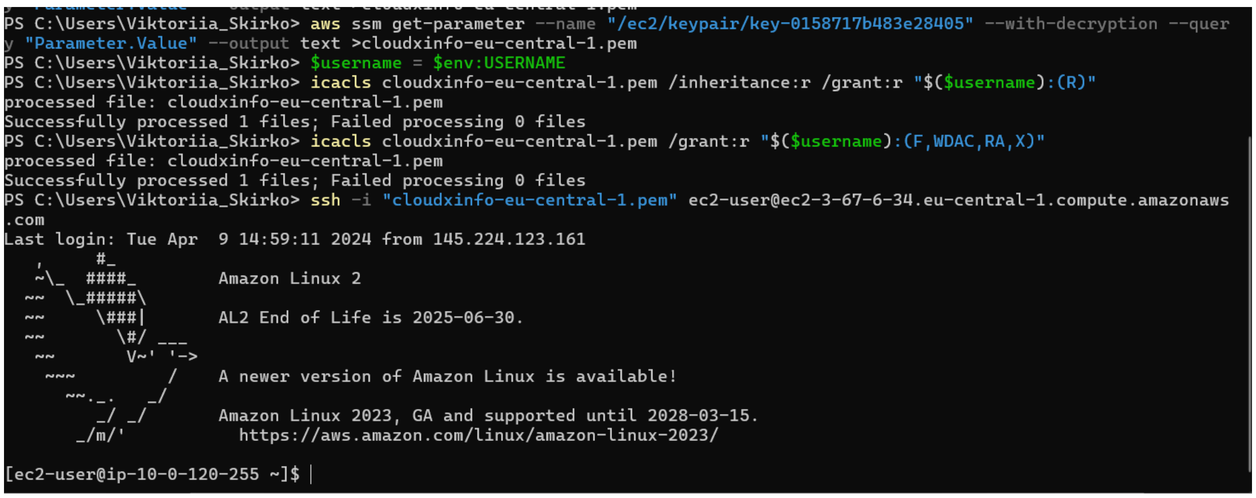
For special permissions or advanced settings, click Advanced. Advanced

OK Cancel Apply

Pic 7 – correct permission on .pem file

4. SSH connect to the public instance:

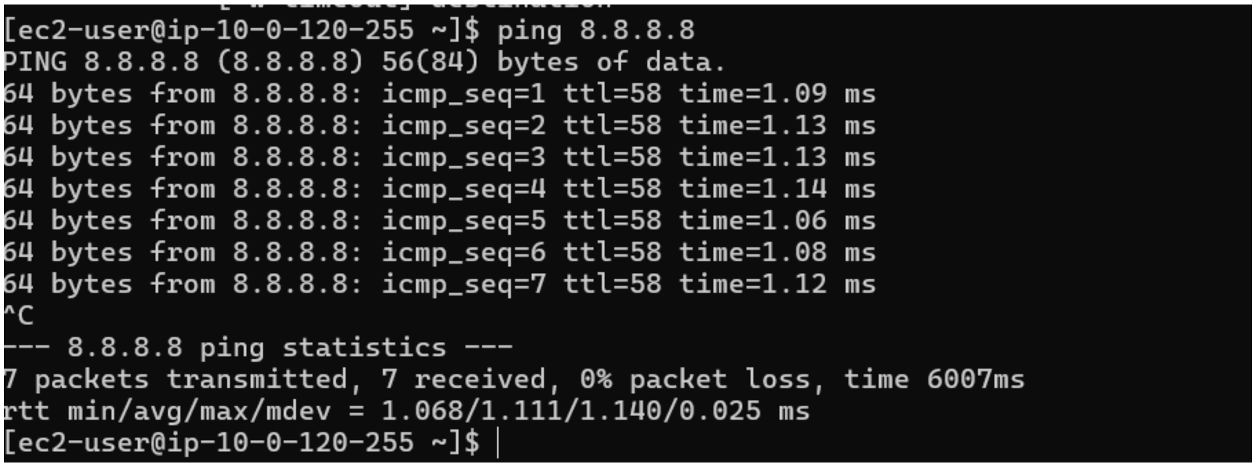
```
ssh -i "cloudxinfo-eu-central-1.pem" ec2-user@ec2-3-67-6-34.eu-central-
1.compute.amazonaws.com
```



```
PS C:\Users\Viktoriia_Skirko> aws ssm get-parameter --name "/ec2/keypair/key-0158717b483e28405" --with-decryption --quer
y "Parameter.Value" --output text >cloudxinfo-eu-central-1.pem
PS C:\Users\Viktoriia_Skirko> $username = $env:USERNAME
PS C:\Users\Viktoriia_Skirko> icacls cloudxinfo-eu-central-1.pem /inheritance:r /grant:r "$($username):(R)"
processed file: cloudxinfo-eu-central-1.pem
Successfully processed 1 files; Failed processing 0 files
PS C:\Users\Viktoriia_Skirko> icacls cloudxinfo-eu-central-1.pem /grant:r "$($username):(F,WDAC,RA,X)"
processed file: cloudxinfo-eu-central-1.pem
Successfully processed 1 files; Failed processing 0 files
PS C:\Users\Viktoriia_Skirko> ssh -i "cloudxinfo-eu-central-1.pem" ec2-user@ec2-3-67-6-34.eu-central-1.compute.amazonaws
.com
Last login: Tue Apr  9 14:59:11 2024 from 145.224.123.161
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-10-0-120-255 ~]$ |
```

