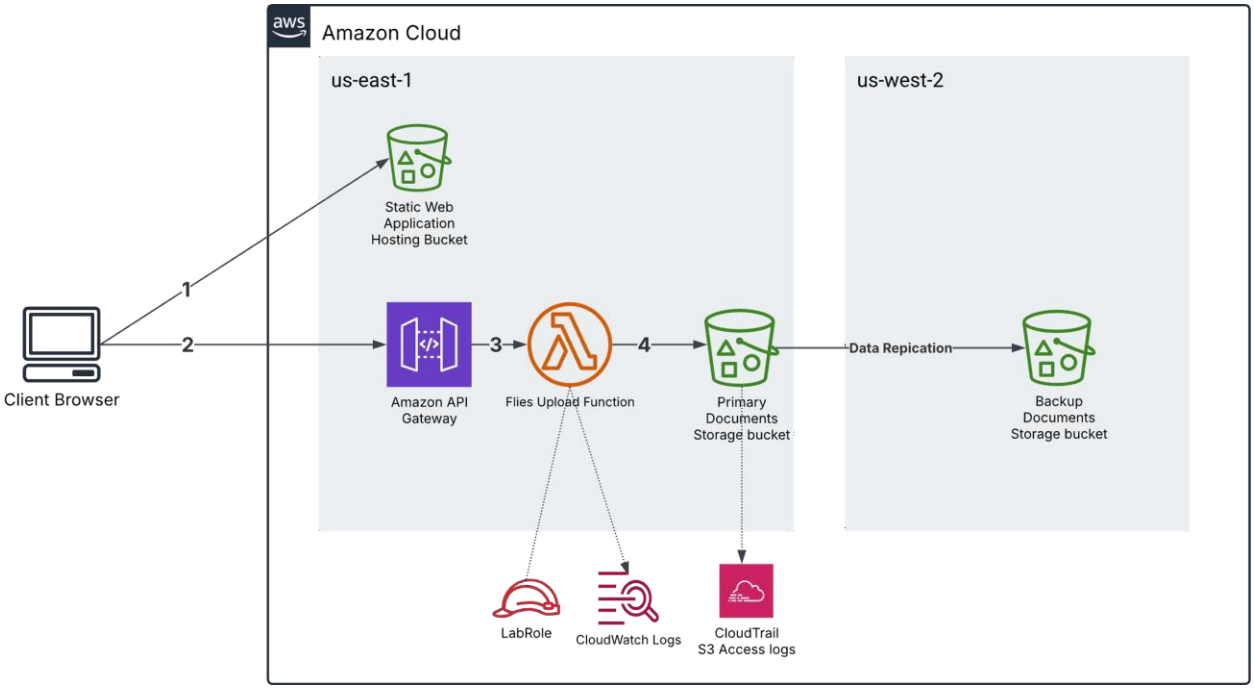
Skills Compétences Canada Ontario

Skills Ontario: Cloud Computing Contest Sample Challenge



Tasks and Duration

● The assignment features 3 modules. Each module can be implemented using AWS Management console or using Infrastructure as Code (Terraform or CloudFormation) for the extra marks.

Restrictions and Limitations

● Region Limitations: Operations must be confined to us-east-1 for primary activities and us-west-2 for disaster recovery replication.
● Budget Constraints: Design and implement solutions within a $10 budget for the duration of the task to avoid exceeding the allocated AWS budget.

Module 1: Secure Document Storage System (Gateway DataSecure Inc.)

Scenario

Gateway DataSecure Inc., a cybersecurity firm, needs a secure, scalable document storage

system for sensitive client data. The system must comply with regulations (e.g., HIPAA, GDPR) and support encryption, lifecycle management, versioning, and disaster recovery. You will implement this system using Amazon S3 and provision it using Infrastructure as Code (IaC).

Objective:

Build and provision a secure document storage solution using AWS S3 that meets the following requirements:

- Enforce encryption at rest (SSE-KMS) and in transit (HTTPS)
- Implement lifecycle policies for cost optimization
- Enable versioning and cross-region replication (CRR)
- Set up monitoring via AWS CloudTrail
- Automate deployment using Terraform or CloudFormation

Tasks

Task 1: Secure S3 Bucket

- Create a bucket named: `gateway-datasecure-inc-docs-XXXX` in `us-east-1`
- Enable SSE-KMS encryption using a KMS key (with key rotation)

Task 2: Bucket Policy

- Enforce HTTPS-only access via a bucket policy
- Grant permissions to the pre-provided `LabRole`

Task 3: Lifecycle Management

- Add rules to:
  - Transition data to S3 Standard-IA after 30 days
  - Transition to S3 Glacier after 90 days

Task 4: Disaster Recovery

- Enable versioning
- Create a replica bucket named `gateway-datasecure-inc-crr-XXXX` in `us-west-2`
- Configure cross-region replication using the `ReplicationRole`

Task 5: Monitoring and Audit

- Enable AWS CloudTrail logging for all S3 operations on the primary bucket

Infrastructure as Code Implementation (Module Extension)

You must implement all of the above using Infrastructure as Code:

- Use either Terraform or CloudFormation
- Include modular code (grouped logically by purpose)
- Reference provided IAM role ARNs (you are not allowed to create new IAM roles)
- Follow naming conventions and region constraints

Submission Checklist

Submit the following:

1. IaC Code:
    a. Terraform: `main.tf`, `variables.tf`, etc.
    b. OR CloudFormation: YAML/JSON templates
2. `README.md or` PDF including:
    a. Tool choice and rationale
    b. Deployment instructions
    c. Assumptions (e.g., manual steps, IAM ARNs)
3. Screenshots or CLI output confirming:
    a. Bucket creation
    b. KMS key setup
    c. Lifecycle rules
    d. Replication setup
    e. CloudTrail logging

Validation

Before submission, verify that:

- SSE-KMS encryption is active
- HTTPS-only access is enforced
- Lifecycle rules are correctly implemented
- Cross-region replication is operational
- CloudTrail is logging S3 API activity
- IAM roles were not created manually
- Naming, region, and budget rules are followed

Module 2: File Upload Interface for "Gateway DataSecure Inc Portal" via simple Web Application

Background

Following the successful implementation of a secure storage service in Module 1, "Gateway DataSecure Inc." seeks to enhance its digital capabilities by integrating a simple, user-friendly interface for its clients. This interface will facilitate the secure uploading of various document types (such as .doc, .txt, .pdf, .ppt) directly to the AWS S3 bucket, ensuring that client interactions are both intuitive and secure. The company aims to streamline the process by which clients submit sensitive documents, which are often related to critical cases and require stringent confidentiality and swift handling.

Objective

Design and deploy a secure, browser-based interface that allows clients to upload documents with metadata to a structured S3 path. You will deploy the interface as a static website, configure secure uploads, and implement infrastructure using Infrastructure as Code (IaC).

Detailed Requirements

1. Create and configure the S3 Bucket to host your static web application

Create a new S3 bucket with the name: “skills-ontario-2025-[yourname]-web-v1". Replace `[yourname]` with your actual name (e.g., `johnsmith`).

2. Deploy the Web Application

The folder upload_to_s3 contains a simple static web app.
Upload all files from the upload_to_s3 folder to your skills-ontario-2025-[yourname]-web-v1 S3 bucket.

Confirm that the application loads correctly in a browser using the static website endpoint.

3. Modify the Application to Upload Files to S3

Configure the app to upload user-selected files skills-ontario-2025-[yourname]-web-v1 bucket.
The application must:

- Accept user file input
- Collect client ID, case ID, and document type
- Upload the file to a structured key like:uploads/{clientId}/{caseId}/{documentType}/{filename}

4. Implementation with Infrastructure as Code

Use Terraform or CloudFormation to:

- Create the S3 bucket with static website hosting.

- Configure all bucket settings
- Deploy the static website files to S3 using IaC (Terraform aws_s3_bucket_object, or CloudFormation AWS::S3::Bucket + custom resources).

5. Submission Requirements

In your report, include the following:

- A screenshot of the web app in your browser showing the success message.
- A screenshot of your S3 bucket file listing showing the uploaded document(s).
- A screenshot of successful deployment using IaC

Create a table that describes all modifications you made to:

- The HTML/JS application
- AWS infrastructure (S3 policy, CORS, etc.)

Example (for the demonstration purposes only, these are _not_ real changes):

Component	Change Made	Reason
Index_to_s3.html	Updated script.js with bucket name	Ensure correct upload target
script_to_s3.js	Added encryption of the object in transit	Objects are uploaded to S3 via insecure channels

Write a short paragraph discussing:

- What are the security risks with the current solution (e.g., hardcoded credentials, public bucket)?
- What would be better practice?
- Choose one improvement below and implement it (e.g., switch to pre-signed URLs, restrict access, move credentials to environment variables).
    - Replace hardcoded credentials with temporary credentials using AWS STS.
    - Implement file validation in the frontend (limit types and sizes).
    - Use pre-signed URLs to avoid exposing bucket write permissions.
    - Restrict S3 bucket access using a bucket policy with allowed referrers.

