CloudX Associate: AWS for Testers

RDS

TASK 6 – RDS

1. DEPLOY THE IMAGE APPLICATION

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

2. DEPLOYMENT VALIDATION

Create a manual/automated deployment validation test suite that covers the following requirements:
CXQA-RDS-01: Application Instance requirements:

1. The application database (MySQL RDS instance) is deployed in the private subnet and should be accessible only from the application's public subnet, but not from the public internet.
2. The application can access to MySQL RDS via an Security Group.

CXQA-RDS-02: RDS Instance requirements:

1 Instance type: db.t3.micro
2 Multi-AZ: no
3 Storage size: 100 GiB
4 Storage type: General Purpose SSD (gp2)
5 Encryption: not enabled
6 Instance tags: cloudx: qa
7 Database type: MySQL
8 Database version: 8.0.32

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 deployment validation test suite that covers the following requirements:
CXQA-RDS-03: The uploaded image metadata is stored in MySQL RDS database:

1 image key
2 image size
3 image type
4 last modification date and time

Key/property names might be different.
CXQA-RDS-04: The image metadata is returned by http://{INSTANCE PUBLIC IP}/api/image/{image_id} GET request
CXQA-RDS-05: The image metadata for the deleted image is also deleted from the database

3.1 Testing Tools:

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

- MySQL database client (NB: database is accessible only from application instance, not from the internet)

4. REGRESSION TESTING

1. Deploy **version 3** of the **cloudximage** application deployment instructions.
2. Execute deployment and functional validation test suites against new application version deployment.
3. 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 RDS THEORY

What is RDS? What engines does it support?

Amazon Relational Database Service (RDS) is a managed relational database service provided by Amazon Web Services (AWS). It simplifies the setup, operation, and scaling of relational databases in the cloud. RDS automates common administrative tasks such as hardware provisioning, database setup, patching, and backups, allowing developers to focus on building applications rather than managing infrastructure.

RDS supports several popular relational database engines, including:

- MySQL: A widely-used open-source relational database management system.
- PostgreSQL: An advanced open-source relational database system known for its extensibility and support for advanced features.
- MariaDB: A community-developed fork of MySQL, designed for high performance, reliability, and ease of use.
- Oracle Database: A commercial relational database management system known for its robustness, scalability, and enterprise-grade features.
- SQL Server: A relational database management system developed by Microsoft, offering comprehensive features for data management, analytics, and business intelligence.
- Amazon Aurora: A MySQL and PostgreSQL-compatible relational database built for the cloud, offering high performance, scalability, and availability with built-in features such as automated failover, continuous backup, and replication.

What is RDS pricing?

Amazon RDS pricing is based on several factors, including the database engine, instance type, storage, data transfer, and additional features. Here's an overview of the key components that contribute to RDS pricing:

- **Database Engine:** The pricing may vary depending on the database engine you choose, such as MySQL, PostgreSQL, MariaDB, Oracle, SQL Server, or Amazon Aurora.
- **Instance Type:** RDS offers different instance types with varying CPU, memory, and storage capacity. The pricing is based on the chosen instance type, and it can vary depending on factors like region and availability zone.
- **Storage:** The cost of storage depends on the amount of provisioned storage for your database instance. RDS offers different storage types, such as General Purpose SSD (gp2), Provisioned IOPS SSD (io1), and Magnetic (standard) storage.
- **Data Transfer:** Data transfer costs may apply if your RDS instance transfers data out of the AWS region, or if it communicates with other AWS services or the internet.
- **Backup and Snapshots:** RDS provides automated backups and manual snapshots for your database instances. While automated backups are included in the service price, additional charges may apply for storing manual snapshots beyond the free allocation.
- **Multi-AZ Deployment:** If you choose a Multi-AZ deployment for high availability, there may be additional costs associated with maintaining standby instances in a secondary availability zone.
- **Reserved Instances:** AWS offers Reserved Instances for RDS, allowing you to commit to a one- or three-year term in exchange for discounted hourly rates compared to on-demand pricing.
- **Additional Features:** Depending on your requirements, you may incur additional charges for features such as encryption at rest, Read Replicas, or database options like Amazon Aurora Serverless.

What is read replica in RDS and how it works?

A Read Replica in Amazon RDS is a copy of your primary database instance that allows you to offload read traffic from your primary instance and scale out read-heavy workloads. Read Replicas are asynchronous copies of the primary database that are created in the same or different AWS regions, depending on your requirements.

