**How to handle IoT secret files in CI CD flow? (Generic Info)(For project specific requirement, refer to next question)**

Handling sensitive files like IoT certificates (e.g., device.cert.pem, private.key) in a CI/CD pipeline requires a strategy that prevents committing them to Git while ensuring they are available during deployment.

Here are the three most common and secure ways to handle this.

### 1. Base64 Encoding (Quickest Solution)

Since most CI/CD secret managers (GitHub Secrets, GitLab Variables) only handle text strings, you can encode your binary or multiline certificate files into a single-line Base64 string.

Step 1: Encode your files locally

Run this command on your local machine for each file:

Bash

# Linux/Mac  
base64 -i ./certs/device.cert.pem -o cert\_encoded.txt  
base64 -i ./certs/private.key -o key\_encoded.txt  
  
# Windows PowerShell  
[Convert]::ToBase64String([IO.File]::ReadAllBytes("./certs/device.cert.pem"))

Step 2: Add to CI/CD Secrets

Copy the content of cert\_encoded.txt and key\_encoded.txt into your CI/CD secrets (e.g., IOT\_CERT\_BASE64, IOT\_KEY\_BASE64).

Step 3: Decode in Pipeline

In your pipeline script (GitHub Actions YAML or .gitlab-ci.yml), decode them back into files before Terraform runs.

*Example (GitHub Actions):*

YAML

steps:  
 - name: Decode IoT Certificates  
 run: |  
 mkdir -p ./certs  
 echo "${{ secrets.IOT\_CERT\_BASE64 }}" | base64 -d > ./certs/device.cert.pem  
 echo "${{ secrets.IOT\_KEY\_BASE64 }}" | base64 -d > ./certs/private.key  
  
 - name: Terraform Apply  
 run: terraform apply -auto-approve

### 2. Secure Storage (AWS S3 + Encryption)

If you have many certificates or they rotate frequently, storing them in a secure remote backend is better than clogging up your CI/CD secrets.

Step 1: Upload to S3 (Private Bucket)

Upload your certs to a private S3 bucket that only your CI/CD runner's IAM role can access.

Step 2: Pull in Pipeline

Add a step in your pipeline to fetch them before Terraform runs.

YAML

steps:  
 - name: Download Certs from S3  
 run: aws s3 cp s3://my-secure-iot-bucket/certs/ ./certs/ --recursive

### 3. Terraform Data Sources (Best for AWS IoT)

If these are AWS IoT certificates generated by AWS, you don't need to store the files at all. You can use Terraform data sources or let Terraform manage the state.

If Terraform created them, the certificate content is already in your terraform.tfstate.

* **Warning:** This creates a security risk if your state file is not encrypted. Ensure your S3 backend for Terraform state has **Server-Side Encryption (SSE)** enabled.

### Summary Recommendation

* **For simple projects:** Use **Method 1 (Base64)**. It is easy to implement and requires no extra infrastructure.
* **For production/enterprise:** Use **Method 2 (S3)** or a dedicated Secret Manager (AWS Secrets Manager / HashiCorp Vault) to inject them at runtime.

**How to handle the IoT certificates within current code and also should it work in CI CD environment?**

To handle IoT certificates securely in your CI/CD pipeline without changing your current Terraform logic (which expects files to exist locally for upload), use the **"Inject-at-Runtime"** strategy.

[cite\_start]Your code in modules/s3\_config/main.tf currently looks for files in ../../resources/certs/. If these files aren't there during terraform plan, the pipeline will fail.

Here is the step-by-step solution:

### 1. The Strategy

Do not commit the certificates to Git. Instead, store them as **CI/CD Secrets** and recreate the files on the fly right before Terraform runs.

### 2. Implementation Steps

#### Step A: Base64 Encode Your Certificates (Local Machine)

Since secrets are usually text strings, encoding your binary/multiline certificate files into Base64 is the safest way to store them.

Run these commands in your terminal to get the strings:

Bash

# MacOS / Linux  
base64 -i resources/certs/device-certificate.pem.crt  
base64 -i resources/certs/private.pem.key  
base64 -i resources/certs/AmazonRootCA1.pem  
  
# Windows PowerShell  
[Convert]::ToBase64String([IO.File]::ReadAllBytes("resources/certs/device-certificate.pem.crt"))  
# (Repeat for the other two files)

#### Step B: Add Secrets to Your CI/CD (e.g., GitHub/GitLab)

Go to your repository settings and add these three secrets with the output from Step A:

* TF\_VAR\_cert\_base64\_device
* TF\_VAR\_cert\_base64\_key
* TF\_VAR\_cert\_base64\_root

#### Step C: Update CI/CD Pipeline Configuration

Add a step **before** your terraform init/plan command to decode these secrets back into the file paths your Terraform code expects.

**Example: GitHub Actions Workflow (.github/workflows/deploy.yml)**

YAML

jobs:  
 deploy:  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout Code  
 uses: actions/checkout@v3  
  
 # ---------------------------------------------------------  
 # NEW STEP: Inject Certificates for Terraform to Upload  
 # ---------------------------------------------------------  
 - name: Inject IoT Certificates  
 run: |  
 # Create the directory your Terraform module expects  
 mkdir -p terraform\_v5/resources/certs  
  
 # Decode secrets into the specific filenames defined in your variables.tf  
 echo "${{ secrets.TF\_VAR\_cert\_base64\_root }}" | base64 -d > terraform\_v5/resources/certs/AmazonRootCA1.pem  
 echo "${{ secrets.TF\_VAR\_cert\_base64\_device }}" | base64 -d > terraform\_v5/resources/certs/device-certificate.pem.crt  
 echo "${{ secrets.TF\_VAR\_cert\_base64\_key }}" | base64 -d > terraform\_v5/resources/certs/private.pem.key  
  
 # Verify files exist (for debugging logs)  
 ls -l terraform\_v5/resources/certs/  
  
 - name: Terraform Init  
 run: terraform init  
 working-directory: terraform\_v5/envs/dev  
  
 - name: Terraform Apply  
 run: terraform apply -auto-approve  
 working-directory: terraform\_v5/envs/dev

### 3. Verify Your Terraform Configuration

Ensure your envs/dev/terraform.tfvars (or variables.tf defaults) matches the filenames you are creating in the CI script.

[cite\_start]Looking at your terraform\_v5/envs/dev/variables.tf:

Terraform

variable "cert\_files" {  
 type = map(string)  
 default = {  
 root\_ca = "AmazonRootCA1.pem"  
 device\_cert = "device-certificate.pem.crt"  
 private\_key = "private.pem.key"  
 }  
}

[cite\_start]

**Result:** The CI script creates exactly these filenames, so your existing Terraform code will successfully find them, calculate their MD5 hash, and upload them to S3 without any code changes.

### 4. Important: .gitignore

To prevent accidentally committing these files locally, ensure your .gitignore includes:

Plaintext

terraform\_v5/resources/certs/\*.key  
terraform\_v5/resources/certs/\*.crt

For example, How to handle the generated text file - root\_ca\_encoded.txt?

You need to copy the content of root\_ca\_encoded.txt and save it as a Secret in your CI/CD platform (like GitHub or GitLab).

Here is the exact step-by-step:

**1. Open and Copy the Content**

Open root\_ca\_encoded.txt on your computer and copy the entire long string of random characters. It should look something like LS0tLS1CRUdJTiBDRVJUSUZJQ0....

**Command Line Tip:**

* **Mac/Linux:** cat root\_ca\_encoded.txt | pbcopy (copies directly to clipboard)
* **Windows:** type root\_ca\_encoded.txt | clip

**2. Create the Secret in GitHub/GitLab**

Go to your repository in your browser.

* **GitHub:** Settings $\rightarrow$ Secrets and variables $\rightarrow$ Actions $\rightarrow$ **New repository secret**.
* **GitLab:** Settings $\rightarrow$ CI/CD $\rightarrow$ Variables $\rightarrow$ **Add variable**.

**3. Name the Secret**

Name it exactly as it appears in your pipeline YAML file. Based on the previous example, use:

**TF\_VAR\_cert\_base64\_root**

**4. Paste and Save**

Paste the copied string into the "Value" field and save.

**Why do this?**

When your pipeline runs this command:

Bash

echo "${{ secrets.TF\_VAR\_cert\_base64\_root }}" | base64 -d > terraform\_v5/resources/certs/AmazonRootCA1.pem

It effectively says: *"Take the secure string we saved in Step 4, decode it back into a real file, and put it exactly where Terraform is looking for it."*