CStor Vector go SDK 使用说明文档

下载与安装

资源下载

可在天翼云官网下载，下载地址：天翼云盘（访问码：wr9y）

环境依赖

go >= 1.24

安装

从零开始创建并运行一个cstor-vector-sdk-go项目

1. 从官网下载sdk压缩包，并解压，进入解压后的路径

```
PS D:\code\go\test\cstor-vector-sdk-golang> ls


    目录: D:\code\go\test\cstor-vector-sdk-golang


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-r---         2026/3/11     17:22                smithy-go
d-----         2026/3/11     17:23                vector-sdk-go
-a----         2026/3/11     19:16           4244 README.md


PS D:\code\go\test\cstor-vector-sdk-golang>
```

2. sdk路径即为解压后的目录路径, 记为GO_SDK_PATH

```
PS D:\code\go\test\cstor-vector-sdk-golang> pwd

Path
----
D:\code\go\test\cstor-vector-sdk-golang
```

3. 创建目录, 从零开始生成一个go项目

3.1 初始化项目

创建一个空目录, 并初始化cstor-vector-sdk-go-example项目

```
# 生成 go.mod 文件
go mod init cstor-vector-sdk-go-example
```

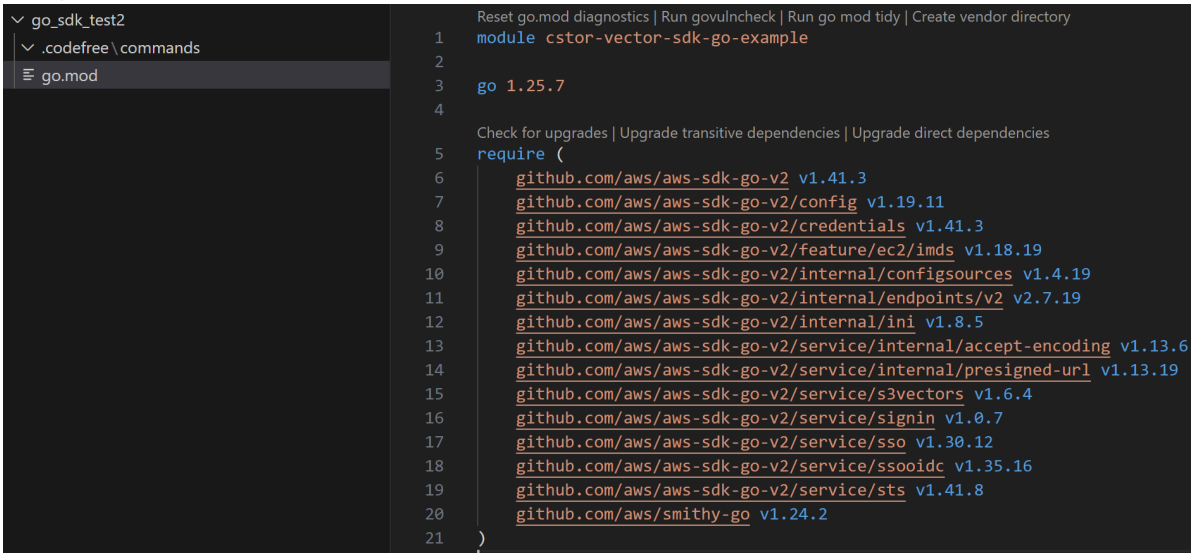
```
PS D:\code\go\test\go_sdk_test2> go mod init cstor-vector-sdk-go-example
go: creating new go.mod: module cstor-vector-sdk-go-example
```

3.2 配置依赖

在项目目录下执行以下命令：

```
# 配置本地依赖
go mod edit -require="github.com/aws/aws-sdk-go-v2@v1.41.3"
go mod edit -require="github.com/aws/aws-sdk-go-v2/config@v1.19.11"
go mod edit -require="github.com/aws/aws-sdk-go-v2/credentials@v1.41.3"
go mod edit -require="github.com/aws/aws-sdk-go-v2/service/s3vectors@v1.6.4"
go mod edit -require="github.com/aws/aws-sdk-go-v2/feature/ec2/imds@v1.18.19"
go mod edit -require="github.com/aws/aws-sdk-go-
v2/internal/configsources@v1.4.19"
go mod edit -require="github.com/aws/aws-sdk-go-
v2/internal/endpoints/v2@v2.7.19"
go mod edit -require="github.com/aws/aws-sdk-go-v2/internal/ini@v1.8.5"
go mod edit -require="github.com/aws/aws-sdk-go-v2/service/internal/accept-
encoding@v1.13.6"
go mod edit -require="github.com/aws/aws-sdk-go-v2/service/internal/presigned-
url@v1.13.19"
go mod edit -require="github.com/aws/aws-sdk-go-v2/service/signin@v1.0.7"
go mod edit -require="github.com/aws/aws-sdk-go-v2/service/sso@v1.30.12"
go mod edit -require="github.com/aws/aws-sdk-go-v2/service/ssooidc@v1.35.16"
go mod edit -require="github.com/aws/aws-sdk-go-v2/service/sts@v1.41.8"
go mod edit -require="github.com/aws/smithy-go@v1.24.2"
```

此时go.mod文件内容如下:



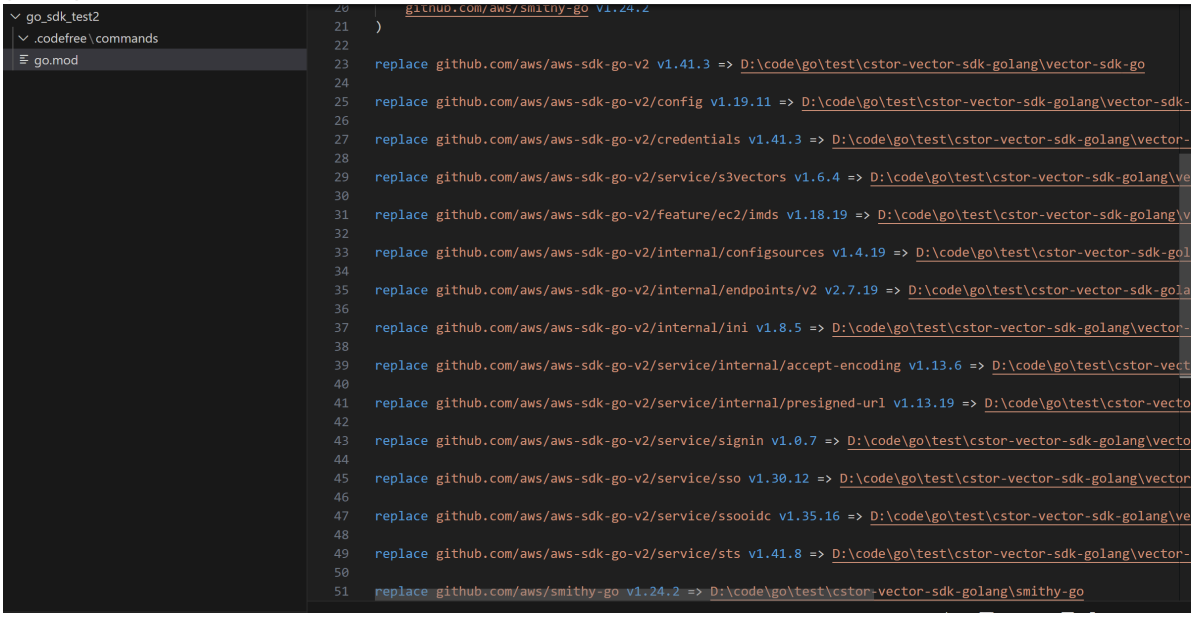
3.3 替换依赖为本地依赖

在第一和第二步中, 我们已经下载了cstor-vector-sdk-go的源码, 现在我们需要将go.mod文件中的依赖替换为本地依赖,
在项目目录下执行以下命令：({GO_SDK_PATH}要替换为第二步中获取到的GO_SDK_PATH)

```
go mod edit -replace="github.com/aws/aws-sdk-go-v2@v1.41.3="
{GO_SDK_PATH}\vector-sdk-go"
go mod edit -replace="github.com/aws/aws-sdk-go-v2/config@v1.19.11="
{GO_SDK_PATH}\vector-sdk-go\config"
go mod edit -replace="github.com/aws/aws-sdk-go-v2/credentials@v1.41.3="
{GO_SDK_PATH}\vector-sdk-go\credentials"
```

```
go mod edit -replace="github.com/aws/aws-sdk-go-v2/service/s3vectors@v1.6.4="
{GO_SDK_PATH}\vector-sdk-go\service\s3vectors"
go mod edit -replace="github.com/aws/aws-sdk-go-v2/feature/ec2/imds@v1.18.19="
{GO_SDK_PATH}\vector-sdk-go\feature\ec2\imds"
go mod edit -replace="github.com/aws/aws-sdk-go-
v2/internal/configsources@v1.4.19="{GO_SDK_PATH}\vector-sdk-
go\internal\configsources"
go mod edit -replace="github.com/aws/aws-sdk-go-
v2/internal/endpoints/v2@v2.7.19="{GO_SDK_PATH}\vector-sdk-
go\internal\endpoints\v2"
go mod edit -replace="github.com/aws/aws-sdk-go-v2/internal/ini@v1.8.5="
{GO_SDK_PATH}\vector-sdk-go\internal\ini"
go mod edit -replace="github.com/aws/aws-sdk-go-v2/service/internal/accept-
encoding@v1.13.6="{GO_SDK_PATH}\vector-sdk-go\service\internal\accept-encoding"
go mod edit -replace="github.com/aws/aws-sdk-go-v2/service/internal/presigned-
url@v1.13.19="{GO_SDK_PATH}\vector-sdk-go\service\internal\presigned-url"
go mod edit -replace="github.com/aws/aws-sdk-go-v2/service/signin@v1.0.7="
{GO_SDK_PATH}\vector-sdk-go\service\signin"
go mod edit -replace="github.com/aws/aws-sdk-go-v2/service/sso@v1.30.12="
{GO_SDK_PATH}\vector-sdk-go\service\sso"
go mod edit -replace="github.com/aws/aws-sdk-go-v2/service/ssooidc@v1.35.16="
{GO_SDK_PATH}\vector-sdk-go\service\ssooidc"
go mod edit -replace="github.com/aws/aws-sdk-go-v2/service/sts@v1.41.8="
{GO_SDK_PATH}\vector-sdk-go\service\sts"
go mod edit -replace="github.com/aws/smithy-go@v1.24.2="{GO_SDK_PATH}\smithy-
go"
```

执行完毕以后, go.mod的文件内容如下:(示例的GO_SDK_PATH为"D:\code\go\test\cstor-vector-sdk-golang")



4. 测试安装

为了确认你已成功安装cstor-vector-sdk-go, 我们可以编写一个简单的程序来测试一下, 在项目目录下创建main.go, 并写入以下内容:

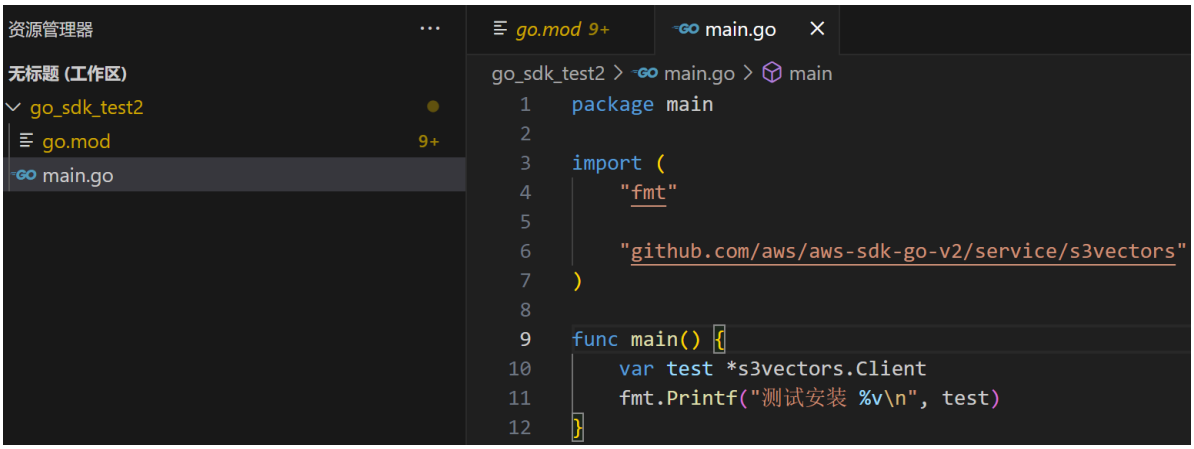
```go
package main

import (
    "fmt"

    "github.com/aws/aws-sdk-go-v2/service/s3vectors"
)

func main() {
    var test *s3vectors.Client
    fmt.Printf("测试安装 %v\n", test)
}
```