Pic 8 – ssh connect to the public instance

5. Ping any internet resource on the instance



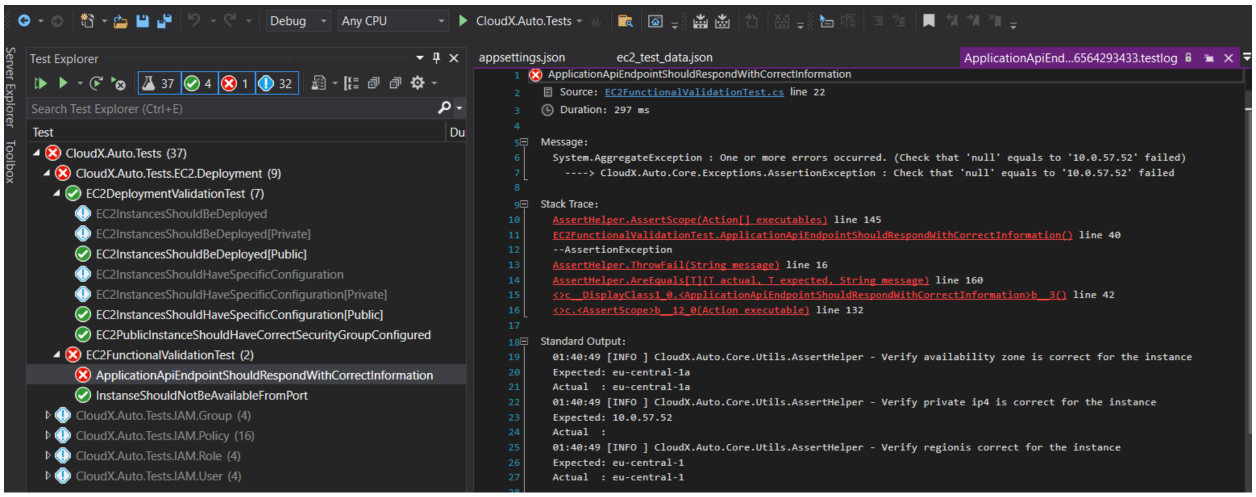
```
[ec2-user@ip-10-0-120-255 ~]$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=1.09 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=58 time=1.13 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=58 time=1.13 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=58 time=1.14 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=58 time=1.06 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=58 time=1.08 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=58 time=1.12 ms
^C
--- 8.8.8.8 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6007ms
rtt min/avg/max/mdev = 1.068/1.111/1.140/0.025 ms
[ec2-user@ip-10-0-120-255 ~]$ |
```

Pic 9 – ping internet resource

2.3 [Public Instance] Regression Deployment and Functional test execution

The cloudxinfo v=2 is deployed on public instance.

