compromises the encryption key, they can go back and decrypt all the secrets that were ever encrypted with it.

- The ability to audit who accessed secrets is minimal. If you're using a cloud key management service (e.g., AWS KMS), it will likely maintain an audit log of who used an encryption key, but you won't be able to tell what the key was actually used for (i.e., what secrets were accessed).
- Most managed key services cost a small amount of money. For example, each key you store in AWS KMS costs $1/month, plus $0.03 per 10,000 API calls, where each decryption and encryption operation requires one API call. A typical usage pattern, where you have a small number of keys in KMS and your apps use those keys to decrypt secrets during boot, usually costs $1–$10/month. For larger deployments, where you have dozens of apps and hundreds of secrets, the price is typically in the $10–$50/month range.
- Standardizing secret management practices is harder. Different developers or teams may use different ways to store encryption keys or manage encrypted files, and mistakes are relatively common, such as not using encryption correctly or accidentally checking in a plain-text file into version control.

Secret stores

The third technique relies on storing your secrets in a centralized secret store.

Some of the more popular secret stores are AWS Secrets Manager, Google Secret Manager, Azure Key Vault, and HashiCorp Vault. Let's look at an example using AWS Secrets Manager. The first step is to store your database credentials in AWS Secrets Manager, which you can do using the AWS Web Console, as shown in Figure 6-2.

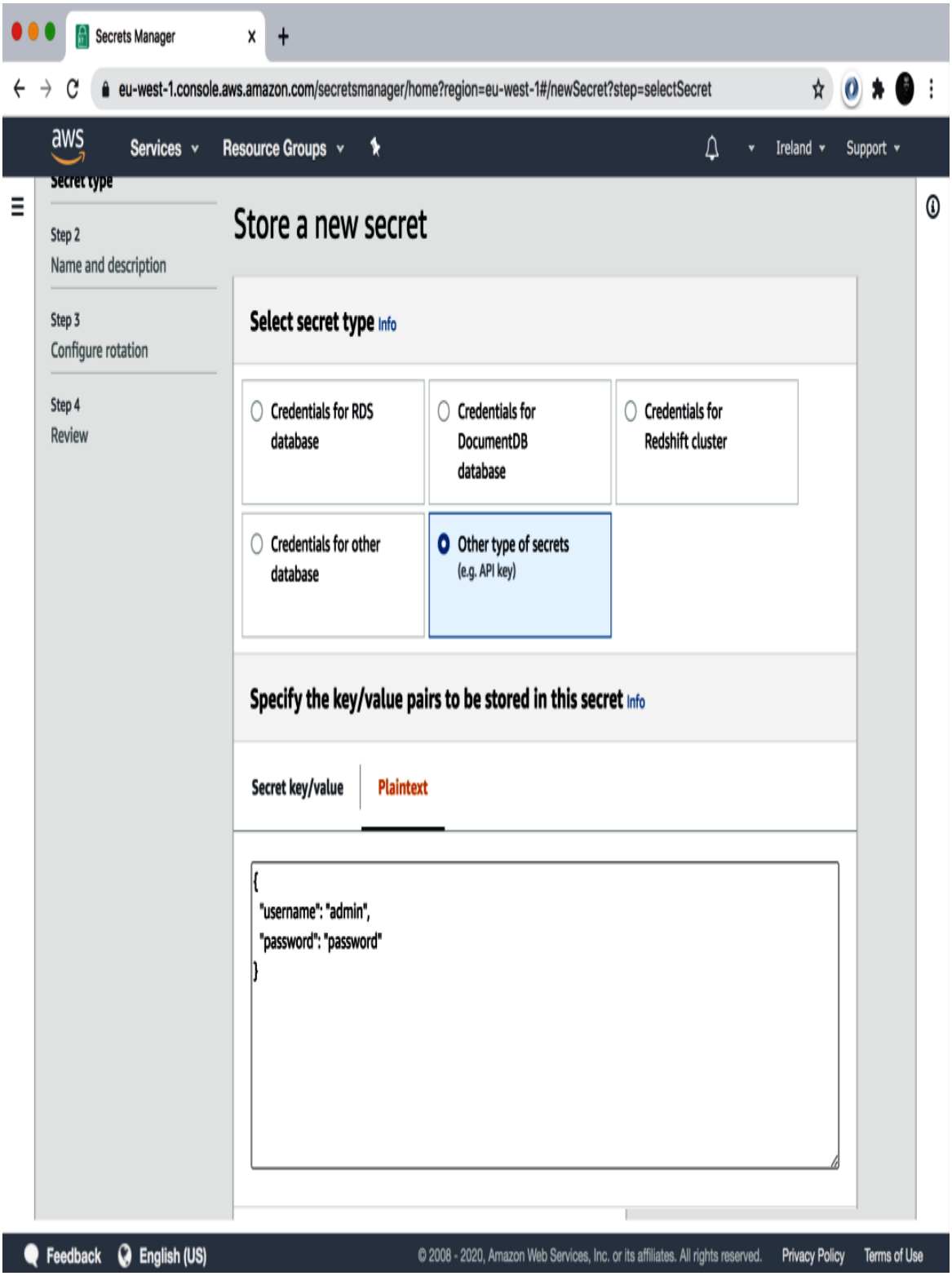

Figure 6-2. Store secrets in JSON format in AWS Secrets Manager.

Note that the secrets in Figure 6-2 are in a JSON format, which is the recommended format for storing data in AWS Secrets Manager.

Go to the next step, and make sure to give the secret a unique name, such as `db-creds`, as shown in Figure 6-3.

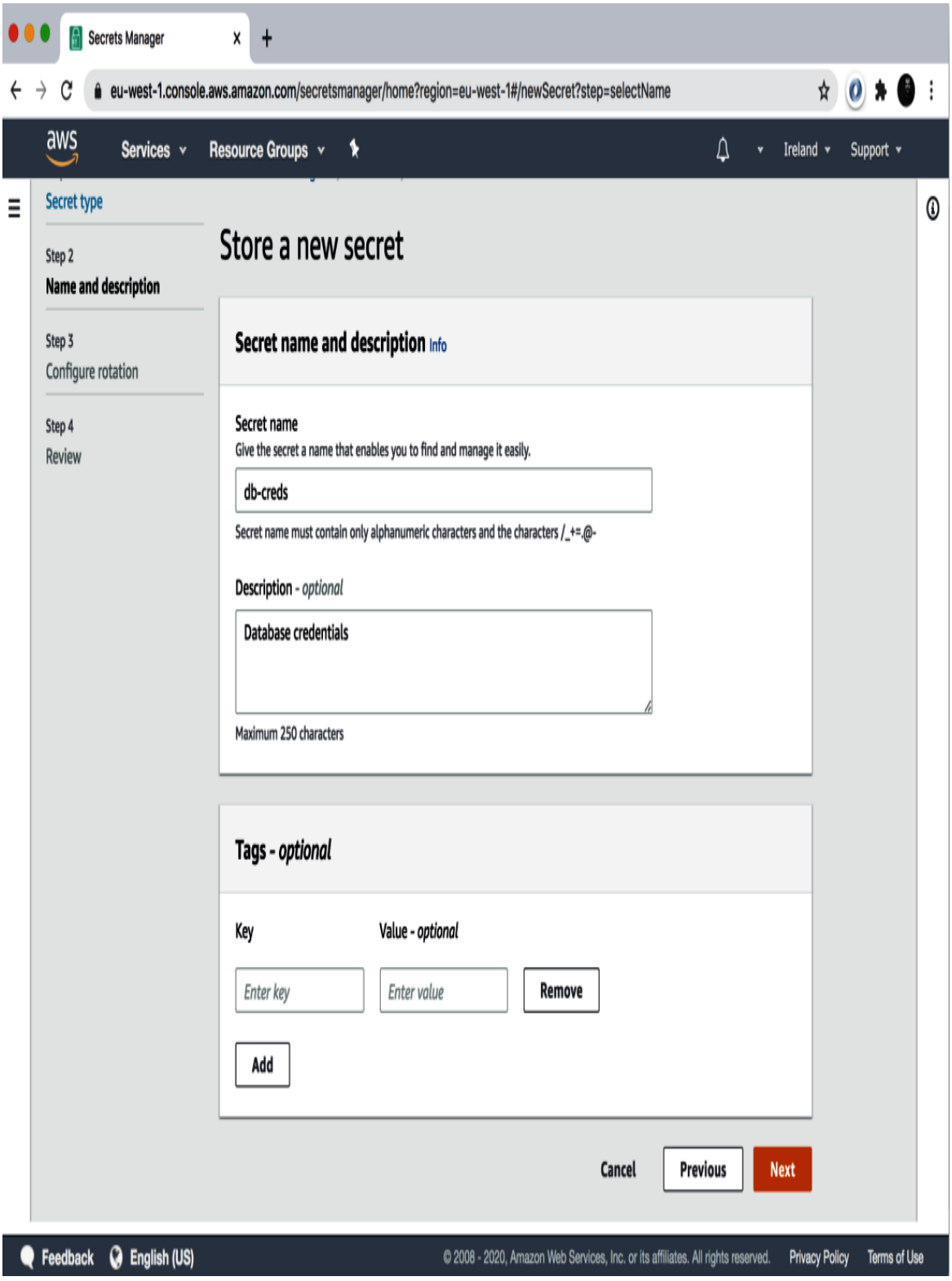

Figure 6-3. Give the secret a unique name in AWS Secrets Manager.

Click Next and Store to save the secret. Now, in your Terraform code, you can use the `aws_secretsmanager_secret_version` data source to read the `db-creds` secret:

```
data "aws_secretsmanager_secret_version" "creds" {
  secret_id = "db-creds"
}
```

Since the secret is stored as JSON, you can use the `jsondecode` function to parse the JSON into the local variable `db_creds`:

```
locals {
  db_creds = jsondecode(
    data.aws_secretsmanager_secret_version.creds.secret_string
  )
}
```

And now you can read the database credentials from `db_creds` and pass them into the `aws_db_instance` resource:

```
resource "aws_db_instance" "example" {
  identifier_prefix   = "terraform-up-and-running"
  engine              = "mysql"
  allocated_storage   = 10
  instance_class      = "db.t2.micro"
  skip_final_snapshot = true
  db_name             = var.db_name

  # Pass the secrets to the resource
  username = local.db_creds.username
  password = local.db_creds.password
}
```

Using secret stores has the following advantages:

- Keep plain-text secrets out of your code and version control system.
- Everything is defined in the code itself. There are no extra manual steps or wrapper scripts required.
- Storing secrets is easy, as you typically can use a web UI.

- Secret stores typically support rotating and revoking secrets, which is useful in case a secret gets compromised. You can even enable rotation on a scheduled basis (e.g., every 30 days) as a preventative measure.
- Secret stores typically support detailed audit logs that show you exactly who accessed what data.
- Secret stores make it easier to standardize all your secret practices, as they enforce specific types of encryption, storage, access patterns, etc.

Using secret stores has the following drawbacks:

- Since the secrets are not versioned, packaged, and tested with your code, configuration errors are more likely, such as adding a new secret in one environment (e.g., staging) but forgetting to add it in another environment (e.g., production).
- Most managed secret stores cost money. For example, AWS Secrets Manager charges $0.40 per month for each secret you store, plus $0.05 for every 10,000 API calls you make to store or retrieve data. A typical usage pattern, where you have several dozen secrets stored across several environments and a handful of apps that read those secrets during boot, usually costs around $10–$25/month. With larger deployments, where you have dozens of apps reading hundreds of secrets, the price can go up to hundreds of dollars per month.
- If you're using a self-managed secret store such as HashiCorp Vault, then you're both spending money to run the store (e.g., paying AWS for 3–5 EC2 Instances to run Vault in a highly available mode) and spending time and money to have your developers deploy, configure, manage, update, and monitor the store. Developer time is very expensive, so depending on how much time they have to spend on setting up and managing the secret store, this could cost you thousands of dollars per month.
- Retrieving secrets is harder, especially in automated environments (e.g., an app booting up and trying to read a database password), as