**Project Requirements:**

You must utilize the following services and deploy the project with terraform: EC2, S3,IAM, Workspaces, Security Hub

**Project Title: Secure File Sharing and Analysis Platform**

**Project Description**

This service enables secure file sharing and analysis. The platform will process and summarize sensitive medical documents, ensuring compliance with security standards and providing alerts for high-severity security findings.

**Real-World Application:**

This project can serve as a prototype for a secure document management and analysis system for a healthcare organization. It ensures that sensitive files (e.g., medical records) are securely uploaded, processed, and analyzed while adhering to strict security standards.

**Architecture and Workflow Description:**

At this time the admin user, me, will upload all the documents to the original S3 bucket. Once the file is uploaded to the S3 bucket an S3 Event will trigger a Lambda function. This Lambda function will deliver the S3 document to the EC2 instance. The EC2 instance is required to be running 24/7. The EC2 instance will process and summarize the medical document using spaCy. SpaCy is an open-source software library for advanced natural language processing. At the time of writing this I have elected to comment out the portion of the shell script where spaCy will summarize the document. I was having difficulty implementing that portion of the project and skipping over it allows me to showcase my ability to implement with all the services mentioned above. I will circle back and learn to work with spaCy in the future. Once it has summarized the medical document it will send it to the summarized S3 bucket. End users at the medical office will use AWS Workspaces to login and securely access both the original medical documents and the summarized documents.

Considering the stringent security compliance for healthcare information we will use a combination of IAM, AWS KMS, AWS Config, Security Hub, Event Bridge, and SNS to harden our infrastructure. IAM will ensure users and services can communicate with one another while maintaining the principle of least privilege. AWS KMS will provide our file encryption. This is the superior choice to SSE S3 for this application as KMS can provide an audit trail for all KMS actions. AWS Config must be enabled to utilize Security Hub. Security Hub will monitor our infrastructure for findings based on AWS Foundational Security Best Practices, CIS AWS Foundations Benchmarks, and PCI DSS. When findings occur, we have created an EventBridge Rule to import the finding and send the details to SNS where our security team can be notified and review the findings.

CloudWatch is currently being implemented for troubleshooting communication difficulties.

**AWS Services Used:**

1. **EC2:** Forprocessing and summarizing the uploaded medical reports.
2. **S3:** For storing the original reports, summarized reports, and lambda package dependencies as spaCy alone exceeds the 50mb limit.
3. **IAM:** For managing access permissions.
4. **AWS KMS:** For encrypting sensitive data.
5. **AWS Config:** For monitoring configuration compliance.
6. **Workspaces:** For providing a virtual desktop environment for deeper analysis.
7. **Security Hub:** For monitoring and ensuring security compliance.
8. **EventBridge:** For routing Security Hub findings to SNS.
9. **SNS:** For sending alerts on high-severity security findings.
10. **Lambda**: For processing documents when they are uploaded to S3.
11. **CloudWatch**: For collecting logs at various stages and troubleshooting failed communications. At this stage we’re only implementing it for the Lambda Function, but as needs grow we will expand log collection.

**Steps to Implement the Project:**

1. **Setup Terraform:**
   * Create a new directory for your project and initialize a Terraform configuration file.
   * Define your AWS provider in main.tf.
2. **Provision S3 Buckets and IAM Roles**
   * Add the S3 bucket configurations to main.tf.
   * Configure IAM roles, policies, and instance profiles.
3. **Launch EC2 Instances:**
   * Use Terraform to launch EC2 instances for processing files.
   * Write a simple script that the EC2 instance will use to process files from the S3 bucket.
4. **Setup AWS Workspaces:**
   * Use Terraform to provision AWS Workspaces.
   * Ensure that the Workspaces can access the S3 bucket and EC2 instances.
5. **Configure AWS Security Hub:**
   * Enable AWS Security Hub to monitor your AWS environment.
   * Set up necessary compliance checks and alerts.
6. **Step 6: Setup AWS KMS**
   * Create a KMS key for encrypting S3 buckets.
7. **Step 7: Setup AWS Config**
   * Enable AWS Config to monitor resource configurations.
8. **Setup** **EventBridge and SNS for Security Alerts**
   * Create an EventBridge rule to capture Security Hub findings and send them to SNS.
9. **EC2 Setup Script (file\_processor.sh):**
   * Installs dependencies and sets up the environment.
   * Creates the process\_reports.sh script to process documents.
10. **Lambda Function (lambda\_function.py):**
    * Triggers the EC2 instance to run process\_reports.sh with the S3 object key as an argument.
    * Add Cloudwatch Log permissions and enable logging for the Lambda Function
11. **Document the Project:**
    * Write clear documentation on how the system works.
    * Include instructions on how to upload files, how the processing works, and how to access the Workspaces.