Tests were re-executed and 1 issues was found. The failed scenario is: CXQA-EC2-04


```
Test Explorer
37  4  1  32
Test
CloudX.Auto.Tests (37)
  CloudX.Auto.Tests.EC2.Deployment (9)
    EC2DeploymentValidationTest (7)
      EC2InstancesShouldBeDeployed
      EC2InstancesShouldBeDeployed[Private]
      EC2InstancesShouldBeDeployed[Public]
      EC2InstancesShouldHaveSpecificConfiguration
      EC2InstancesShouldHaveSpecificConfiguration[Private]
      EC2InstancesShouldHaveSpecificConfiguration[Public]
      EC2PublicInstanceShouldHaveCorrectSecurityGroupConfigured
    EC2FunctionalValidationTest (2)
      ApplicationApiEndpointShouldRespondWithCorrectInformation
      InstanseShouldNotBeAvailableFromPort
  CloudX.Auto.Tests.IAM.Group (4)
  CloudX.Auto.Tests.IAM.Policy (16)
  CloudX.Auto.Tests.IAM.Role (4)
  CloudX.Auto.Tests.IAM.User (4)
```

```
ApplicationApiEndpointShouldRespondWithCorrectInformation
 Source: EC2FunctionalValidationTest.cs line 22
 Duration: 297 ms

Message:
  System.AggregateException : One or more errors occurred. (Check that 'null' equals to '10.0.57.52' failed)
    ----> CloudX.Auto.Core.Exceptions.AssertionException : Check that 'null' equals to '10.0.57.52' failed

Stack Trace:
  AssertHelper.AssertScope(Action[] executables) line 145
  EC2FunctionalValidationTest.ApplicationApiEndpointShouldRespondWithCorrectInformation() line 40
  --AssertionException
  AssertHelper.ThrowFail(String message) line 16
  AssertHelper.AreEquals[T](T actual, T expected, String message) line 160
  <>c__DisplayClass1_0.<ApplicationApiEndpointShouldRespondWithCorrectInformation>b__3() line 42
  <>c.<AssertScope>b__12_0(Action executable) line 132

Standard Output:
  01:40:49 [INFO ] CloudX.Auto.Core.Utils.AssertHelper - Verify availability zone is correct for the instance
  Expected: eu-central-1a
  Actual  : eu-central-1a
  01:40:49 [INFO ] CloudX.Auto.Core.Utils.AssertHelper - Verify private ip4 is correct for the instance
  Expected: 10.0.57.52
  Actual  :
  01:40:49 [INFO ] CloudX.Auto.Core.Utils.AssertHelper - Verify regionis correct for the instance
  Expected: eu-central-1
  Actual  : eu-central-1
```

Pic 10 – tests are executed on cloudinfo v=2 and 1 test failed

2.4 Bug report

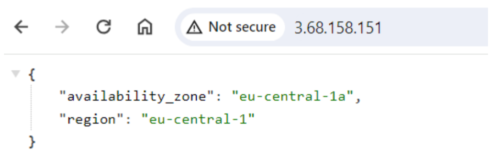
Summary: Application API endpoint responds with incorrect instance information from EC2 metadata

Type:	Bug
Priority:	Major
Affects Version:	2
Component:	EC2
Labels:	Regression

Pre-conditions:
1. EC2 instance with `publicInstanceIp` is established and running
2. cloudxinfo is deployed on the EC2 instance

Steps to reproduce:
1. Invoke API call `http://{publicInstanceIp}:{port}` any suitable way
2. Observe response.

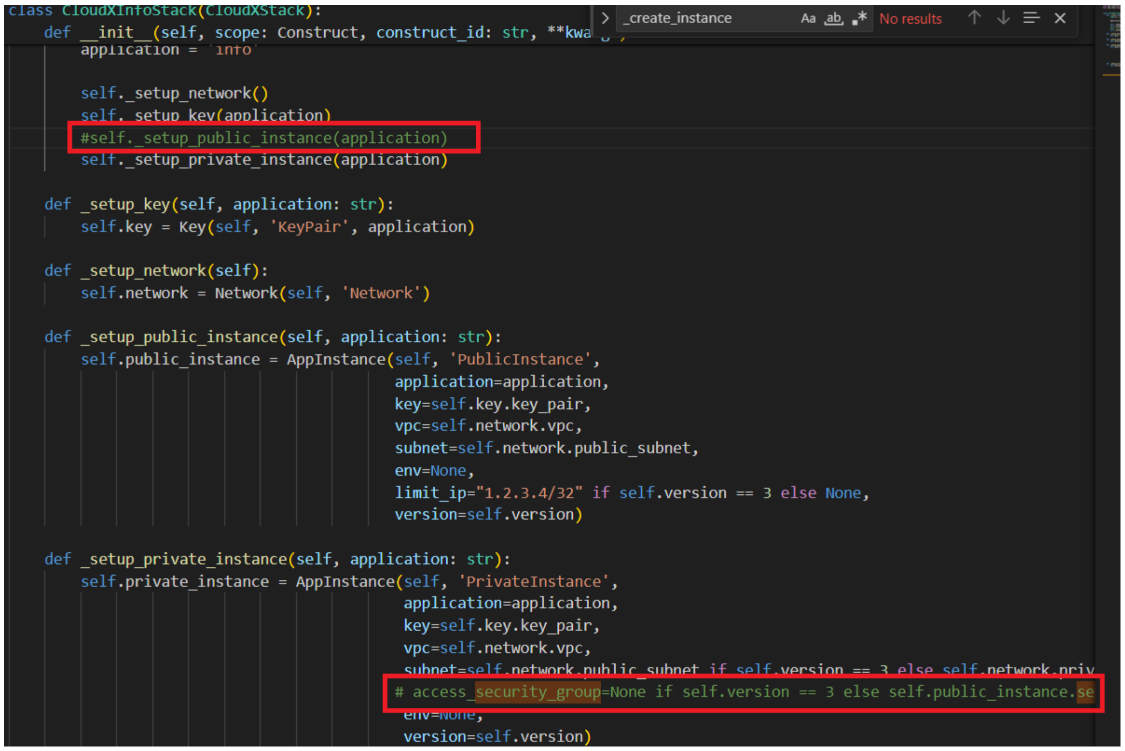
Actual results: The response has the following meta (no instance private IP):
- aws region
- availability zone


Not secure 3.68.158.151
```
{
    "availability_zone": "eu-central-1a",
    "region": "eu-central-1"
}
```

Expected results: The response should have the following meta
- aws region
- availability zone
- instance private IP

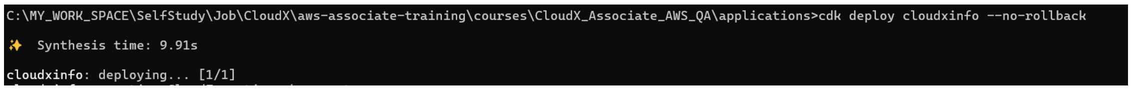
PART 2: PRIVATE INSTANCE

Destroy cloudxinfo and terminate public instance to be able to launch private instance.



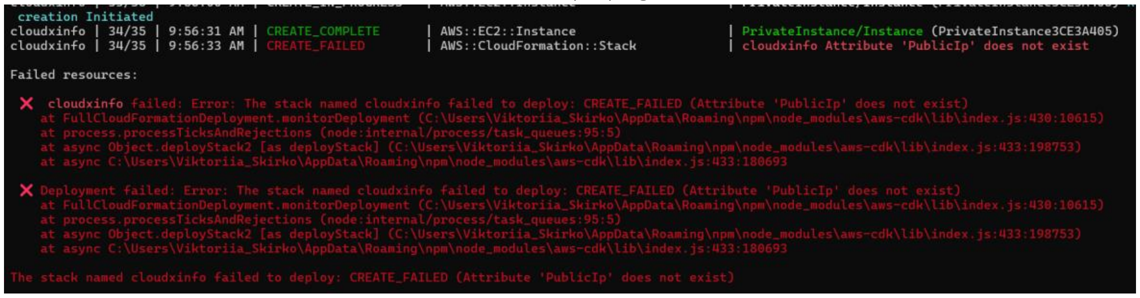
```
class CloudXInfoStack(CloudXStack):
    def __init__(self, scope: Construct, construct_id: str, **kwa
        application = 'info'

        self._setup_network()
        self._setup_key(application)
        #self._setup_public_instance(application)
        self._setup_private_instance(application)

    def _setup_key(self, application: str):
        self.key = Key(self, 'KeyPair', application)

    def _setup_network(self):
        self.network = Network(self, 'Network')

    def _setup_public_instance(self, application: str):
        self.public_instance = AppInstance(self, 'PublicInstance',
                                           application=application,
                                           key=self.key.key_pair,
                                           vpc=self.network.vpc,
                                           subnet=self.network.public_subnet,
                                           env=None,
                                           limit_ip="1.2.3.4/32" if self.version == 3 else None,
                                           version=self.version)

    def _setup_private_instance(self, application: str):
        self.private_instance = AppInstance(self, 'PrivateInstance',
                                            application=application,
                                            key=self.key.key_pair,
                                            vpc=self.network.vpc,
                                            subnet=self.network.public_subnet if self.version == 3 else self.network.priv
                                            # access_security_group=None if self.version == 3 else self.public_instance.se
                                            env=None,
                                            version=self.version)
```

Pic 11 – comment the code line to deploy cloudxinfo on private instance only



```
C:\MY_WORK_SPACE\SelfStudy\Job\CloudX\aws-associate-training\courses\CloudX_Associate_AWS_QA\applications>cdk deploy cloudxinfo --no-rollback

✨  Synthesis time: 9.91s

cloudxinfo: deploying... [1/1]
```

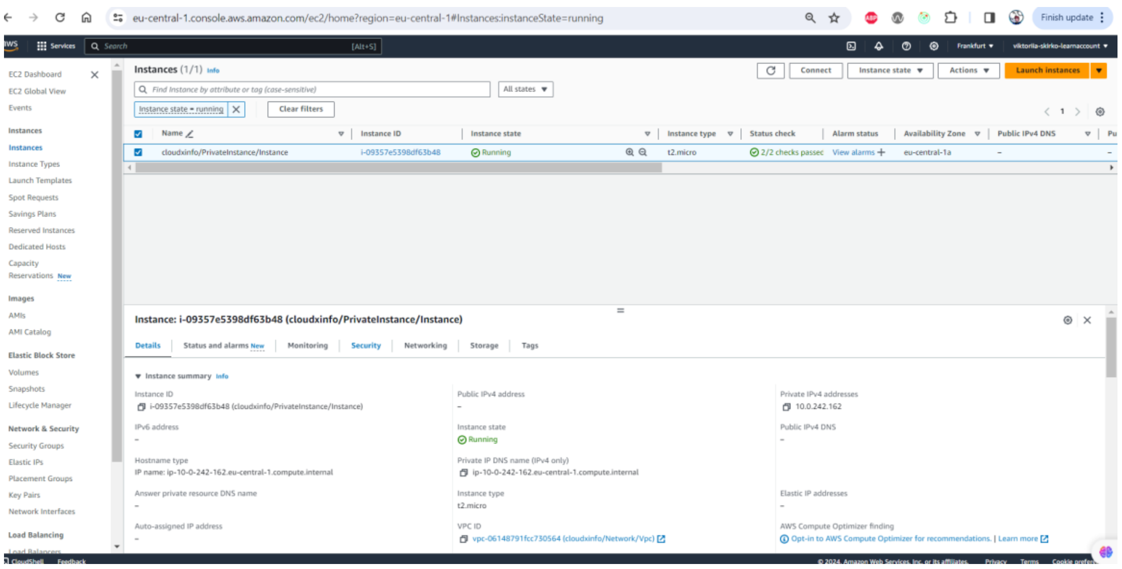
Pic 12 – start deploying of cloudxinfo stack



```
  creation Initiated
cloudxinfo | 34/35 | 9:56:31 AM | CREATE_COMPLETE      | AWS::EC2::Instance              | PrivateInstance/Instance (PrivateInstance3CE3A405)
cloudxinfo | 34/35 | 9:56:33 AM | CREATE_FAILED        | AWS::CloudFormation::Stack      | cloudxinfo Attribute 'PublicIp' does not exist

Failed resources:

 ❌  cloudxinfo failed: Error: The stack named cloudxinfo failed to deploy: CREATE_FAILED (Attribute 'PublicIp' does not exist)
    at FullCloudFormationDeployment.monitorDeployment (C:\Users\Viktoriia_Skirko\AppData\Roaming\npm\node_modules\aws-cdk\lib\index.js:430:10615)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)
    at async Object.deployStack2 [as deployStack] (C:\Users\Viktoriia_Skirko\AppData\Roaming\npm\node_modules\aws-cdk\lib\index.js:433:198753)
    at async C:\Users\Viktoriia_Skirko\AppData\Roaming\npm\node_modules\aws-cdk\lib\index.js:433:180693

 ❌ Deployment failed: Error: The stack named cloudxinfo failed to deploy: CREATE_FAILED (Attribute 'PublicIp' does not exist)
    at FullCloudFormationDeployment.monitorDeployment (C:\Users\Viktoriia_Skirko\AppData\Roaming\npm\node_modules\aws-cdk\lib\index.js:430:10615)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)
    at async Object.deployStack2 [as deployStack] (C:\Users\Viktoriia_Skirko\AppData\Roaming\npm\node_modules\aws-cdk\lib\index.js:433:198753)
    at async C:\Users\Viktoriia_Skirko\AppData\Roaming\npm\node_modules\aws-cdk\lib\index.js:433:180693

The stack named cloudxinfo failed to deploy: CREATE_FAILED (Attribute 'PublicIp' does not exist)
```

