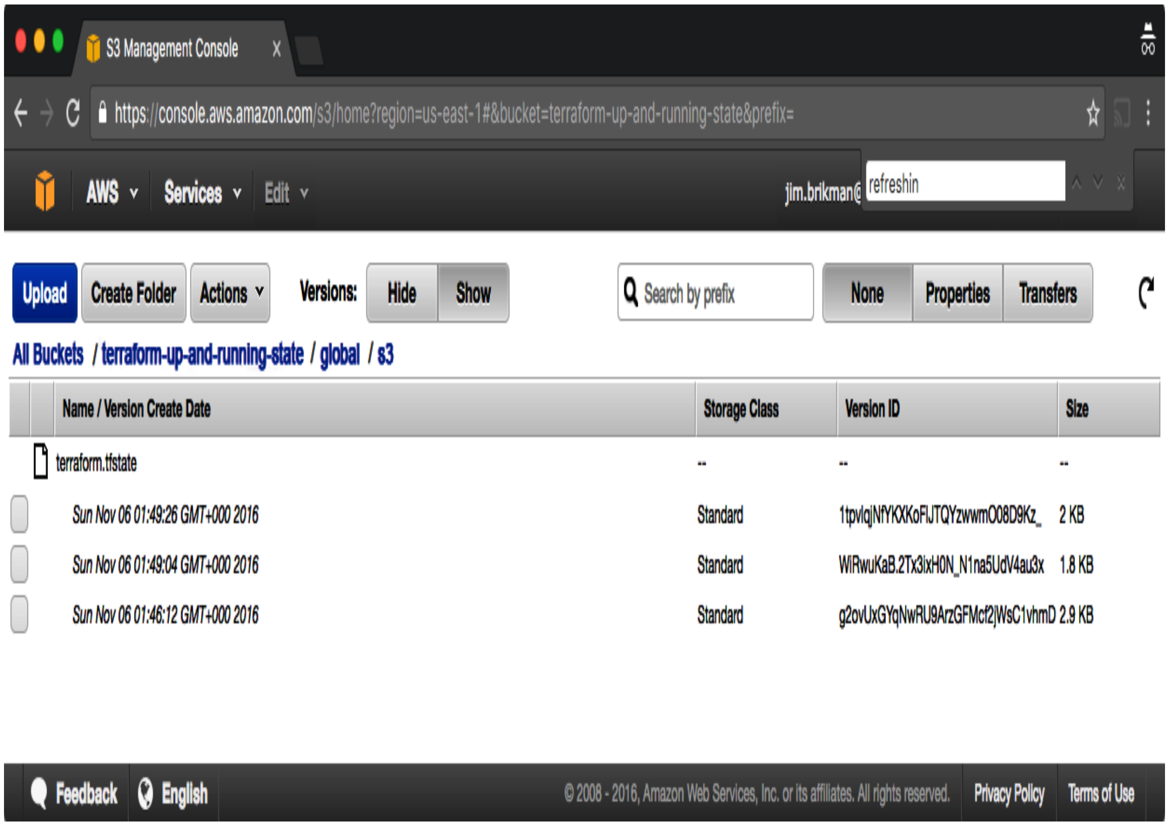
terraform.tfstate file in the S3 bucket, as shown in Figure 3-2.


S3 Management Console
https://console.aws.amazon.com/s3/home?region=us-east-1#&bucket=terraform-up-and-running-state&prefix=
AWS
Services
Edit
jim.brikman@
refreshin
Upload
Create Folder
Actions
Versions:
Hide
Show
Search by prefix
None
Properties
Transfers
All Buckets / terraform-up-and-running-state / global / s3
Feedback
English
© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

Name / Version Create Date	Storage Class	Version ID	Size
terraform.tfstate	--	--	--
Sun Nov 06 01:49:26 GMT+000 2016	Standard	1tpvlqjNfYKXKoFlJTQYzwwmO08D9Kz_	2 KB
Sun Nov 06 01:49:04 GMT+000 2016	Standard	WiRwuKaB.2Tx3ixH0N_N1na5UdV4au3x	1.8 KB
Sun Nov 06 01:46:12 GMT+000 2016	Standard	g2ovUxGYqNwRU9ArzGFMcf2jWsC1vhmD	2.9 KB

Figure 3-2. If you enable versioning for your S3 bucket, every change to the state file will be stored as a separate version.

This means that Terraform is automatically pushing and pulling state data to and from S3, and S3 is storing every revision of the state file, which can be useful for debugging and rolling back to older versions if something goes wrong.

Limitations with Terraform's Backends

Terraform's backends have a few limitations and gotchas that you need to be aware of. The first limitation is the chicken-and-egg situation of using Terraform to create the S3 bucket where you want to store your Terraform state. To make this work, you had to use a two-step process:

1. Write Terraform code to create the S3 bucket and DynamoDB table, and deploy that code with a local backend.

2. Go back to the Terraform code, add a remote `backend` configuration to it to use the newly created S3 bucket and DynamoDB table, and run `terraform init` to copy your local state to S3.

If you ever wanted to delete the S3 bucket and DynamoDB table, you'd have to do this two-step process in reverse:

1. Go to the Terraform code, remove the `backend` configuration, and rerun `terraform init` to copy the Terraform state back to your local disk.

2. Run `terraform destroy` to delete the S3 bucket and DynamoDB table.

This two-step process is a bit awkward, but the good news is that you can share a single S3 bucket and DynamoDB table across all of your Terraform code, so you'll probably only need to do it once (or once per AWS account if you have multiple accounts). After the S3 bucket exists, in the rest of your Terraform code, you can specify the `backend` configuration right from the start without any extra steps.

The second limitation is more painful: the `backend` block in Terraform does not allow you to use any variables or references. The following code will *not* work:

```
# This will NOT work. Variables aren't allowed in a backend
configuration.
terraform {
  backend "s3" {
    bucket         = var.bucket
    region         = var.region
    dynamodb_table = var.dynamodb_table
    key            = "example/terraform.tfstate"
    encrypt        = true
  }
}
```

This means that you need to manually copy and paste the S3 bucket name, region, DynamoDB table name, etc., into every one of your Terraform modules (you'll learn all about Terraform modules in Chapters 4 and 8; for now, it's enough to understand that modules are a way to organize and reuse Terraform code and that real-world Terraform code typically consists of many small modules). Even worse, you must very carefully *not* copy and paste the `key` value but ensure a unique `key` for every Terraform module you deploy so that you don't accidentally overwrite the state of some other module! Having to do lots of copy-and-pastes *and* lots of manual changes is error prone, especially if you need to deploy and manage many Terraform modules across many environments.

One option for reducing copy-and-paste is to use *partial configurations*, where you omit certain parameters from the `backend` configuration in your Terraform code and instead pass those in via `-backend-config` command-line arguments when calling `terraform init`. For example, you could extract the repeated *backend* arguments, such as `bucket` and `region`, into a separate file called *backend.hcl*:

```
# backend.hcl
bucket         = "terraform-up-and-running-state"
region         = "us-east-2"
dynamodb_table = "terraform-up-and-running-locks"
encrypt        = true
```

Only the `key` parameter remains in the Terraform code, since you still need to set a different `key` value for each module:

```
# Partial configuration. The other settings (e.g., bucket,
region) will be
# passed in from a file via -backend-config arguments to
'terraform init'
terraform {
  backend "s3" {
    key = "example/terraform.tfstate"
  }
}
```

To put all your partial configurations together, run `terraform init` with the `-backend-config` argument: