How do you handle secrets and sensitive data in CloudFormation templates?

Security is a top priority for me, especially when working with financial institutions in India where RBI compliance and data sensitivity are critical.
The core principle I follow is: *never hardcode secrets into CloudFormation templates* — because they're usually version-controlled in Git, and that can expose sensitive data.
I use **three main approaches** based on the use case:

1. **AWS Systems Manager Parameter Store**
   For simple config and secure values like DB passwords.
   I use SecureString parameters, encrypted with KMS. These are referenced in templates using !Sub with the ssm-secure syntax.
   It's cost-effective and integrates easily.
2. **AWS Secrets Manager**
   For more complex secrets like API keys and DB credentials — especially when **automatic rotation** is required.
   I define secrets in CloudFormation and reference specific keys at deployment. I also configure rotation rules using Lambda.
3. **CloudFormation Parameters with NoEcho**
   CloudFormation Parameters with the NoEcho property set to true are used to prevent sensitive information from being displayed in plain text in the AWS CloudFormation console, API calls, or stack events.

In terms of security best practices:

- I follow **least privilege access** by restricting IAM roles to access only the required parameters.
- I separate secrets by environment (like /dev/, /prod/) to avoid accidental cross-access.
- It is also possible to **encryption at rest** using customer-managed KMS keys.

In summary, I choose the right tool based on the complexity of the secret and compliance needs — and always aim to automate securely without compromising visibility or control.

Here's a **CloudFormation example using an existing secret** from AWS Secrets Manager. This is useful when your secret is **already created manually or by another stack**, and you simply want to reference it in your template.

✅ **CloudFormation YAML: Use Existing Secret in RDS**

yaml

```
AWSTemplateFormatVersion: "2010-09-09"
Parameters:
  ExistingSecretArn:
    Type: String
    Description: ARN of the existing Secrets Manager secret

Resources:

  MyDBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
      DBInstanceIdentifier: mydb-instance
      Engine: mysql
      DBInstanceClass: db.t3.micro
      AllocatedStorage: 20
      MasterUsername: !Sub
"{{resolve:secretsmanager:${ExistingSecretArn}:SecretString:username}}"
      MasterUserPassword: !Sub
"{{resolve:secretsmanager:${ExistingSecretArn}:SecretString:password}}"
```

📄 **Notes:**

- This expects a secret with key-value structure like:

json

```
{
  "username": "admin",
  "password": "your-password"
}
```

- Pass the **secret ARN** as a parameter when deploying the stack.

🚀 **Deployment Example (CLI):**

bash

```
aws cloudformation deploy \
  --template-file db-stack.yaml \
  --stack-name MyDBStack \
  --parameter-overrides ExistingSecretArn=arn:aws:secretsmanager:ap-south-
1:123456789012:secret:my-db-secret-AbCdEf
```

Which Parameter Store type is encrypted using AWS KMS?
A. String
B. StringList
C. SecureString
D. EncryptedList
Answer: C

Which AWS service allows automatic rotation of secrets such as database credentials?
A. CloudTrail
B. Secrets Manager
C. Parameter Store
D. IAM
Answer: B

Which feature of Secrets Manager is particularly helpful for compliance requirements?
A. IAM Roles
B. SecureString support
C. Automatic rotation
D. Parameter hierarchy
Answer: C

Which CloudFormation parameter property hides sensitive input during stack deployment?
A. Type: Hidden
B. SecureInput: True
C. NoEcho: true
D. Mask: true
Answer: C