Pic 13 – deployment failed due to "No PublicIp" issue (it's obviously since the public instance was not launched)

Nevertheless, the private instance is running and some part of task for private instance validation can be implemented.



eu-central-1.console.aws.amazon.com/ec2/home?region=eu-central-1#Instances:instanceState=running

viktoriia-skirko-learnaccount

Instances (1/1)

Instance state = running

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
cloudxinfo/PrivateInstance/Instance	i-09357e5398df63b48	Running	t2.micro	2/2 checks passec	View alarms	eu-central-1a	–

Instance: i-09357e5398df63b48 (cloudxinfo/PrivateInstance/Instance)

Instance summary

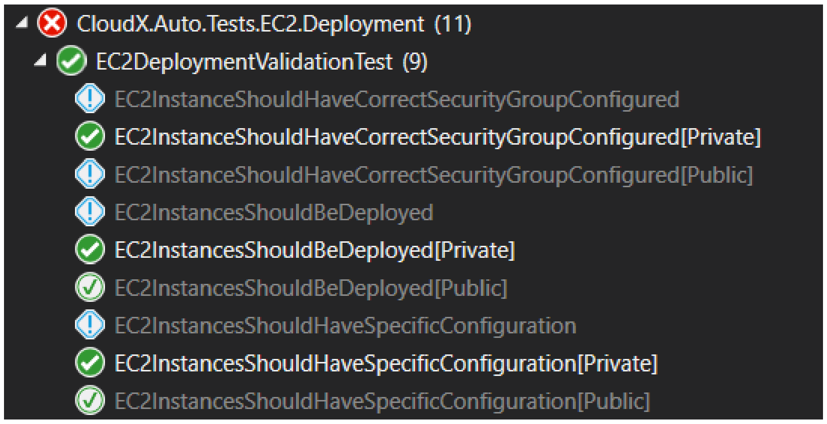
Instance ID: i-09357e5398df63b48 (cloudxinfo/PrivateInstance/Instance)	Public IPv4 address: –	Private IPv4 addresses: 10.0.242.162
IPv6 address: –	Instance state: Running	Public IPv4 DNS: –
Hostname type: IP name: ip-10-0-242-162.eu-central-1.compute.internal	Private IP DNS name (IPv4 only): ip-10-0-242-162.eu-central-1.compute.internal	
Answer private resource DNS name: –	Instance type: t2.micro	Elastic IP addresses: –
Auto-assigned IP address: –	VPC ID: vpc-06148791fcc730564 (cloudxinfo/Network/Vpc)	AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations. | Learn more

© 2024, Amazon Web Services, Inc. or its affiliates.

Pic 14 – private issue is running

2.5 [Private Instance] Deployment test execution

Since the private instance was deployed without public instance launching only deployment checks are allowed to execute



```
CloudX.Auto.Tests.EC2.Deployment (11)
  EC2DeploymentValidationTest (9)
    EC2InstanceShouldHaveCorrectSecurityGroupConfigured
    EC2InstanceShouldHaveCorrectSecurityGroupConfigured[Private]
    EC2InstanceShouldHaveCorrectSecurityGroupConfigured[Public]
    EC2InstancesShouldBeDeployed
    EC2InstancesShouldBeDeployed[Private]
    EC2InstancesShouldBeDeployed[Public]
    EC2InstancesShouldHaveSpecificConfiguration
    EC2InstancesShouldHaveSpecificConfiguration[Private]
    EC2InstancesShouldHaveSpecificConfiguration[Public]
```

Pic 15 – deployment test execution for **private** instance only

Link to test execution on private instance: demo

Since there is no ability to launch private and public instance simultaneously the following requirements were not covered for private instance:

1. Functional tests (CXQA-EC2-04)
2. Part of CXQA-EC2-03:
   a. The private instance should be accessible only from the public instance by SSH and HTTP protocols only;
   b. Private instance should have access to the internet;