**Lessons Learned and Modifications to my original plan:**

1. I originally decided to deploy this section using terraform but determined that probably wouldn’t be the best real-world application for the S3 buckets and security policies. The security policies should be maintained in their own terraform project and this bucket would never be deleted. I left the creation of these resources in the code but commented them out.
2. When I originally implemented the IAM policies I had difficulties getting one of them to correctly translate using the jsonencode() function. I was getting a successful resource creation notification from my Terraform Apply, but the resource was not actually in AWS. This led me to learn about the “EOF” functionality for terraform. I used EOF to create a multiline string of the JSON policy I generated within AWS to import the code directly without translation.
3. I created an AWS config recorder through via terraform without realizing you can only view and interact with config recorders through the AWS CLI. This created some confusion and trouble for me as I then tried to create the recorder through the front end and hit the error for a regional limit of 1 config recorder. Only after reading the official AWS config documentation did I learn that you can only view and interact with the recorders through the CLI. I used the terraform destroy -target command to destroy my recorder and start over.
4. I could not get the AWS Config Recorder permissions set up for the service to be able to write to the bucket. The s3:GetBucketAcl permission is needed for AWS Config because, during the setup and operation of the AWS Config delivery channel, AWS Config verifies that it has the necessary permissions to write to the S3 bucket by checking the bucket's ACL (Access Control List). I was only giving it the s3:PutObject permission initially.
5. When I decided to set up S3 event bridge notifications that trigger a lambda function to
6. When you create an IAM Role with a trust relationship for the ec2 service an instance profile ARN is generated. This instance profile is what you must attach to the EC2 instance in terraform. I was attaching the role itself for a while which was giving me permission errors preventing the ec2 instance from generating.
7. I couldn’t get my Lambda function to trigger when uploading a file to the S3 bucket so I needed to enable cloudwatch logs for the Lambda function to have a way to troubleshoot.
8. I had a lot of difficulties getting the Lambda function to communicate with the EC2 instance. Eventually realizing the Lambda function uses the SSM agent to communicate with the EC2 instance led me to realize the SSM agent must be installed and running on the EC2 instance. I added lines for this to the shell script that is pulled into the user data of the EC2 instance. The SSM command still wasn’t reaching my EC2 instance so I learned on Stack Overflow that you must have certain SSM permissions to accomplish this. The SSMManagedNode Policy should be attached to your EC2 role. After a short period of time you will be able to see the EC2 instance in your SSM Inventory.
9. I then had issues with any logs being generated or showing any activity after I knew the SSM command was reaching the EC2 instance. I cut out the prior portion of my process and began running the SSM command directly through my EC2 instance connect. This allowed me to save the time of deleting, re-uploading, and executing the S3 event/Lambda portion. Once I ran the SSM command I used the following ssm status checks to see what was happening with the command.
   1. aws ssm list-commands --region us-east-1
   2. aws ssm list-command-invocations --command-id 3741697f-5100-451d-8238-22c74f9bf6c6 --details --region us-east-1

These commands showed me my EC2 instance was attempting to fetch the file from the S3 bucket, but did not have the KMS key access I was using to encrypt the files in the S3 bucket.

**Improving the project:**

**Hardening the Infrastructure**

* IP Access Controls for Work Spaces – Lock down the Ips that access the workstations.
* Can I migrate the bucket policy from Lessons learned 4 to IAM roles rather than the bucket policy?