Submission

Submit the following:

- Markdown report (README.md or .pdf) containing:
- Screenshots
- Change table

- Security discussion and enhancement
- Deployed and working website (link to your S3 static website endpoint)

Module 3: File Upload Portal with Environment-Specific API Gateway and Metadata Storage in DynamoDB

Scenario

Gateway DataSecure Inc. is expanding its file submission system to support multiple environments (development and production) with secure and auditable document uploads. Clients will submit sensitive documents through a simple web portal. Uploaded files will be stored in S3, and metadata will be processed and recorded in DynamoDB to support audit, retrieval, and compliance workflows.

Objective

Build and deploy a secure, environment-aware document upload system using the `upload_via_gateway/` folder. Your solution must include:

- A frontend hosted in S3
- Backend implemented with **API Gateway and Lambda** for both **dev** and **prod** environments
- Metadata for each uploaded file stored in DynamoDB
- Proper IAM permissions, error handling, and logging across the stack

Requirements

1. Provided Code and Environments

- Use code in the `upload_via_gateway/` folder:
  - `frontend/`: web portal for file uploads
  - `backend/`: Lambda function template for processing uploads
- Set up two environments:
  - `dev`: used for testing
  - `prod`: simulates a production deployment

2. Frontend Hosting

- Host the `frontend/` code in an S3 bucket with static website hosting enabled
- The portal must:
  - Validate allowed file types: `.doc`, `.txt`, `.pdf`, `.ppt`
  - Limit file size to 2MB
  - Require metadata input fields: `ClientID`, `CaseID`, `DocumentType`
  - Display upload result (success or error)

3. API Gateway Setup

- Create an **API Gateway** with:
  - Two stages: `dev` and `prod`
  - Separate Lambda integrations for each stage (use environment variables or aliases)
  - CORS enabled
  - HTTPS enforced
  - Logging enabled per stage

4. Lambda Processing (Per Environment)

Each environment must have its own Lambda function that:

- Validates file size and type
- Uploads the file to S3 under:

`uploads/{ClientID}/{CaseID}/{DocumentType}/{filename}`

- Extracts and stores metadata in DynamoDB:
  - `ClientID` (partition key)
  - `CaseID` (sort key)

    - `DocumentType`
    - `Filename`
    - `FileSize` (bytes)
    - `UploadTime` (ISO format)
    - `UploadLocation` (S3 path or URL)
- Implements error handling:
    - Rejects missing or invalid inputs
    - Logs processing steps and errors to CloudWatch
    - Returns appropriate HTTP response codes

5. DynamoDB Metadata Table

- Create a DynamoDB table named `DocumentMetadata`
- Keys:
    - **Partition key**: `ClientID`
    - **Sort key**: `CaseID`
- Store one record per uploaded file with all metadata attributes
- Ensure efficient querying by `ClientID` and `CaseID`

6. Security and IAM

- Use HTTPS for all frontend-to-backend communication
- Do not hardcode credentials
- Use the provided `LabRole` for all Lambda permissions
- Grant only required access to S3 and DynamoDB

Tasks

Task 1: Static Web Hosting

- Upload `frontend/` to an S3 bucket
- Configure static website hosting
- Ensure the portal sends file uploads to the correct API Gateway endpoint (based on stage)

Task 2: API Gateway and Stages

- Create REST API with POST `/upload` route
- Deploy two stages: `dev` and `prod`
- Link each stage to its corresponding Lambda function

Task 3: Environment-Specific Lambda Functions

- Deploy a Lambda function for each environment
- Use environment variables for configuration (e.g., table name, bucket name)
- Ensure the function handles:
    - File validation
    - S3 upload
    - Metadata insertion into DynamoDB
    - Logging

Task 4: DynamoDB Table Setup

- Create a table with:
    - Partition key: `ClientID`
    - Sort key: `CaseID`
- Populate metadata with each upload
- Use `UploadTime` and `Filename` as additional attributes

Task 5: Test and Document

- Test both `dev` and `prod` workflows
- Submit the required deliverables

4. Implementation with Infrastructure as Code

Use Terraform or CloudFormation to:

- Create the S3 bucket with static website hosting.
- Create API Gateway woth dev and prod environments
- Create and deploy lambda function

Submission Requirements

- URLs:
    - S3-hosted frontend
    - API Gateway endpoints (`dev` and `prod`)
- Screenshots:
    - Successful upload through the UI
    - S3 file structure
    - DynamoDB entry
- Source code:
    - Lambda functions (dev and prod)
    - Modified frontend (if applicable)
    - Infrastructure as code implementation

- README.md or PDF:
    - Configuration steps
    - How dev and prod are separated
    - Metadata schema description
    - API sample request/response
    - Summary of security and logging implementation