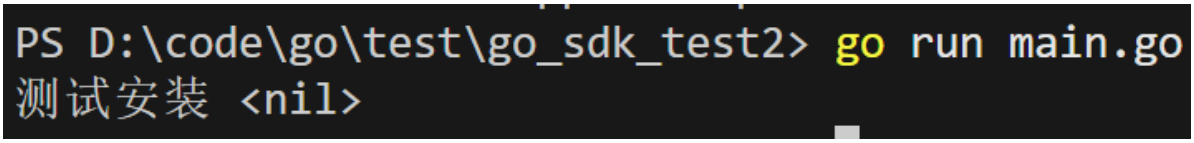
如下所示:



在命令行运行main.go, 如无报错, 则说明安装成功:

```
go run main.go
```

如下所示:



通用：Client 初始化

在上述示例中, 我们讲述了如何安装, 下面我们在项目目录下修改main.go, 并初始化client

```go
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/credentials"
    "github.com/aws/aws-sdk-go-v2/service/s3vectors"
)

const (
    Region    = "us-east-1"
    AccessKey = "your ak"
    SecretKey = "your sk"
    Url       = "your endpoint"
```

```
)

func GetS3VectorClient(ctx context.Context) (*s3vectors.Client, error) {
    // 初始化config
    cfg, err := config.LoadDefaultConfig(ctx, config.WithBaseEndpoint(Url),
config.WithRegion(Region),

config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(AccessKe
y, SecretKey, "")))
    if err != nil {
        fmt.Printf("加载 S3 Vector 配置失败: %s\n", err)
        return nil, fmt.Errorf("加载 S3 Vector 配置失败: %w", err)
    }

    // 创建s3vectors client
    return s3vectors.NewFromConfig(cfg), nil
}
```

公共参数说明

- `AccessKey` / `SecretKey`：认证信息
- `Region`：区域
- `Url`：S3 Vectors 访问地址

---

1. CreateVectorBucket

功能

创建向量桶。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要创建的向量桶名称	string	是

输出参数

- `VectorBucketArn`：向量桶 ARN

Go 示例

```
// create vector bucket 展示完整的示例, 后续示例不再重复展示client初始化代码
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/credentials"
    "github.com/aws/aws-sdk-go-v2/service/s3vectors"
)
```

```
const (
    Region    = "us-east-1"
    AccessKey = "yourak"
    SecretKey = "yoursk"
    Url       = "your endpoint"
)

func GetS3VectorClient(ctx context.Context) (*s3vectors.Client, error) {
    // 初始化config
    cfg, err := config.LoadDefaultConfig(ctx, config.WithBaseEndpoint(Url),
config.WithRegion(Region),

config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(AccessKe
y, SecretKey, "")))
    if err != nil {
        fmt.Printf("加载 S3 Vector 配置失败: %s\n", err)
        return nil, fmt.Errorf("加载 S3 Vector 配置失败: %w", err)
    }

    // 创建s3vectors client
    return s3vectors.NewFromConfig(cfg), nil
}

func main() {
    ctx := context.Background()
    client, err := GetS3VectorClient(ctx)
    if err != nil {
        fmt.Printf("获取向量vector失败: %s\n", err)
        return
    }
    bucketName := "my-vector-bucket"
    bucket, err := client.CreateVectorBucket(ctx,
&s3vectors.CreateVectorBucketInput{
        VectorBucketName: &bucketName,
    })
    if err != nil {
        fmt.Printf("操作失败: %s\n", err)
        return
    }
    fmt.Printf("ARN: %s\n", *bucket.VectorBucketArn)
}
```

---

2. DeleteVectorBucket

功能

删除向量桶。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个

输出参数

- 空输出结构体（无字段）

Go 示例

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}
bucketName := "my-vector-bucket"
_, err = client.DeleteVectorBucket(ctx, &s3vectors.DeleteVectorBucketInput{
    VectorBucketName: &bucketName,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}
fmt.Printf("删除向量桶 succ\n")
```

---

3. GetVectorBucket

功能

查询向量桶详情。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个

输出参数

- `VectorBucket.VectorBucketName`：向量桶名称
- `VectorBucket.VectorBucketArn`：向量桶 ARN
- `VectorBucket.CreationTime`：创建时间

Go 示例

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}
bucketName := "my-vector-bucket"
rsp, err := client.GetVectorBucket(ctx, &s3vectors.GetVectorBucketInput{
    VectorBucketName: &bucketName,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}
fmt.Printf("bucket: %s bucketArn: %s createTime: %v\n",
*rsp.VectorBucket.VectorBucketName,
    *rsp.VectorBucket.VectorBucketArn, *rsp.VectorBucket.CreationTime)
```

---

4. ListVectorBuckets

功能

列出向量桶列表。

输入参数

参数名称	参数含义	参数类型	是否必须
MaxResults	最大返回条数	int	否
Prefix	名称前缀过滤	string	否
NextToken	分页token	string	否

输出参数

- `NextToken`：分页 token
- `VectorBuckets[]`：向量桶列表
  - `VectorBucketName`
  - `VectorBucketArn`
  - `CreationTime`

Go 示例

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}
maxResults := int32(5)
prefix := "my"
nextToken := "" // 如有

rsp, err := client.ListVectorBuckets(ctx, &s3vectors.ListVectorBucketsInput{
    MaxResults: &maxResults,
    Prefix:     &prefix,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}

if rsp.NextToken != nil {
    fmt.Printf("nextToken: %s\n", *rsp.NextToken)
}

for _, bucket := range rsp.VectorBuckets {
    fmt.Printf("bucket: %s bucketArn: %s createTime: %v\n",
*bucket.VectorBucketName,
        *bucket.VectorBucketArn, *bucket.CreationTime)
}
```

---

5. PutVectorBucketPolicy

功能

设置向量桶策略。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个
Policy	策略 JSON 字符串	string	是

policy参数说明:

字段	描述	类型	是否必须
Version	保持与Amazon S3一致，当前支持"2012-10-17"	string	否
Id	桶策略ID，桶策略的唯一标识	string	否
Statement	桶策略描述，定义完整的权限控制。每条桶策略的Statement可由多条描述组成，每条描述是一个dict，每条描述可包含以下字段： Sid Effect Principal Action ResSource Condition	list	是
Sid	本条桶策略描述的ID	string	否
Effect	桶策略的效果，即指定本条桶策略描述的权限是接受请求还是拒绝请求。 接受请求：配置为“Allow”， 拒绝请求：配置为“Deny”	string	是
Principal	被授权人，即指定本条桶策略描述所作用的用户，支持通配符“*”，表示所有用户。当对某个user进行授权时，Principal格式为: "AWS":"arn:cstor:iam:::user/userId"	dict	否
Action	操作，即指定本条桶策略描述所作用的ZOS操作。以列表形式表示，可配置多条操作，以逗号间隔。支持通配符 “*“，表示该资源能进行的所有操作。Action格式示例为: "zosvectors:GetVectorBucket"，"zosvectors:ListVectorBuckets"	list	否
Resource	指定策略作用的一组资源, 支持用通配 符"*"表示所有资源, 如需要指定具体资源, 则需要在Resource中指定资源的ARN, "arn:cstor:vectors:{region-id}:{acount-id}:bucket/{bucket-name}"	string	否

输出参数

- 空输出结构体（无字段）

Go 示例

```go
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}

bucketName := "my-vector-bucket"
policy := `{
    "Version": "2012-10-17",
    "Id": "id1",
    "Statement": [
        {
            "Sid": "sid1",
            "Effect": "Allow",
            "Action": "zosvectors:*",
            "Resource": "arn:cstor:vectors:region:myak:bucket/my-vector-bucket",
            "Principal": "*"
        }
```

```
    ]
}`
rsp, err := client.PutVectorBucketPolicy(ctx,
&s3vectors.PutVectorBucketPolicyInput{
    VectorBucketName: &bucketName,
    Policy:           &policy,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}
fmt.Printf("操作成功: %v\n", rsp)
```

---

6. GetVectorBucketPolicy

功能

获取向量桶策略。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个

输出参数

- `Policy`：策略 JSON 字符串

Go 示例

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}

bucketName := "my-vector-bucket"

rsp, err := client.GetVectorBucketPolicy(ctx,
&s3vectors.GetVectorBucketPolicyInput{
    VectorBucketName: &bucketName,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
```

```
}
fmt.Printf("policy: %v\n", *rsp.Policy)
```

---

7. DeleteVectorBucketPolicy

功能

删除向量桶策略。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个

输出参数

- 空输出结构体（无字段）

Go 示例

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}

bucketName := "my-vector-bucket"

_, err = client.DeleteVectorBucketPolicy(ctx,
&s3vectors.DeleteVectorBucketPolicyInput{
    VectorBucketName: &bucketName,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}
fmt.Printf("操作成功\n")
```

---

8. CreateIndex

功能

在向量桶中创建索引。

输入参数

- `VectorBucketName` 或 `VectorBucketArn`
- `IndexName`
- `Dimension`
- `DataType`（默认 `float32`）
- `DistanceMetric`（`cosine` 或 `euclidean`）
- `MetadataConfiguration.NonFilterableMetadataKeys`（可选）

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个
IndexName	索引名称	string	是
Dimension	向量维度	int	是
DataType	向量数据类型, 默认float32	string	是
DistanceMetric	距离度量, 支持cosine和euclidean	string	是
MetadataConfiguration	非过滤元数据键列表	dict	否

输出参数

- `IndexArn`：索引 ARN

Go 示例

```go
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}

bucketName := "my-vector-bucket"
indexName := "my-index"
Dimension := int32(4)

rsp, err := client.CreateIndex(ctx, &s3vectors.CreateIndexInput{
    VectorBucketName: &bucketName,
    IndexName:        &indexName,
```

```
    Dimension:        &Dimension,
    DataType:         "float32",
    DistanceMetric:   "euclidean",
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}
fmt.Printf("index arn: %s\n", *rsp.IndexArn)
```

---

9. GetIndex

功能

查询索引详情。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个

输出参数

- `Index.VectorBucketName`
- `Index.IndexName`
- `Index.IndexArn`
- `Index.CreationTime`
- `Index.DataType`
- `Index.Dimension`
- `Index.DistanceMetric`
- `Index.MetadataConfiguration.NonFilterableMetadataKeys[]`

Go 示例

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}

bucketName := "my-vector-bucket"
indexName := "my-index"

rsp, err := client.GetIndex(ctx, &s3vectors.GetIndexInput{
```

```
    VectorBucketName: &bucketName,
    IndexName:        &indexName,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}
fmt.Printf("index arn %s  create time %s, dimension %d, data type %s, distance
metric %s\n", *rsp.Index.IndexArn, *rsp.Index.CreationTime,
    *rsp.Index.Dimension, rsp.Index.DataType, rsp.Index.DistanceMetric)
```

---

10. ListIndexes

功能

列出向量桶内的索引。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个
MaxResults	最大返回条数	int	否
Prefix	名称前缀过滤	string	否
NextToken	分页token	string	否

输出参数

- `NextToken`：分页 token
- `Indexes[]`：索引列表
  - `VectorBucketName`
  - `IndexName`
  - `IndexArn`
  - `CreationTime`

Go 示例

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}
```

```
bucketName := "my-vector-bucket"
maxResults := int32(3)
prefix := "my"
// nextToken := "" // 如有

rsp, err := client.ListIndexes(ctx, &s3vectors.ListIndexesInput{
    VectorBucketName: &bucketName,
    MaxResults:       &maxResults,
    Prefix:           &prefix,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}
if rsp.NextToken != nil {
    fmt.Printf("NextToken: %s\n", *rsp.NextToken)
}

for _, index := range rsp.Indexes {
    fmt.Printf("index: %s\n", *index.IndexName)
    fmt.Printf("creationTime: %s\n", *index.CreationTime)
    fmt.Printf("VectorBucketName: %s\n", *index.VectorBucketName)
}
```

---

11. DeleteIndex

功能

删除索引。

输入参数

- `VectorBucketName`
- `IndexName` 或 `IndexArn`

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个
IndexName	索引名称	string	IndexName或IndexArn必须指定一个
IndexArn	索引ARN	string	IndexName或IndexArn必须指定一个

输出参数

- 空输出结构体（无字段）

Go 示例

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}

bucketName := "my-vector-bucket"
indexName := "my-index"

_, err = client.DeleteIndex(ctx, &s3vectors.DeleteIndexInput{
    VectorBucketName: &bucketName,
    IndexName:        &indexName,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}
fmt.Printf("操作成功\n", )
```

---

12. PutVectors

功能

插入或更新向量。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个
IndexName	索引名称	string	IndexName或IndexArn必须指定一个
IndexArn	索引ARN	string	IndexName或IndexArn必须指定一个
Vectors[]	向量列表	list	是

输出参数

- 空输出结构体（无字段）

Go 示例

```
# 创建vector新引入了types包，这里展示一下完整代码.
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/credentials"
    "github.com/aws/aws-sdk-go-v2/service/s3vectors"
    "github.com/aws/aws-sdk-go-v2/service/s3vectors/types"
)

const (
    Region    = "us-east-1"
    AccessKey = "your ak"
    SecretKey = "your sk"
    Url       = "your endpoint"
)

func GetS3VectorClient(ctx context.Context) (*s3vectors.Client, error) {
    // 初始化config
    cfg, err := config.LoadDefaultConfig(ctx, config.WithBaseEndpoint(Url),
config.WithRegion(Region),

config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(AccessKe
y, SecretKey, "")))
    if err != nil {
        fmt.Printf("加载 S3 Vector 配置失败: %s\n", err)
        return nil, fmt.Errorf("加载 S3 Vector 配置失败: %w", err)
    }

    // 创建s3vectors client
    return s3vectors.NewFromConfig(cfg), nil
}

func main() {
    ctx := context.Background()
    client, err := GetS3VectorClient(ctx)
    if err != nil {
        fmt.Printf("获取向量vector client失败: %s\n", err)
        return
    }

    bucketName := "my-vector-bucket"
    indexName := "my-index"
    vectorKey := "my-vector"
    vectorKey2 := "my-vector2"

    // 保证之前已经创建的index和vector bucket, index的Dimension为4
    vectors := []types.PutInputVector{
```

```
        {
            Data: &types.VectorDataMemberFloat32{
                Value: []float32{1.0, 2.0, 3.0, 4.0},
            },
            Key: &vectorKey,
        },

        {
            Data: &types.VectorDataMemberFloat32{
                Value: []float32{1.0, 2.0, 3.0, 4.0},
            },
            Key: &vectorKey2,
        },
    }

    _, err = client.PutVectors(ctx, &s3vectors.PutVectorsInput{
        VectorBucketName: &bucketName,
        IndexName:        &indexName,
        Vectors:          vectors,
    })
    if err != nil {
        fmt.Printf("操作失败: %s\n", err)
        return
    }
    fmt.Printf("操作成功\n", )
}
```

---

13. GetVectors

功能

获取指定向量。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个
IndexName	索引名称	string	IndexName或IndexArn必须指定一个
IndexArn	索引ARN	string	IndexName或IndexArn必须指定一个
Keys[]	向量键列表	list	是
ReturnData	是否返回向量数据	bool	否
ReturnMetadata	是否返回元数据	bool	否

输出参数

- `Vectors[]`:
  - `Key`
  - `Data` (当 `ReturnData=true`)
  - `Metadata` (当 `ReturnMetadata=true`)

Go 示例

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}

bucketName := "my-vector-bucket"
indexName := "my-index"
vectorKey := "my-vector"
vectorKey2 := "my-vector2"
keys := []string{vectorKey, vectorKey2}

rsp, err := client.GetVectors(ctx, &s3vectors.GetVectorsInput{
    VectorBucketName: &bucketName,
    IndexName:        &indexName,
    Keys:             keys,
    ReturnData:       true,
    ReturnMetadata:   true,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
```

```
    return
}
for _, vector := range rsp.Vectors {
    fmt.Printf("vector key: %s, metadata: %v\n", *vector.Key, vector.Metadata)
    fmt.Printf("vector data: %v\n", vector.Data)
}
```

---

14. ListVectors

功能

列出索引中的向量列表。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个
IndexName	索引名称	string	IndexName或IndexArn必须指定一个
IndexArn	索引ARN	string	IndexName或IndexArn必须指定一个
MaxResults	最大返回结果数	int32	否
NextToken	分页游标	string	否
SegmentCount	分段数量	int32	否
SegmentIndex	分段索引	int32	否
ReturnData	是否返回向量数据	bool	否
ReturnMetadata	是否返回元数据	bool	否

输出参数

- `NextToken`
- `Vectors[]`:
  - `Key`
  - `Data.Float32[]` (当 `ReturnData=true`)
  - `Metadata` (当 `ReturnMetadata=true`)

Go 示例（含 bucket/index 初始化）

```go
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}

bucketName := "my-vector-bucket"
indexName := "my-index"
maxResults := int32(2)

rsp, err := client.ListVectors(ctx, &s3vectors.ListVectorsInput{
    VectorBucketName: &bucketName,
    IndexName:        &indexName,
    MaxResults:       &maxResults,
    ReturnData:       true,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}

if rsp.NextToken != nil {
    fmt.Printf("next token: %s\n", *rsp.NextToken)
}

for _, v := range rsp.Vectors {
    fmt.Printf("vector key: %s\n", *v.Key)
    fmt.Printf("vector data: %v\n", v.Data)
}
```

---

15. DeleteVectors

功能

删除指定向量。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个
IndexName	索引名称	string	IndexName或IndexArn必须指定一个
IndexArn	索引ARN	string	IndexName或IndexArn必须指定一个
Keys[]	向量键列表	list	是

输出参数

- 空输出结构体（无字段）

Go 示例（含 bucket/index 初始化）

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}

bucketName := "my-vector-bucket"
indexName := "my-index"
vectorKey := []string{"my-vector"}

_, err = client.DeleteVectors(ctx, &s3vectors.DeleteVectorsInput{
    VectorBucketName: &bucketName,
    IndexName:        &indexName,
    Keys:       vectorKey,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}
fmt.Printf("操作成功\n")
```

---

16. QueryVectors

功能

向量相似度搜索。

输入参数

参数名称	参数含义	参数类型	是否必须
VectorBucketName	需要操作的向量桶名称	string	vectorBucketName和vectorBucketArn只能指定一个
VectorBucketArn	需要操作的向量桶, 以arn格式指定(arn在创建桶时返回)	string	vectorBucketName和vectorBucketArn只能指定一个
IndexName	索引名称	string	IndexName或IndexArn必须指定一个
IndexArn	索引ARN	string	IndexName或IndexArn必须指定一个
QueryVector	查询向量	float32[]	是
TopK	返回结果数量	int32	是
Filter	过滤条件	map[string]string	否
ReturnDistance	是否返回距离	bool	否
ReturnMetadata	是否返回元数据	bool	否

输出参数

- `Vectors[]`:
  - `Key`
  - `Distance`（当 `ReturnDistance=true`）
  - `Metadata`（当 `ReturnMetadata=true`）
- `DistanceMetric`（`cosine` 或 `euclidean`）

Go 示例（含 bucket/index 初始化）

```
ctx := context.Background()
client, err := GetS3VectorClient(ctx)
if err != nil {
    fmt.Printf("获取向量vector client失败: %s\n", err)
    return
}

bucketName := "my-vector-bucket"
indexName := "my-index"
TopK := int32(2)
vectorData := &types.VectorDataMemberFloat32{
```

```
    Value: []float32{1.0, 2.0, 3.0, 4.0},
}

rsp, err := client.QueryVectors(ctx, &s3vectors.QueryVectorsInput{
    VectorBucketName: &bucketName,
    IndexName:        &indexName,
    TopK:             &TopK,
    ReturnDistance:   true,
    QueryVector:       vectorData,
})
if err != nil {
    fmt.Printf("操作失败: %s\n", err)
    return
}

for _, v := range rsp.Vectors {
    fmt.Printf("vector key: %s\n", *v.Key)
    fmt.Printf("vector distance: %v\n", *v.Distance)
}
```