Here's how Read Replicas work in Amazon RDS:

- Creation: You can create one or more Read Replicas from your primary RDS instance using the AWS Management Console, CLI, or SDK. When creating a Read Replica, you specify the source DB instance (primary) and the target DB instance (replica).
- Asynchronous Replication: Once the Read Replica is created, it starts receiving changes (data modifications) from the primary instance asynchronously. Amazon RDS uses asynchronous replication to propagate changes from the primary instance to the Read Replica.
- Read Scaling: Read Replicas are primarily used to offload read traffic from the primary instance, allowing you to scale out read-heavy workloads. Applications can distribute read queries across multiple Read Replicas to improve read throughput and reduce the load on the primary instance.
- High Availability: Read Replicas can also improve the availability and fault tolerance of your database environment. If the primary instance becomes unavailable due to a failure or maintenance, you can promote one of the Read Replicas to become the new primary instance, minimizing downtime and ensuring continuity of operations.
- Use Cases: Read Replicas are commonly used for reporting, analytics, read-intensive applications, and scaling out read-heavy workloads. By distributing read traffic across multiple replicas, you can achieve better performance, scalability, and fault tolerance for your applications.
- Replication Lag: It's important to note that Read Replicas operate with some level of replication lag, meaning there may be a delay between when a write operation occurs on the primary instance and when it is applied to the Read Replica. The replication lag depends on factors such as the workload, network latency, and the distance between the primary and replica instances.

What RDS operational (maintenance, monitoring) practices do you know?

There are several operational practices for managing Amazon RDS effectively, including maintenance, monitoring, and optimization. Here are some key practices:

- Regular Backup and Restore: Schedule automated backups to ensure data durability and availability. Additionally, perform regular database restores to test backup integrity and recovery procedures.
- Database Patching and Updates: Stay up to date with the latest database engine versions and security patches provided by AWS. Schedule maintenance windows for patching and updates to minimize downtime and impact on production workloads.
- Performance Monitoring: Use Amazon CloudWatch and RDS Performance Insights to monitor database performance metrics such as CPU utilization, memory usage, disk I/O, and query execution times. Set up alarms and notifications to detect performance anomalies and issues.
- Security Hardening: Implement security best practices such as enabling encryption at rest using AWS Key Management Service (KMS), configuring network security groups to restrict access to RDS instances, and regularly reviewing and rotating database credentials.
- Scaling and Right-Sizing: Monitor database performance and usage patterns to identify scaling needs. Use Amazon RDS features such as Auto Scaling, Read Replicas, and Multi-AZ deployments to scale resources up or down based on demand and workload requirements.
- Database Parameter Tuning: Adjust database parameters and configurations to optimize performance and resource utilization. Experiment with different parameter settings and monitor the impact on database performance metrics.
- High Availability and Failover Testing: Configure Multi-AZ deployments for high availability and automatic failover. Regularly test failover scenarios to ensure continuity of operations and minimize downtime in the event of a primary instance failure.
- Logging and Auditing: Enable database logs such as Amazon RDS Enhanced Monitoring, error logs, slow query logs, and audit logs. Analyze logs to troubleshoot performance issues, identify security threats, and comply with regulatory requirements.

- Cost Optimization: Analyze and optimize RDS costs by right-sizing database instances, utilizing Reserved Instances, implementing storage tiering strategies, and optimizing data transfer costs.
- Database Lifecycle Management: Implement database lifecycle policies for data retention, archiving, and purging. Define data retention policies based on regulatory compliance requirements and business needs.

What RDS capacity planning best practices do you know?

Here are some best practices for RDS capacity planning:

- Understand Workload Characteristics: Analyze your application's workload patterns, including read/write ratios, query complexity, concurrency, and data access patterns. Use tools like Amazon CloudWatch and RDS Performance Insights to gather performance metrics and understand resource utilization patterns.
- Right-Sizing Instances: Choose the appropriate RDS instance type and size based on your application's performance requirements, workload characteristics, and anticipated growth. Consider factors such as CPU, memory, storage, and I/O performance when selecting instance types.
- Use Provisioned IOPS (PIOPS) for High Performance: If your workload requires high I/O performance and low latency, consider using Provisioned IOPS (PIOPS) storage volumes. Provision sufficient IOPS to meet your application's I/O requirements and ensure consistent performance.
- Implement Read Replicas for Scalability: If you have read-heavy workloads or require horizontal scalability, consider using Read Replicas to offload read traffic from the primary instance. Distribute read queries across multiple replicas to improve read throughput and scale out read-intensive workloads.
- Enable Auto Scaling: Use Auto Scaling to automatically adjust the number and size of RDS instances based on changes in workload demand. Configure scaling policies to add or remove instances dynamically in response to changes in CPU utilization, database connections, or other metrics.
- Monitor Resource Utilization: Continuously monitor resource utilization metrics such as CPU, memory, storage, and I/O to identify performance bottlenecks and capacity constraints. Set up CloudWatch alarms and notifications to alert you of potential resource shortages or performance issues.
- Forecast Growth and Scale Proactively: Estimate future growth and capacity requirements based on historical usage data, application roadmaps, and business projections. Plan for capacity upgrades and scaling activities proactively to avoid performance degradation and downtime due to resource constraints.
- Regularly Review and Optimize: Conduct periodic reviews of your RDS environment to identify opportunities for optimization and cost reduction. Analyze performance metrics, usage patterns, and cost implications to optimize resource allocation, instance types, and storage configurations.
- Test Performance at Scale: Perform load testing and performance benchmarking exercises to evaluate the scalability and performance of your RDS deployments under realistic production conditions. Identify performance bottlenecks, fine-tune configurations, and validate capacity planning assumptions.
- Leverage AWS Tools and Services: Utilize AWS tools and services such as AWS Trusted Advisor, AWS Cost Explorer, and AWS Budgets to gain insights into your RDS usage, costs, and performance. Use these insights to refine capacity planning strategies and optimize resource utilization over time.

What RDS testing and profiling best practices do you know?

Here are some best practices for testing and profiling RDS instances:

- Database Schema Testing: Perform thorough testing of database schema changes, including table modifications, index additions, and data migrations. Use tools like AWS Database Migration Service (DMS) to replicate and validate schema changes in a test environment before applying them to production.
- Functional Testing: Conduct functional testing to validate the correctness and integrity of database operations, including CRUD (Create, Read, Update, Delete) operations, data validation, transaction handling, and error handling. Use automated testing frameworks and tools to streamline testing processes and ensure comprehensive coverage.

- **Performance Testing:** Use performance testing techniques to assess the scalability, throughput, and latency of RDS instances under various workloads and load conditions. Benchmark database performance metrics such as query execution times, transaction rates, and resource utilization to identify performance bottlenecks and optimize configurations.
- **Load Testing:** Simulate realistic production workloads and user concurrency levels using load testing tools and frameworks. Measure database response times, throughput, and resource consumption under load to evaluate system performance, scalability, and stability. Identify performance thresholds and capacity limits through stress testing exercises.
- **Query Profiling:** Profile database queries using built-in database profiling tools, query analyzers, and performance monitoring solutions. Analyze query execution plans, identify long-running or inefficient queries, and optimize SQL statements, indexes, and database schemas to improve query performance and resource utilization.
- **Index Optimization:** Evaluate and optimize database indexes to enhance query performance and minimize disk I/O. Use index tuning tools, query optimizers, and database advisors to identify redundant, unused, or poorly performing indexes. Consider index types, column cardinality, and query patterns when optimizing indexes.
- **Concurrency Testing:** Test database concurrency and transaction isolation levels to ensure data consistency, integrity, and isolation. Simulate concurrent database transactions, locking scenarios, and race conditions to identify potential concurrency issues and ensure proper transactional behavior.
- **Failover and Disaster Recovery Testing:** Validate High Availability (Multi-AZ) and Disaster Recovery (DR) configurations by performing failover and failback tests. Simulate primary instance failures, network outages, and regional disasters to verify the effectiveness of automated failover mechanisms and data replication processes.
- **Backup and Restore Testing:** Test backup and restore procedures regularly to ensure data recoverability and integrity. Perform full backups, incremental backups, and point-in-time restores to validate backup retention policies, recovery time objectives (RTO), and recovery point objectives (RPO).
- **Security Testing:** Conduct security assessments and penetration tests to identify vulnerabilities, misconfigurations, and compliance gaps in RDS deployments. Test authentication mechanisms, encryption settings, network access controls, and data protection measures to mitigate security risks and ensure regulatory compliance.

What are the advantages of RDS over manually managed databases?

Here are some of the key advantages of RDS:

- **Managed Service:** RDS is a fully managed service provided by AWS, which automates routine database administration tasks such as hardware provisioning, database setup, patching, backups, and maintenance. This relieves developers and administrators from the burden of managing infrastructure and allows them to focus on building and optimizing applications.
- **Ease of Deployment:** RDS makes it easy to deploy and scale relational databases in the cloud with just a few clicks or API calls. You can choose from multiple database engines (e.g., MySQL, PostgreSQL, SQL Server, Oracle) and instance types to meet your application's requirements, without the need for manual setup and configuration.
- **High Availability and Fault Tolerance:** RDS offers built-in high availability and fault tolerance features, such as Multi-AZ deployments and automated backups. Multi-AZ deployments replicate data synchronously across multiple Availability Zones (AZs) to provide automatic failover and data redundancy, ensuring business continuity and minimizing downtime.
- **Scalability:** RDS allows you to scale your database resources up or down dynamically based on changing workload demands. You can easily resize compute and storage resources, add read replicas for read scalability, and leverage features like Auto Scaling to automatically adjust capacity in response to traffic

spikes.

- **Security and Compliance:** RDS provides robust security features to protect your data, including encryption at rest and in transit, network isolation using Virtual Private Cloud (VPC), IAM authentication, and fine-grained access controls. RDS also helps you comply with regulatory requirements such as GDPR, HIPAA, and PCI DSS.
- **Performance Monitoring and Optimization:** RDS offers built-in performance monitoring and optimization tools, such as Amazon CloudWatch and RDS Performance Insights. These tools allow you to monitor database performance metrics, identify performance bottlenecks, and optimize database configurations to improve efficiency and reliability.
- **Cost-Effectiveness:** With RDS, you pay only for the resources you use on a pay-as-you-go basis, with no upfront costs or long-term commitments. RDS also offers pricing models such as Reserved Instances and Savings Plans for cost savings and flexibility.
- **Automated Backups and Disaster Recovery:** RDS automatically performs regular backups of your databases and retains them for a specified retention period. You can also create manual snapshots for point-in-time recovery and disaster recovery purposes, providing data protection and resiliency against data loss or corruption.

What is the difference between multi-AZ deployment and read replicas in RDS?

Multi-AZ deployment and read replicas are both features of Amazon RDS designed to improve availability, scalability, and performance, but they serve different purposes and have distinct characteristics:

Multi-AZ Deployment:

- **Purpose:** Multi-AZ (Availability Zone) deployment is primarily focused on providing high availability and fault tolerance for the primary database instance.
- **Configuration:** In a Multi-AZ deployment, a standby replica of the primary database instance is automatically created in a different Availability Zone within the same AWS region. The standby replica remains in sync with the primary instance through synchronous replication.
- **Failover:** In the event of a planned maintenance event, availability zone failure, or instance failure, Amazon RDS automatically fails over to the standby replica, promoting it to become the new primary instance. This process is automatic and does not require manual intervention.
- **Use Case:** Multi-AZ deployments are suitable for production workloads that require high availability and business continuity, where minimal downtime is critical.

Read Replicas:

- **Purpose:** Read replicas are primarily used to offload read traffic from the primary database instance, improving read scalability and performance.
- **Configuration:** Read replicas are asynchronous copies of the primary database instance that can be created in the same or different AWS regions. They receive and apply changes from the primary instance asynchronously through replication.
- **Scaling:** Read replicas can serve read-only queries, distributing read traffic across multiple replicas to improve read throughput and reduce the load on the primary instance. They can also be used for read scaling and read-heavy workloads.
- **Failover:** Read replicas do not support automatic failover like Multi-AZ deployments. If the primary instance fails, failover to a read replica requires manual intervention and promotion of the replica to become the new primary instance.
- **Use Case:** Read replicas are suitable for scenarios where read scalability, reporting, analytics, or read-intensive workloads are required, but they do not provide the same level of high availability and automatic failover as Multi-AZ deployments

What security features does RDS provide?

Some of the key security features provided by RDS include:

Network Isolation:

- RDS instances can be deployed within a Virtual Private Cloud (VPC), allowing you to control network access using security groups and network access control lists (ACLs).
- You can define inbound and outbound rules to restrict access to RDS instances based on IP addresses, CIDR blocks, or other security group memberships.

Encryption:

- RDS supports encryption at rest and in transit to protect your data.
- At rest: You can enable encryption for RDS instances using AWS Key Management Service (KMS) to encrypt data stored in the database and automated backups.
- In transit: RDS encrypts data in transit using SSL/TLS to secure connections between client applications and database instances.

Authentication and Authorization:

- RDS supports multiple authentication mechanisms, including password-based authentication, IAM database authentication, and integration with AWS Directory Service.
- IAM database authentication allows you to authenticate database users using IAM credentials, eliminating the need to manage database passwords.
- Fine-grained access control can be implemented using database roles, privileges, and permissions to restrict access to sensitive data and operations.

Database Auditing and Logging:

- RDS provides built-in database auditing and logging features to monitor database activity, track changes, and comply with regulatory requirements.
- You can enable database logs such as error logs, slow query logs, and audit logs to capture database events and activities for troubleshooting, security analysis, and compliance audits.

Patch Management:

- RDS automates patch management and software updates for the underlying database engines, including security patches and bug fixes.
- You can configure maintenance windows to schedule updates and minimize downtime impact on production workloads.

Backup and Recovery:

- RDS offers automated backup and recovery capabilities to protect against data loss and corruption.
- You can schedule automated backups and configure retention periods to retain backup copies for a specified duration.
- RDS also supports point-in-time recovery, allowing you to restore databases to a specific point in time within the backup retention period.

Compliance and Certifications:

- RDS is compliant with various industry standards and certifications, including SOC 1, SOC 2, SOC 3, ISO 27001, PCI DSS, HIPAA, and GDPR.
- By leveraging RDS, you can build and operate compliant database environments that meet regulatory requirements and industry standards.

What is AWS Aurora?

Amazon Aurora is a MySQL and PostgreSQL-compatible relational database engine built for the cloud, designed to offer high performance, scalability, availability, and durability. It is a fully managed database service provided by Amazon Web Services (AWS) that combines the speed and reliability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases.

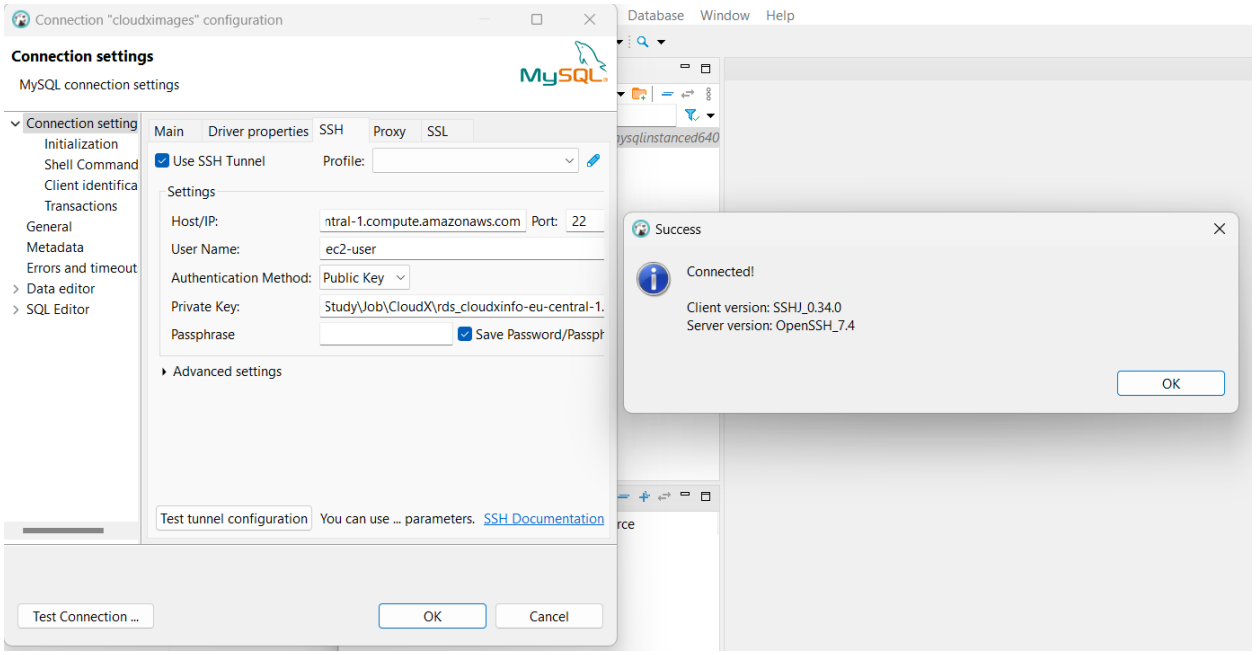
TASK IMPLEMENTATION

0. PRE-SETUP

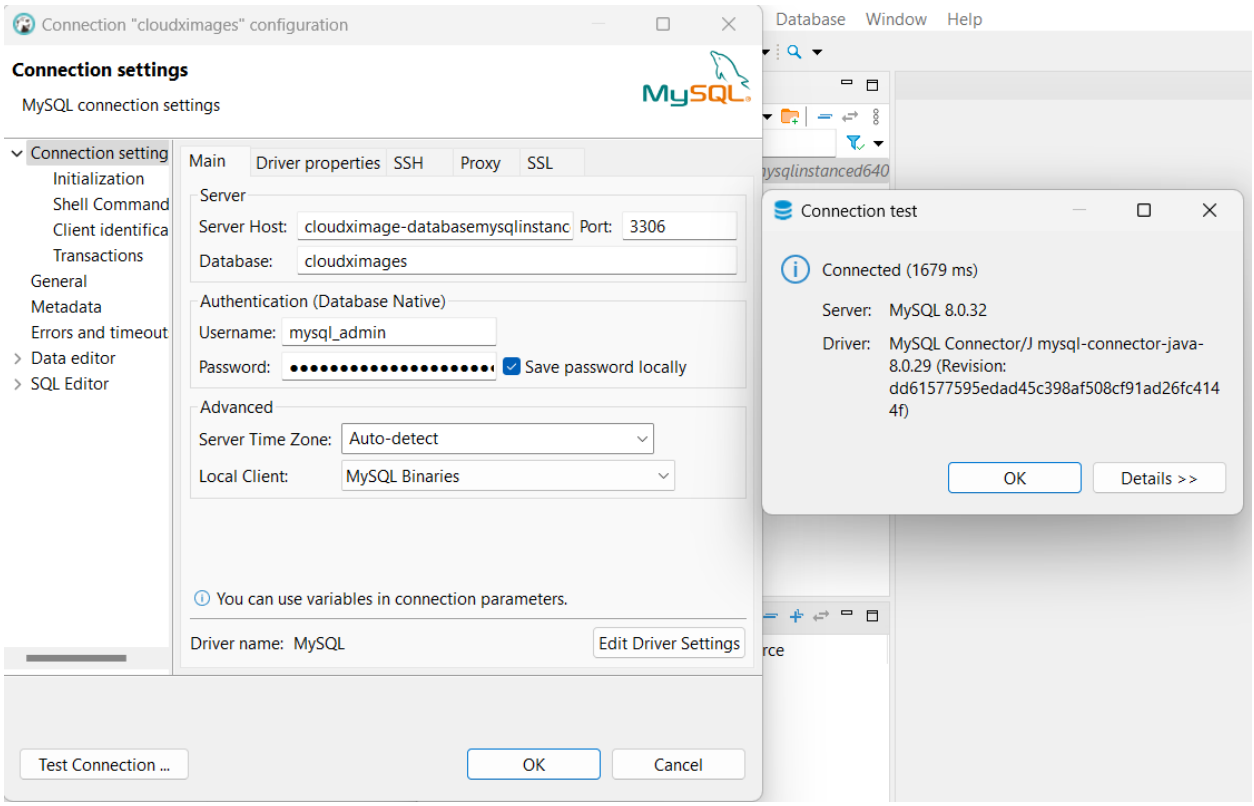
Connection to DB cloudximages via DBVeawer.

```
arn:aws:cloudformation:eu-central-1:765688149020:stack/cloudximage/3074fa90-0bba-11ef-b920-026bd087bd1f
Outputs:
cloudximage.AppInstanceInstanceIdD25F85EA = i-0a55278741e79eeb2
cloudximage.AppInstancePrivateDnsD4DDF466 = ip-10-0-46-116.eu-central-1.compute.internal
cloudximage.AppInstancePrivateIp933436FD = 10.0.46.116
cloudximage.AppInstancePublicDnsDF9B0F20 = ec2-3-70-128-151.eu-central-1.compute.amazonaws.com
cloudximage.AppInstancePublicIp619D309B = 3.70.128.151
cloudximage.DatabaseDbHost133C3116 = cloudximage-databasemysqlinstanced64026b2-uegrnefhl4j9.cloyam22ocjh.eu-central-1.rds.amazonaws.com
cloudximage.DatabaseDbName732AC082 = cloudximages
cloudximage.DatabaseDbPortD023B3A0 = 3306
cloudximage.DatabaseDbSecretNameAC9C8968 = DatabaseDBSecretD08C53F5-XCP2fY5bOp3O
cloudximage.DatabaseDbUsername3CEA241B = mysql_admin
cloudximage.ImageStoreImageBucketName23D060B2 = cloudximage-imagestorebucketf57d958e-eorc4wcuuxxj
cloudximage.KeyPairKeyId00586FD9 = key-02653e8fc5c7e6329
cloudximage.QueueQueueUrlC30FF916 = https://sqs.eu-central-1.amazonaws.com/765688149020/cloudximage-QueueSQSQueueE7532512-KwLXGuVIOUUI
cloudximage.TopicTopicArnB66B79C2 = arn:aws:sns:eu-central-1:765688149020:cloudximage-TopicSNSTopic086466D7-UAgrQINNdQ6c
Stack ARN:

âœ¨  Total time: 350.08s


PS C:\MY_WORK_SPACE\SelfStudy\Job\CloudX\aws-associate-training\courses\CloudX_Associate_AWS_QA\applications> aws secretsmanager get-secret-value --secret-i
d DatabaseDBSecretD08C53F5-XCP2fY5bOp3O
{
    "ARN": "arn:aws:secretsmanager:eu-central-1:765688149020:secret:DatabaseDBSecretD08C53F5-XCP2fY5bOp3O-XRiJ4t",
    "Name": "DatabaseDBSecretD08C53F5-XCP2fY5bOp3O",
    "VersionId": "1565c25c-446d-443a-9731-3de487a4b55c",
    "SecretString": "{\"password\":\"                                    \",\"dbname\":\"cloudximages\",\"engine\":\"mysql\",\"port\":3306,\"dbInstanceIdentifie
r\":\"cloudximage-databasemysqlinstanced64026b2-uegrnefhl4j9\",\"host\":\"cloudximage-databasemysqlinstanced64026b2-uegrnefhl4j9.cloyam22ocjh.eu-central-1.r
ds.amazonaws.com\",\"username\":\"mysql_admin\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": "2024-05-06T18:11:07.889000+03:00"
```

Pic 1 – Obtaining DB Secrets



Pic 2 – SSH setup



Pic 3 – Connection setup

1. DEPLOYMENT VALIDATION

1.1 TAF

Link to repo with task implemented: https://github.com/Chubaka2612/CloudX/pull/4/files

The TAF solution was extended:

- CloudX.Auto.Tests – the following test suite was added:
  - RDS – covers CXQA-RDS-01, CXQA-RDS-02, CXQA-RDS-03, CXQA-RDS-04, CXQA-RDS-05 scenarios;
- RDSService class – wrapper on `AmazonRDS` for interaction with rds resources;
- MySqlClient class – class for interaction with cloudximages db.

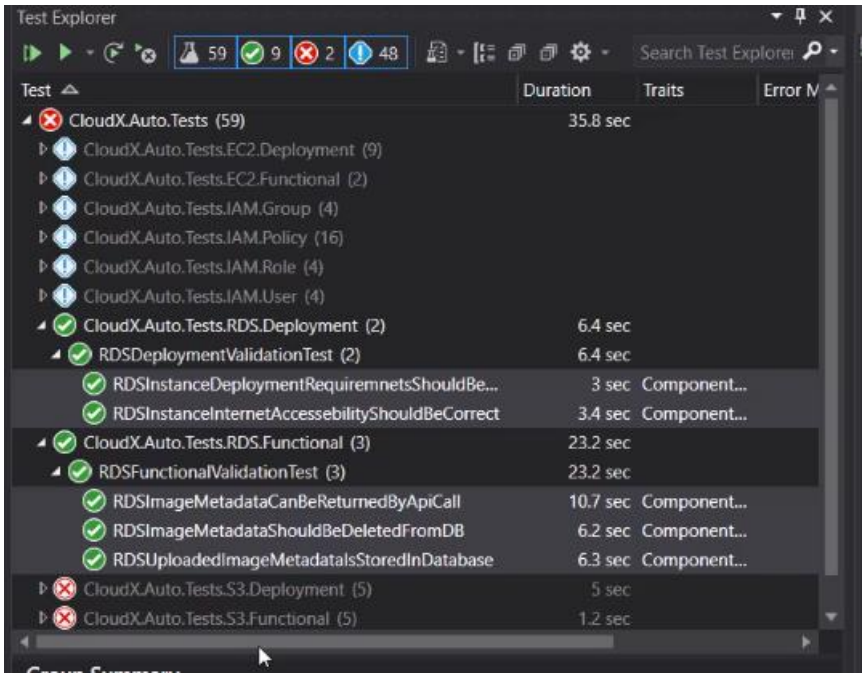
In total includes 5 new tests.

Used tools:

1. NUnit – test runner
2. AWS SDK for .NET – API for AWS resource interaction
3. Renci.SshNet – Library for ssh checks
4. MySqlConnected – DriverManager for CRUD operation on MySQL db

1.2 Deployment and Functional test execution

For cloudximage v=1 there are no issues found. Results are below:



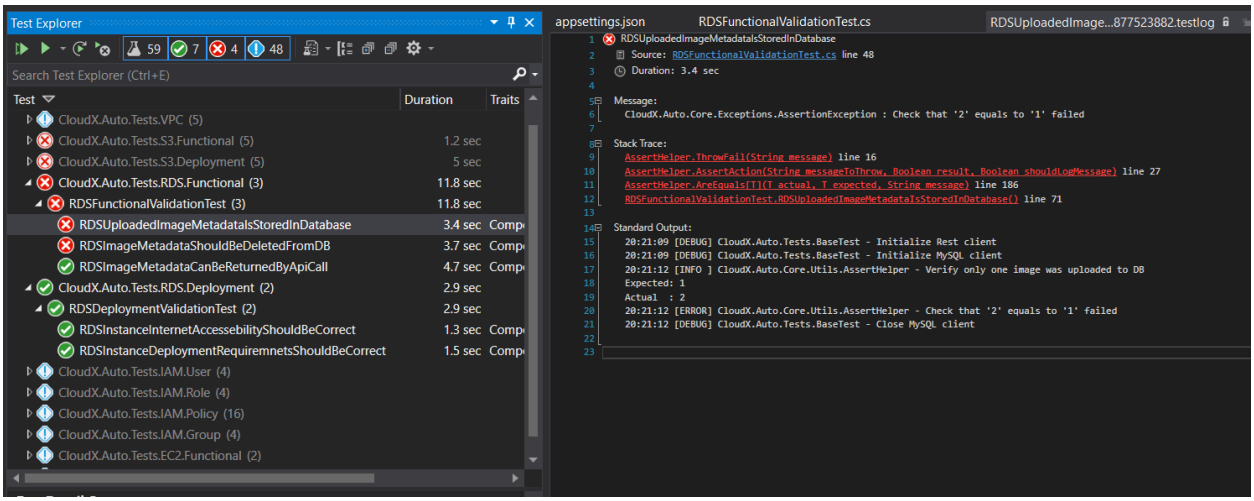
Pic 4 – test results execution for cloudimage v = 1

Link to test execution: demo

1.3 Regression Deployment and Functional test execution

The cloudximage v=3 is deployed. Two tests failed.

Two functional test detects 2 issues.



Pic 5 – functional tests are executed on cloudximage v=3 and 2 test failed

Link to regression test execution: demo

1.4 Bug report

1. Summary: [Functional] The image file is uploaded twice during API POST call

Type: Bug
Priority: Critical
Affects Version: 3
Component: RDS
Labels: Regression

Pre-conditions:

1. cloudximage is deployed on the EC2 instance
2. The rds is configured

Steps to reproduce:

1. Call POST http://{publicIp}/api/ui/image with {*name*} (i.e. s3_test_photo.jpg) (use swagger, postman or another appropriate option)
2. Observe results in cloudximages DB using the following sql query:
   `SELECT * FROM images WHERE object_key LIKE '%{name}%'`

Actual results:

1. POST method returns 200 (OK) and has id of entity created.
2. In cloudximages DB there are 2 images with object_key = '{name}' returned to sql query

Expected results:

1. POST method returns 200 (OK) and has id of entity created.
2. In cloudximages DB there is only one image with object_key = '*{name}*'. No duplication is occurred.

Link to demo: demo

2. Summary: [Functional] User is not able to delete image from DB via DELETE API call

Type: Bug
Priority: Major
Affects Version: 3
Component: RDS
Labels: Regression

Pre-conditions:

1. cloudximage is deployed on the EC2 instance
2. The rds is configured

Steps to reproduce:

1. Call DELETE http://{publicIp}/api/ui/image/{imageId}, where imageId – id of existing image. (use swagger, postman or another appropriate option)
2. Observe results in cloudximages DB using the following sql query:
   `SELECT * FROM images WHERE id = '{imageId}'`

Actual results:

1. DELETE method returns 200 (OK)
2. One item is returned as a result of execution of sql query provided

Expected results:

1. DELETE method returns 200 (OK).
2. No item should be returned as a result of execution of sql query provided.

Note: Possible root cause is the same as for issue#1

Link to demo: demo