Atividade 3 – Manipulando objetos na nuvem(AWS S3) usando SDK.

Disciplina : Desenvolvimento de Aplicações Distribuídas
Professor : Dr Flávio Sousa
Alunos : Thyago Freitas da Silva - 392035
José Marcílio De Sousa - 385199

Fortaleza - CE
2020

Descrição da Atividade

Para essa atividade, utilizaremos a instância do EC2 criada na Atividade 2 (Subindo uma aplicação web para Nuvem (AWS)) e o SDK do AWS S3 para Golang (https://docs.aws.amazon.com/sdk-for-go/api/service/s3/) .

Para iniciar o desenvolvimento, precisaremos importar os seguintes pacotes:

```
import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)
```

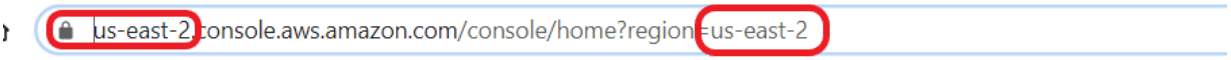
Criamos uma sessão usando o pacote **github.com/aws/aws-sdk-go/aws/session.**
Para este exemplo, especificamos a Região("us-east-2"):

```
const (
    REGION = "us-east-2"
)

var (
    s3session *s3.S3
)

func init() {
    if s3session == nil {
        s3session = s3.New(session.Must(session.NewSession(&aws.Config{
            Region: aws.String(REGION),
        })))
    }
}
```

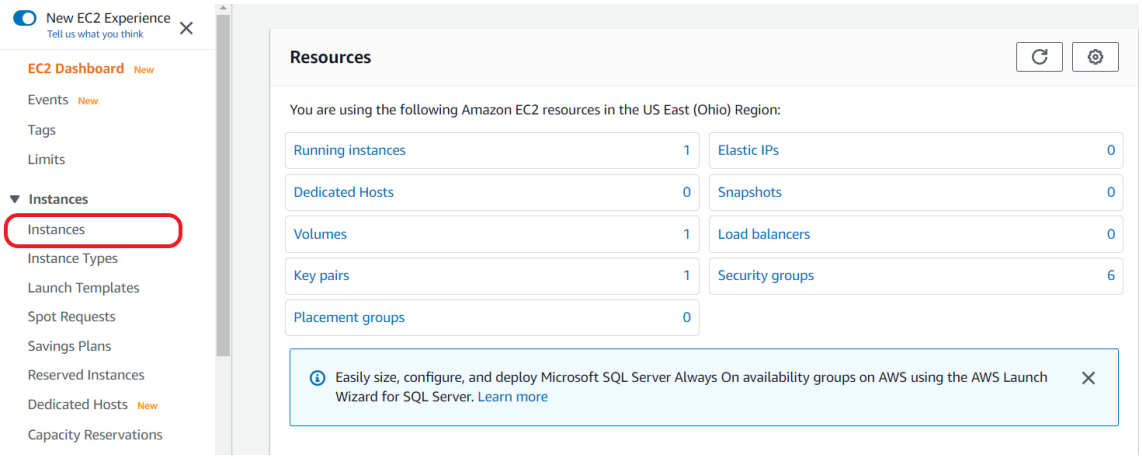
Para saber qual Região usar, basta acessar https://console.aws.amazon.com . A URL mudará para https://{region}.console.aws.amazon.com/console/home?region={region}



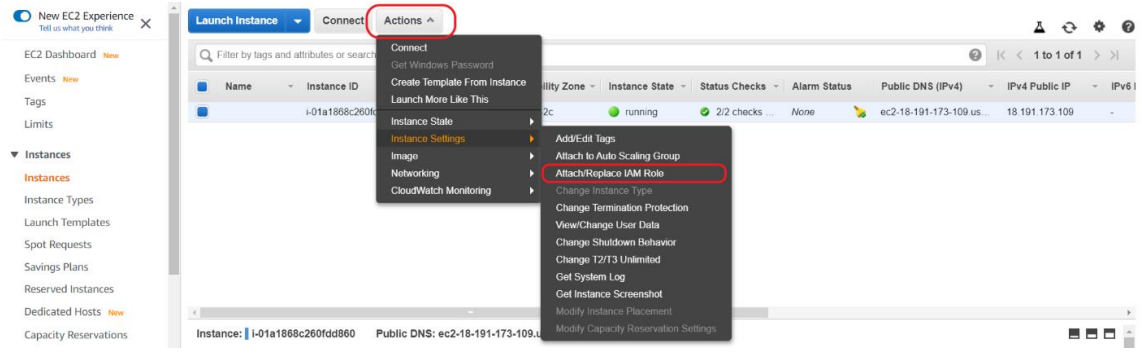
Para finalizar a configuração, precisamos especificar as credenciais que serão usadas nas solicitações ao AWS S3, como por exemplo, a criação de um Bucket.
Existem algumas formas de especificar as credenciais, como definindo um arquivo de configurações ou variáveis de ambiente, mas como estamos executando nossa aplicação numa instância do EC2, podemos fazer isso de uma forma mais simples, criando uma Função(Role) no AWS IAM(Identity and Access Management) e associando esta Função à nossa instância do EC2.

Passo a passo para criação de uma Função (Role) do IAM e sua atribuição à uma instância do EC2

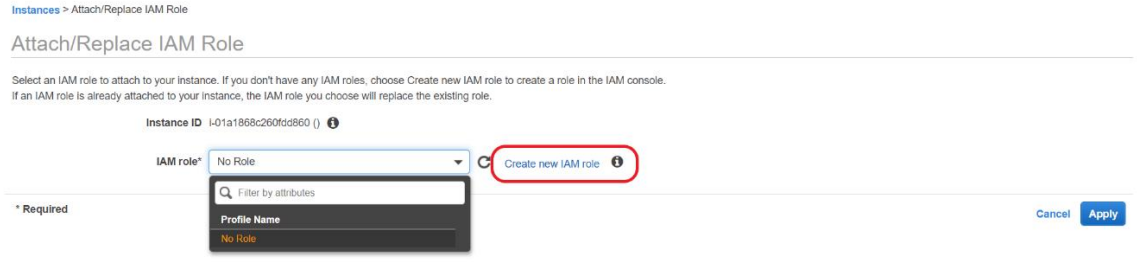
Passo 1 – Na tela inicial do AWS EC2, procuramos por “Instances” para visualizar as instâncias criadas no EC2:



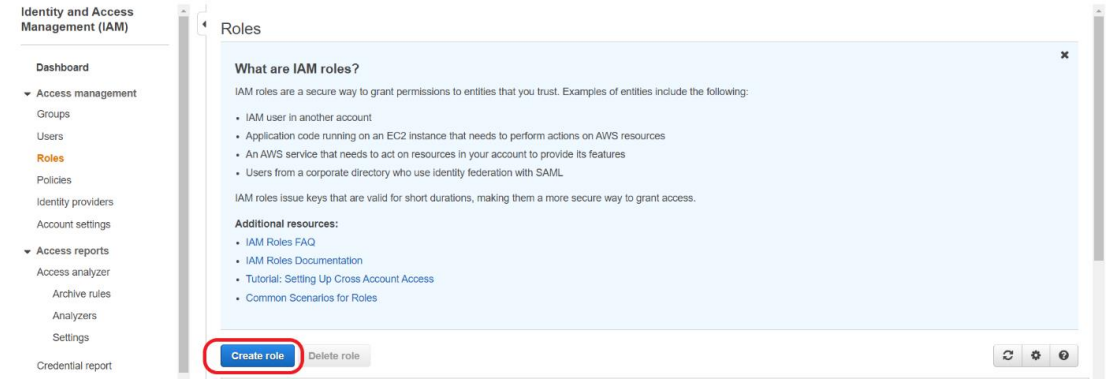
Passo 2 – Selecionamos a instância que desejamos configurar e selecionamos Actions > Instance Settings > Attach/Replace IAM Role:



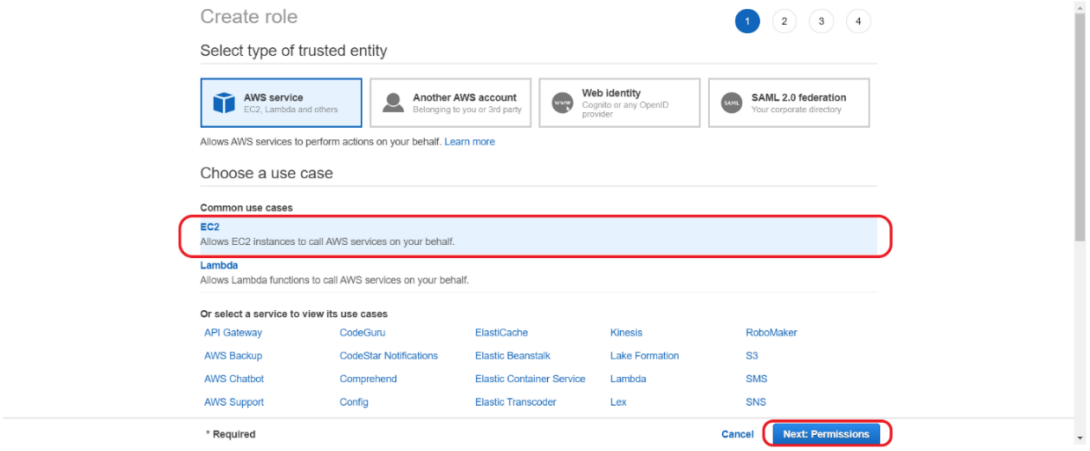
Passo 3 – Na tela seguinte, selecionamos “Create new IAM role”:



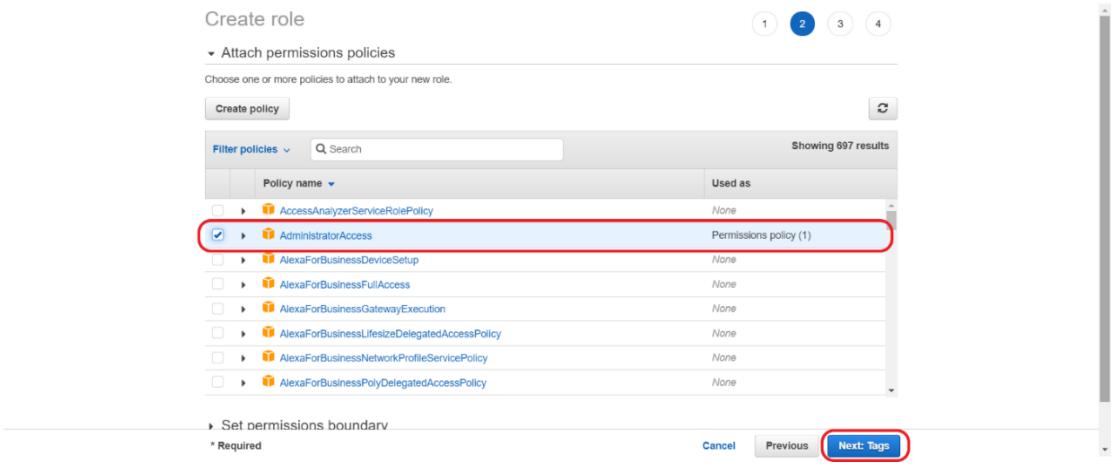
Passo 4 – Clicamos em “Create Role”:



Passo 5 – Escolhemos EC2, para permitir que a instância do EC2 possa chamar os serviços da AWS e clicamos em “Next: Permissions”:

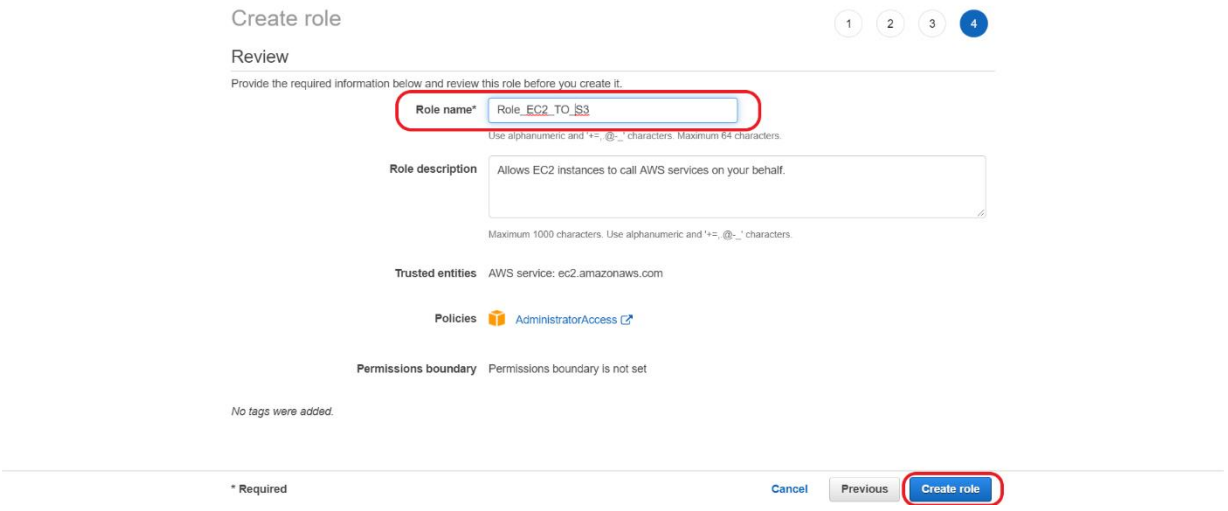


Passo 6 – Para esse exemplo, escolhemos a permissão “AdministratorAccess” e clicamos em “Next: Tags”:

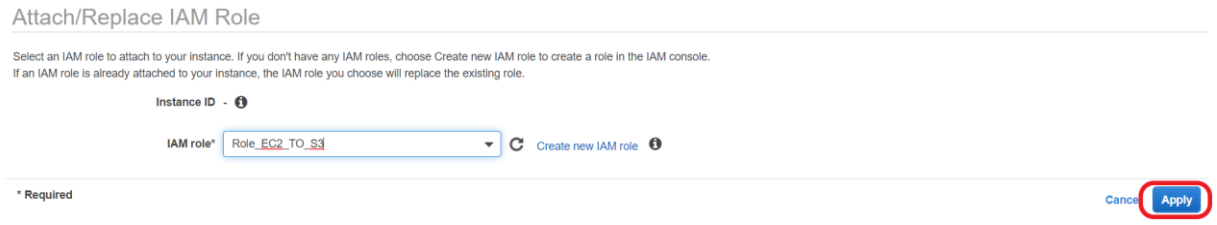


Na tela seguinte, basta clicar em “Next: Review”.

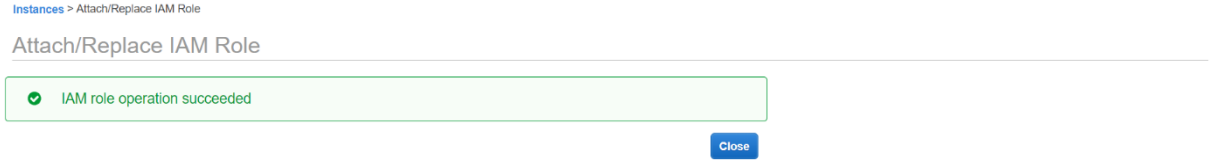
Passo 7 – Definimos o nome da função, para este exemplo “Role_EC2_TO_S3” e clicamos em “Create Role”:



Passo 8 – Ao retornar à tela Attach/Replace IAM Role, escolhemos Role_EC2_TO_S3 e clicamos em “Apply”:



Após isso, podemos ver a mensagem de sucesso. A instância já está configurada para realizar as chamadas ao S3 que utilizaremos no exemplo:



Aplicação Utilizada

A aplicação web criada na atividade 2 consiste em uma pequena API escrita em Golang para o gerenciamento de livros, rodando em um container Docker.
Na atividade 3, usando o SDK do S3 para Go, criamos as seguintes funções:

- Para criação de um Bucket no S3, quando é solicitada a criação de um livro em um Bucket específico:

```
func CreateBucket(bucket_name string) (err error) {
    _, err0 := s3session.CreateBucket(&s3.CreateBucketInput{
        Bucket: aws.String(bucket_name),
        CreateBucketConfiguration: &s3.CreateBucketConfiguration{
            LocationConstraint: aws.String(REGION),
        },
    })

    if err0 != nil {
        if aerr, ok := err0.(awserr.Error); ok {
            switch aerr.Code() {
            case s3.ErrCodeBucketAlreadyExists:
                fmt.Println("Bucket name already exists!")
                err = err0
            case s3.ErrCodeBucketAlreadyOwnedByYou:
                fmt.Println("Bucket name exists and is owned by you!")
            default:
                err = err0
            }
        }
    }

    return err
}
```

- Para listar os objetos que pertencem a um Bucket:

```
func ListObjects(bucketName string) (resp *s3.ListObjectsV2Output, err error) {
    return s3session.ListObjectsV2(&s3.ListObjectsV2Input{
        Bucket: aws.String(bucketName),
    })
}
```

- Para listar os objetos que pertencem a um Bucket, dentro de uma “pasta” específica:

```
func ListObjectsInFolder(bucketName string, folderName string) (resp *s3.ListObjectsV2Output, err error) {
    return s3session.ListObjectsV2(&s3.ListObjectsV2Input{
        Bucket: aws.String(bucketName),
        Prefix: aws.String(folderName),
    })
}
```

- Para obter um objeto específico dentro de um Bucket:

```
func GetObject(fileName string, bucketName string) (obj []byte, err error){
    resp, err := s3session.GetObject(&s3.GetObjectInput{
        Bucket: aws.String(bucketName),
        Key: aws.String(fileName),
    })

    obj, err1 := ioutil.ReadAll(resp.Body)

    if err1 != nil {
        panic(err1)
    }

    return obj, err
}
```

- Para excluir um objeto específico de um Bucket:

```
func DeleteObject(bucketName string, fileName string) (resp *s3.DeleteObjectOutput, err error) {
    return s3session.DeleteObject(&s3.DeleteObjectInput{
        Bucket: aws.String(bucketName),
        Key: aws.String(fileName),
    })
}
```

- Para excluir bucket:

```
func DeleteBucket(bucketName string) (resp *s3.DeleteBucketOutput, err error) {
    return s3session.DeleteBucket(&s3.DeleteBucketInput{
        Bucket: aws.String(bucketName),
    })
}
```

Testando a Aplicação

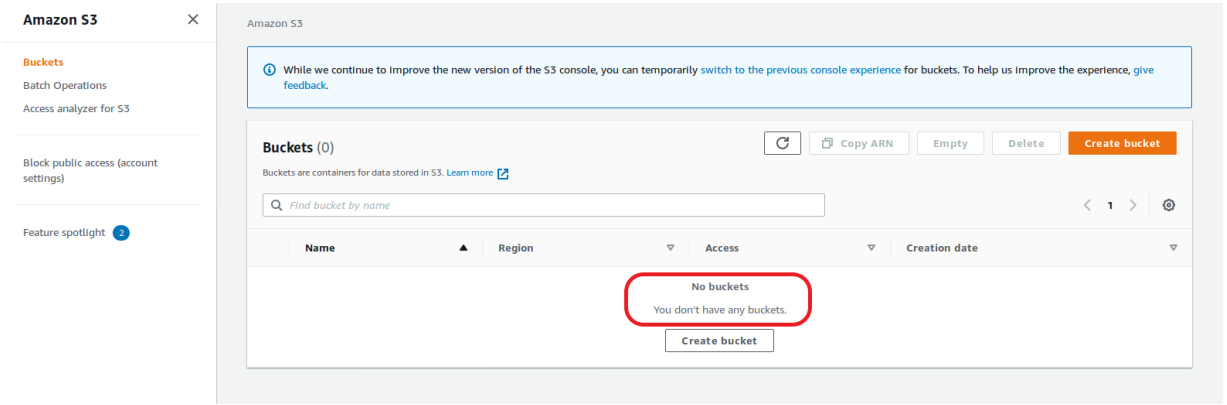
Disponibilizamos as seguintes rotas para teste, de acordo com o IPv4 mostrado no EC2:

1. http://ec2-18-188-98.us-east-2.compute.amazonaws.com/books/{bucket_name}
   GET (Retornar todos os livros que estão no Bucket especificado)
2. http://ec2-18-188-98.us-east-2.compute.amazonaws.com /books/{bucket_name}/{id}
   GET (Retornar o livro com ID especificado e dentro no Bucket especificado)
3. http://ec2-18-188-98.us-east-2.compute.amazonaws.com/books/{bucket_name}/{id}
   DELETE (Deleta o livro com ID especificado e dentro no Bucket especificado)
4. http://ec2-18-188-98.us-east-2.compute.amazonaws.com/books/{bucket_name}
   DELETE (Deleta o Bucket especificado)
5. http://ec2-18-188-98.us-east-2.compute.amazonaws.com/books
   POST (Cadastra livro)

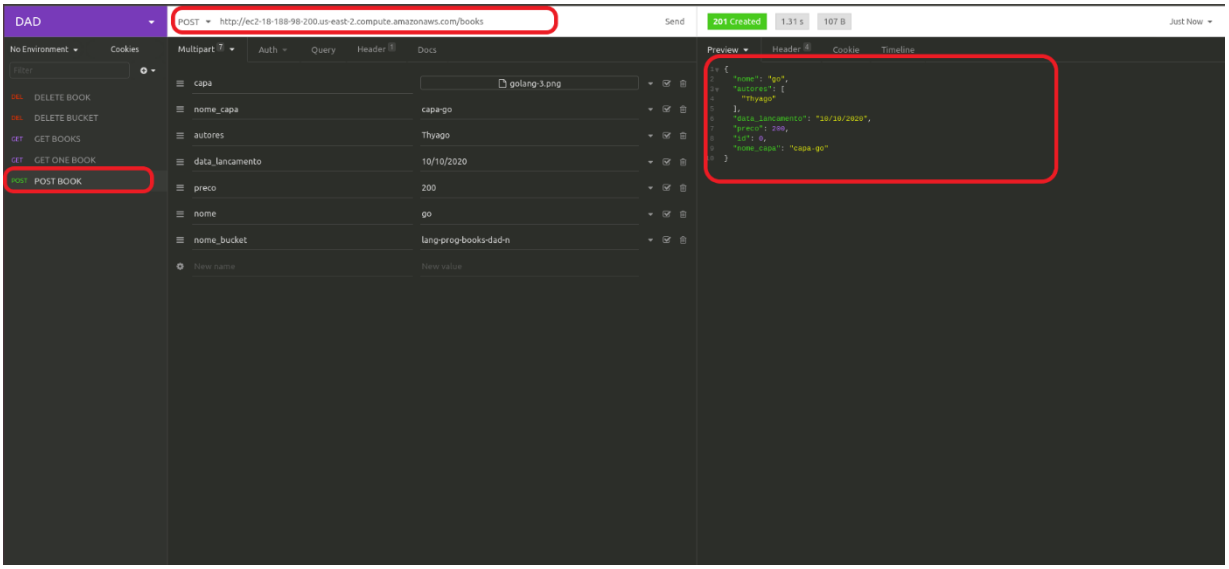
No cadastro, especificamos alguns atributos do livro:

- nome (texto) – Representa o nome do livro e é usado para nomear a “pasta” no Bucket onde ficará o livro e a sua capa.
- autores (texto, separando os autores por vírgula) – Representa um ou mais autores do livro, com seus nomes separados por vírgula.
- data_lancamento (texto) – Representa a data de lançamento do livro.
- preco (texto) – Representa o preço do livro.
- nome_bucket (texto) – O nome do Bucket em que o livro deve ser salvo. Se o Bucket não existir, será criado. Se existir e pertencer ao usuário será usado o existente, se existir e pertencer a outro usuário, será retornada uma mensagem de erro.
- capa (arquivo) – A imagem que representa a capa do livro, será salvo numa “pasta”, junto com o JSON representando os atributos do livro, dentro do Bucket.
- nome_capa (texto) – O nome que será atribuído ao arquivo da capa, dentro do Bucket.

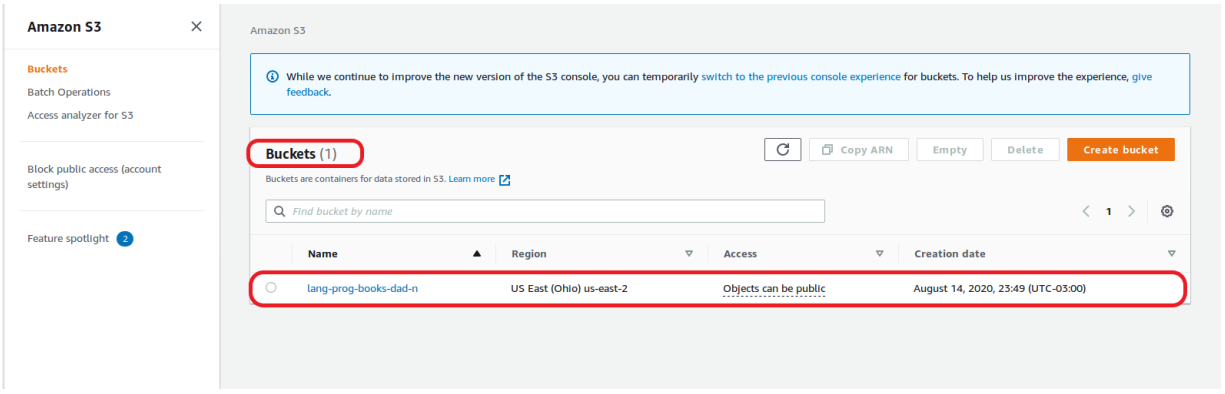
Esses atributos, exceto o nome do Bucket, serão salvos em um arquivo JSON no S3, dentro do Bucket especificado.
Para testar a aplicação utilizaremos o Insomnia. Primeiro, vemos que no S3 ainda não existe nenhum Bucket criado:

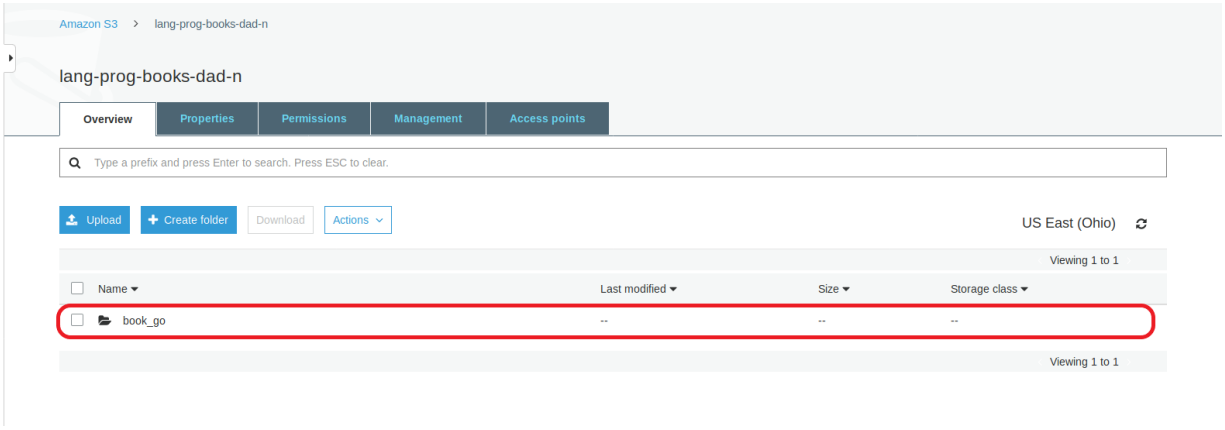


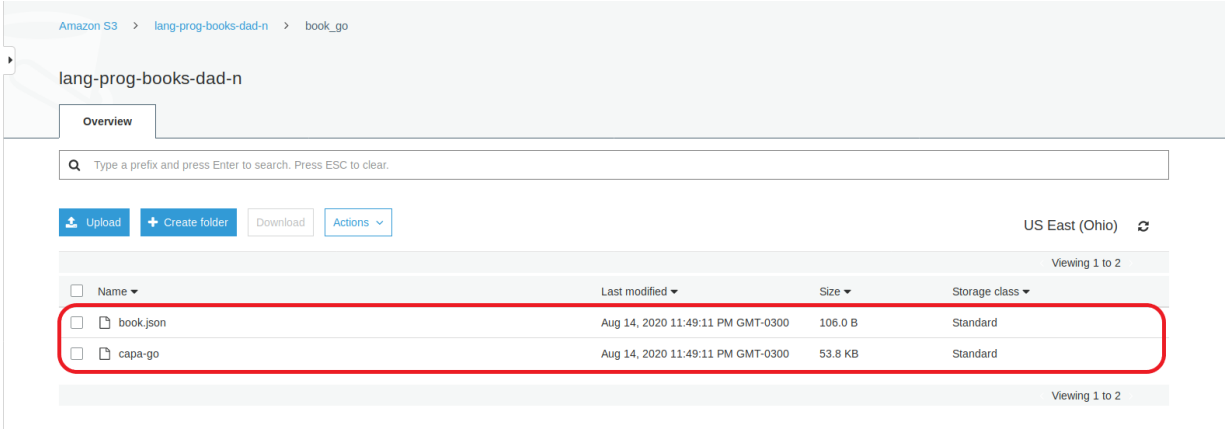
1. Cadastrando um livro e criando o Bucket “lang-prog-books-dad-n”:



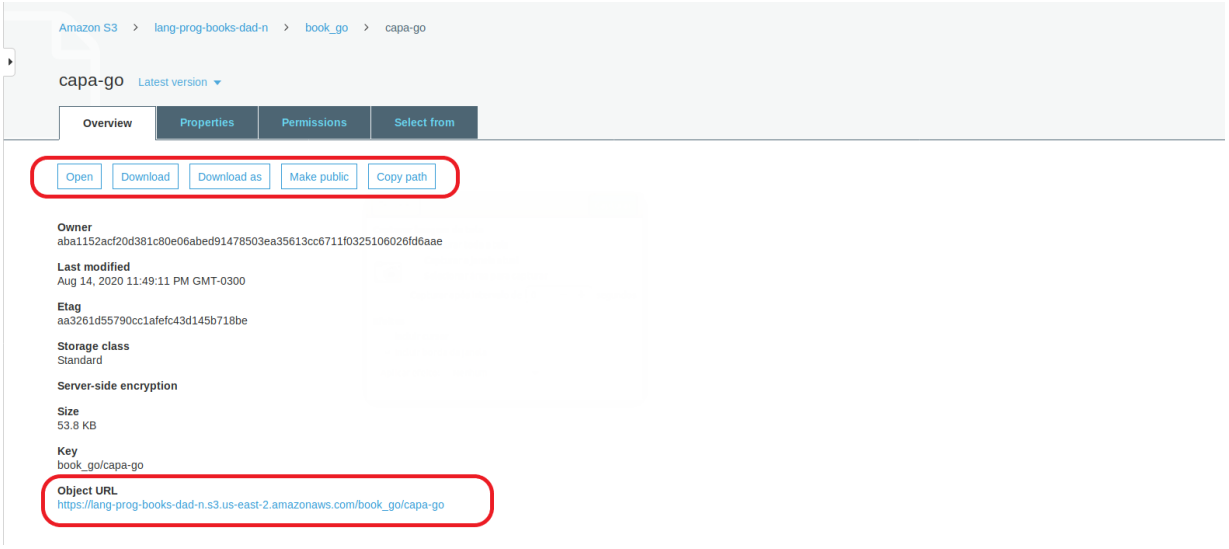
Após o retorno, vemos no S3 que o Bucket e os arquivos foram criados:



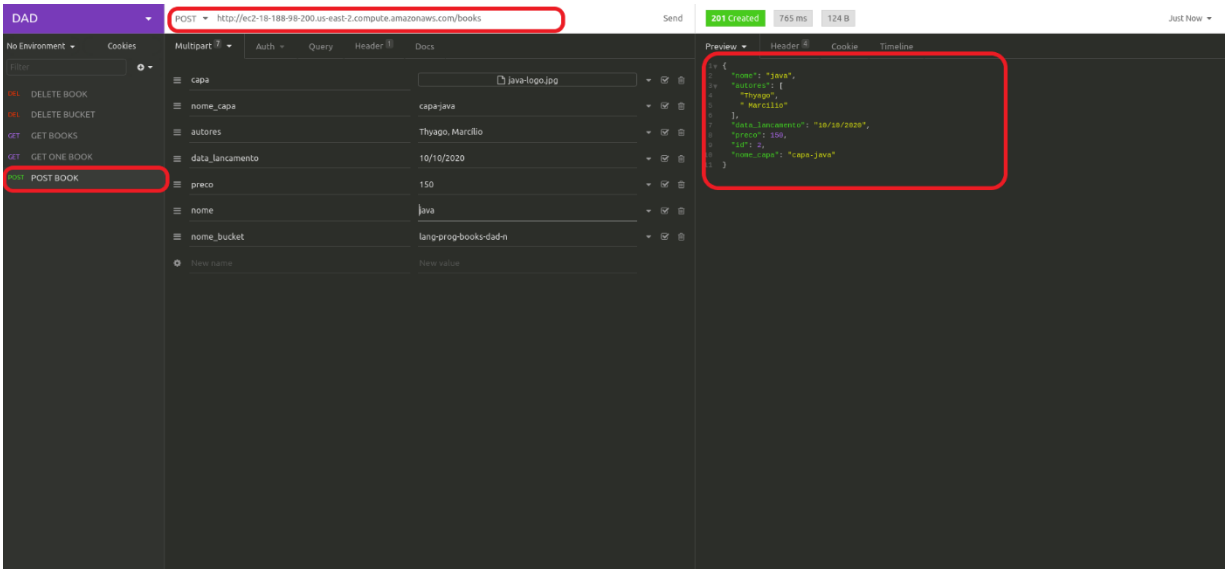


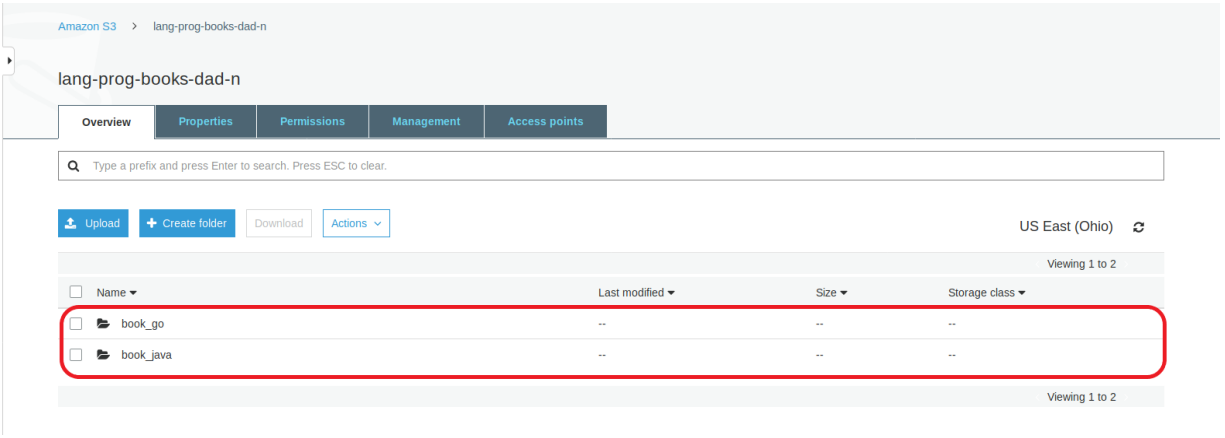


Também podemos ver os detalhes de um objeto específico, bem como fazer o download pela própria plataforma do S3:



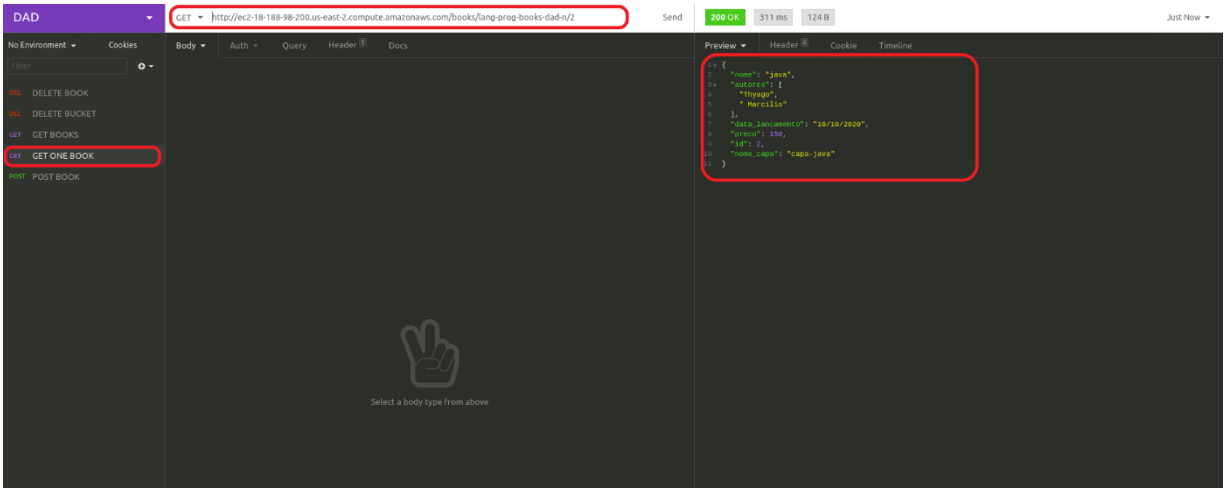
2. Cadastrando outro livro:



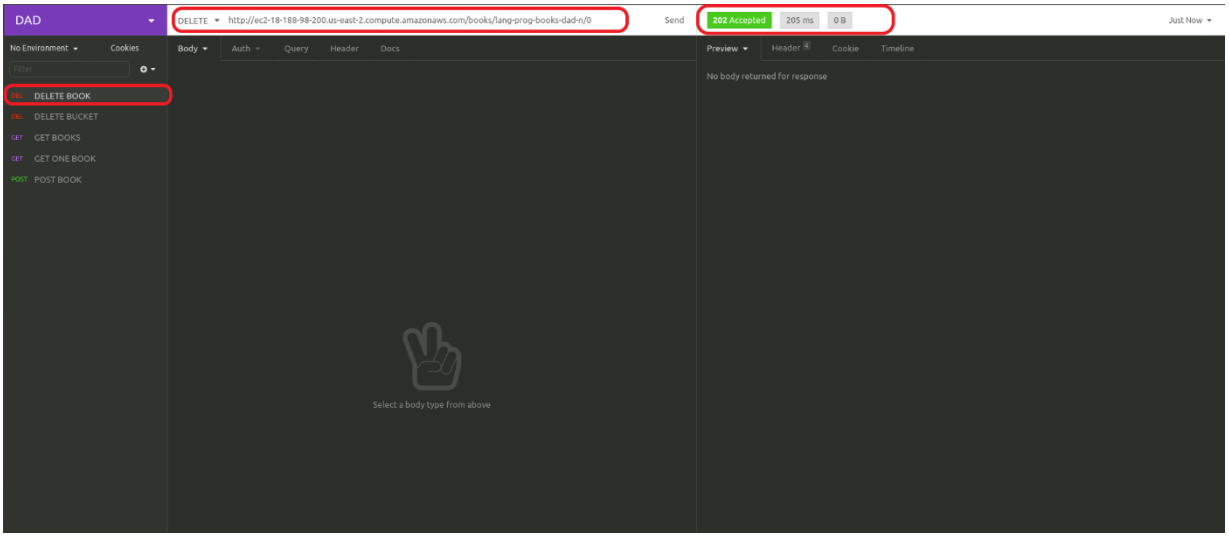


3. Buscando todos os livros cadastrados em um Bucket.

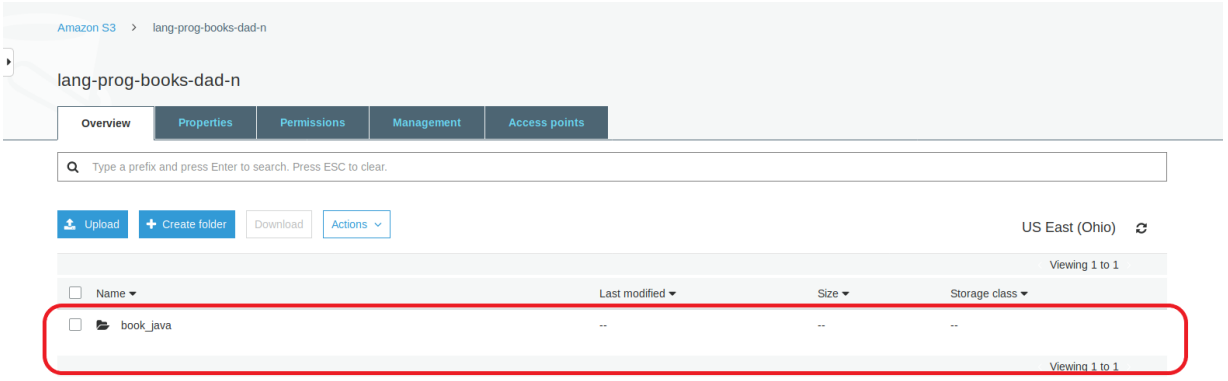
4. Buscando o livro com ID 2:



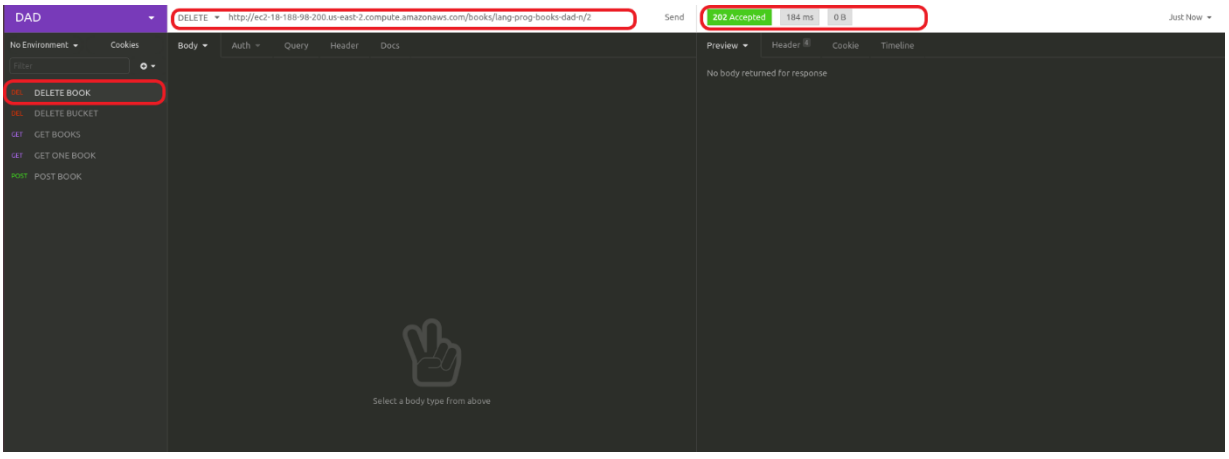
5. Deletando o livro com ID 0:



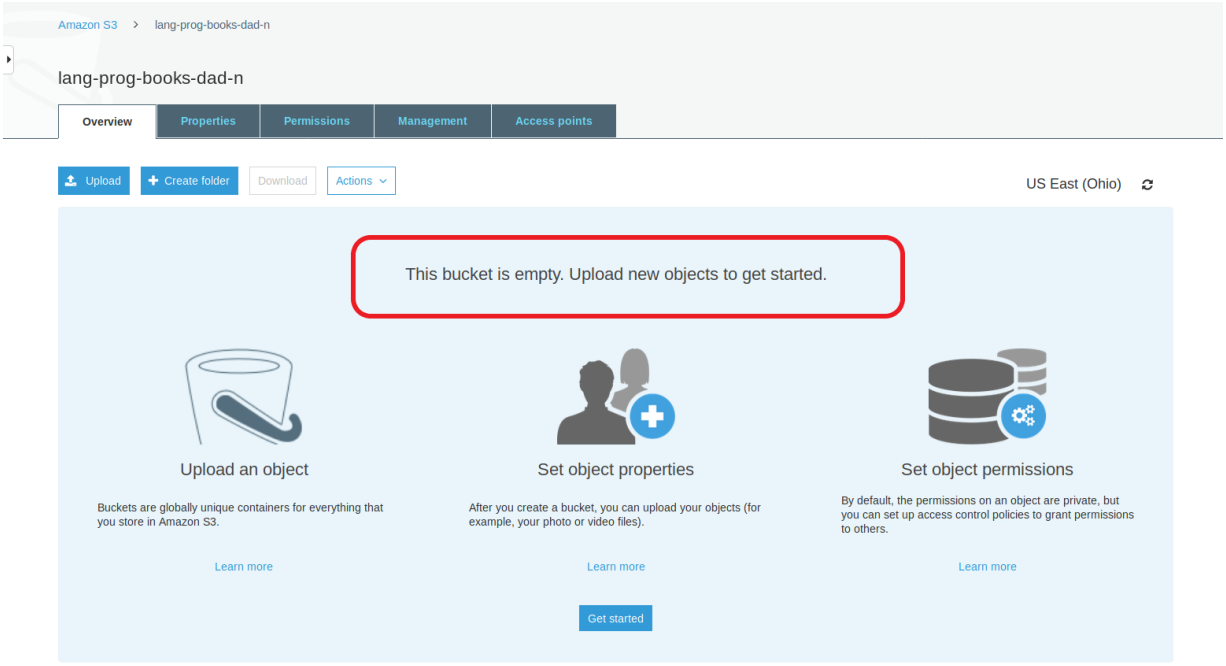
Atualizando o S3, vemos que agora só existe 1 livro:



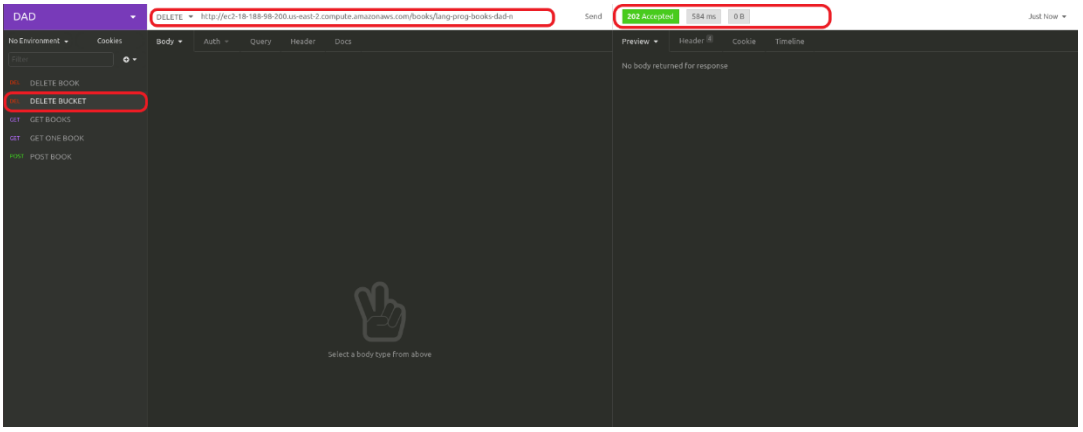
6. Deletando o livro com ID 2:



Atualizando o S3, vemos que o Bucket está vazio e, portanto, pode ser excluído:



7. Deletando o Bucket:



Atualizando o S3, verificamos que o Bucket foi excluído: