Part 1:

Problem Statement:
Write an ‘HTTP Service’ in any programming language, which should expose an endpoint GET "http://IP:PORT/list-bucket-content/". This endpoint should return the content of an S3 bucket path as specified in the request. If no path is specified, top-level content is returned.

Overview:
This service provides a simple API to list the contents of a specific Amazon S3 bucket. It allows querying for both top-level objects and nested objects (directories) using a prefix parameter, mimicking a folder structure.

Key Features:
- RESTful API: Exposes a GET endpoint to list S3 objects.
- Dynamic Path Querying: List files from the root or within a specific "directory" (prefix).
- Error Handling: Gracefully handles missing credentials, access issues, and invalid requests.

Design Decisions:

1. AWS SDK (boto3):
- The service uses boto3 for interacting with S3.
- We leverage s3.list_objects_v2() to list the objects. This supports prefixes (for "directories") and delimiters, simplifying the process of listing "folders" in S3.

2. Flask as the Web Framework:
- Flask is chosen for its simplicity and flexibility in building RESTful APIs.
- Two routes are defined:
  (i) GET /list-bucket-content/: Lists the top-level contents of the S3 bucket.
  (ii) GET /list-bucket-content/<prefix>: Lists objects under a specific prefix.

3. Path Handling:
- If no prefix is provided, the service defaults to listing top-level objects of the S3 Bucket.
- If a path is specified, it lists objects under that specific directory within the S3 bucket.
- The service ensures that if a prefix is provided without a trailing /, it adds one for proper path handling.

4. Error Handling:
- The service raises appropriate error messages for common issues such as Missing AWS Credentials, Access Denied (Lack of proper S3 permissions) and Directory Not Found.
- Error responses are formatted as JSON to provide clear and actionable feedback.

5. Security:
- The service assumes the existence of proper AWS credentials.
- Only users with appropriate IAM policies (e.g., s3:ListBucket) will be able to access the bucket contents.

Assumptions:

1. AWS Credentials:
AWS credentials (access key ID and secret access key) must be set up in the environment.

2. S3 Bucket:
The S3 bucket (one2n-assignment-bucket in the code) exists and is accessible with the provided AWS credentials.

3. IAM Permissions:
The IAM role or user running the service must have the s3:ListBucket permission for the specified S3 bucket.

4. Prefix Handling:
The service treats provided paths as prefixes within the bucket. If no prefix is specified, it defaults to listing the root contents.

Part 2:

Problem Statement:
Write a Terraform layout to provision infrastructure on AWS and deploy the above code.

Overview:
The Terraform configuration automates the creation of AWS infrastructure required to deploy the above Python HTTP service code which interacts with the S3 bucket. The configuration provisions a S3 bucket, IAM roles, an EC2 instance, and a security group, with everything necessary to run the HTTP service that lists the contents of the S3 bucket with the help of the Python Code above.

Key Features:
- S3 Bucket Creation: Creates an S3 bucket and sets up directories (prefixes) within it.
- IAM Role: Defines an IAM role with the permissions for EC2 to access the S3 bucket.
- EC2 Instance: Provisions an EC2 instance that runs the Flask-based Python HTTP service, allowing the user to interact with the S3 bucket.
- Security Group: Configures a security group to allow HTTP and SSH traffic.
- Output: Provides the public URL of the running Flask application after provisioning.

Design Decisions:

1. AWS Provider Configuration:
- The AWS provider is configured to use us-east-1 as the default region.
- The Terraform configuration specifies the required provider version (~> 5.0), ensuring compatibility with the latest version of the 'hashicorp/aws' provider.

2. S3 Bucket and Directories:
- An S3 bucket, one2n-assignment-bucket, is created with two parent directories and one child directory inside one of the parent directories. These directories are represented as S3 objects with a trailing '/' in their keys, allowing them to act as "folders" in the S3 bucket.

3. IAM Role and Policy for EC2:
- An IAM role terraform_access_role is created, allowing EC2 instances to assume the role. This role is then assigned a policy (iam_role_policy.json) that grants the necessary permissions to access the S3 bucket.
- The EC2 instance also has an instance profile (terraform_access_instance_profile) attached to it, which is necessary to allow the EC2 instance to assume the role and interact with the S3 bucket.

4. EC2 Instance with User Data:
- An EC2 instance (t2.micro instance type) is provisioned with the specified AMI (ami-01816d07b1128cd2d). The instance is configured to run the Python HTTP service (Flask and Boto3) that interacts with the newly created S3 bucket.
- The user data script installs the required packages (python3, pip3, flask, and boto3), sets up the application, and starts the Flask app on port 5000.

5. Security Group:
- A security group is configured to allow SSH access (port 22) and HTTP traffic (port 5000) to the EC2 instance. This ensures that the instance is accessible for both management and interaction with the HTTP service.

6. Output:
- Once the infrastructure is deployed, Terraform outputs the public URL of the running Flask application, which provides access to the S3 bucket listing service.

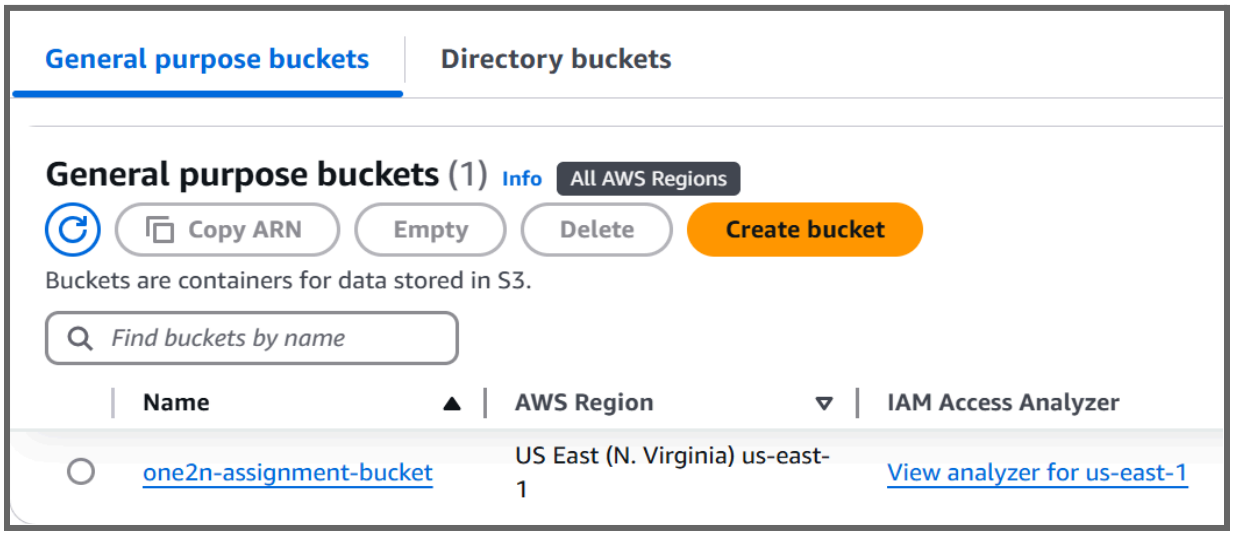
Assumptions:

AMI ID: The AMI (ami-01816d07b1128cd2d) is a pre-configured Amazon Linux AMI available in the selected region (us-east-1). It may need to be updated depending on other AWS regions.

S3 Bucket Structure:

```
one2n-assignment-bucket
|
- parent-dir-1
|    |
|    - child-dir-1
|
- parent-dir-2
```

Snippets of S3 Bucket and Directory Structure:


General purpose buckets | Directory buckets

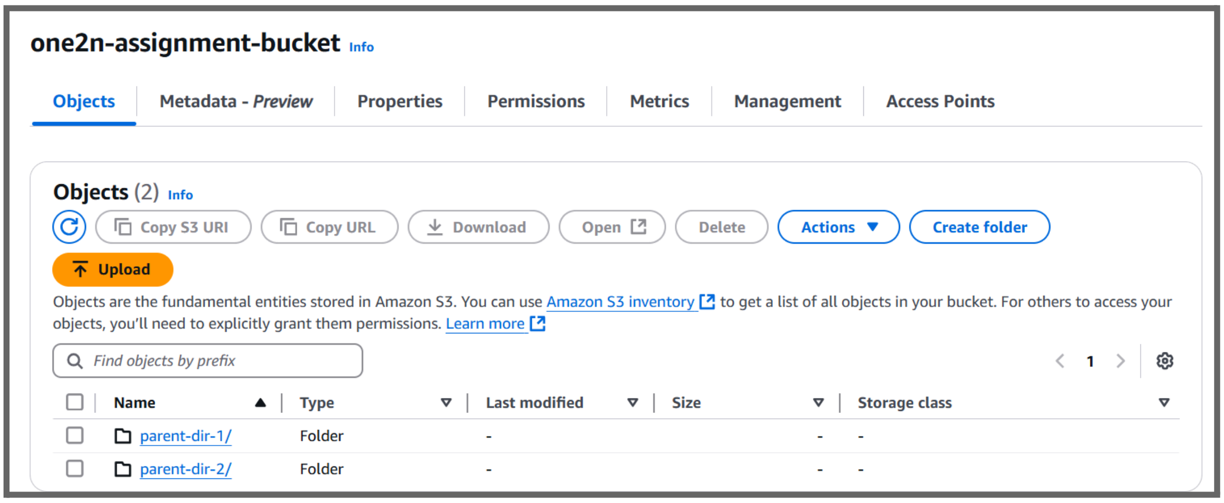
General purpose buckets (1) Info All AWS Regions

Copy ARN | Empty | Delete | Create bucket

Buckets are containers for data stored in S3.

Find buckets by name

Name	AWS Region	IAM Access Analyzer
one2n-assignment-bucket	US East (N. Virginia) us-east-1	View analyzer for us-east-1


one2n-assignment-bucket Info

Objects | Metadata - *Preview* | Properties | Permissions | Metrics | Management | Access Points

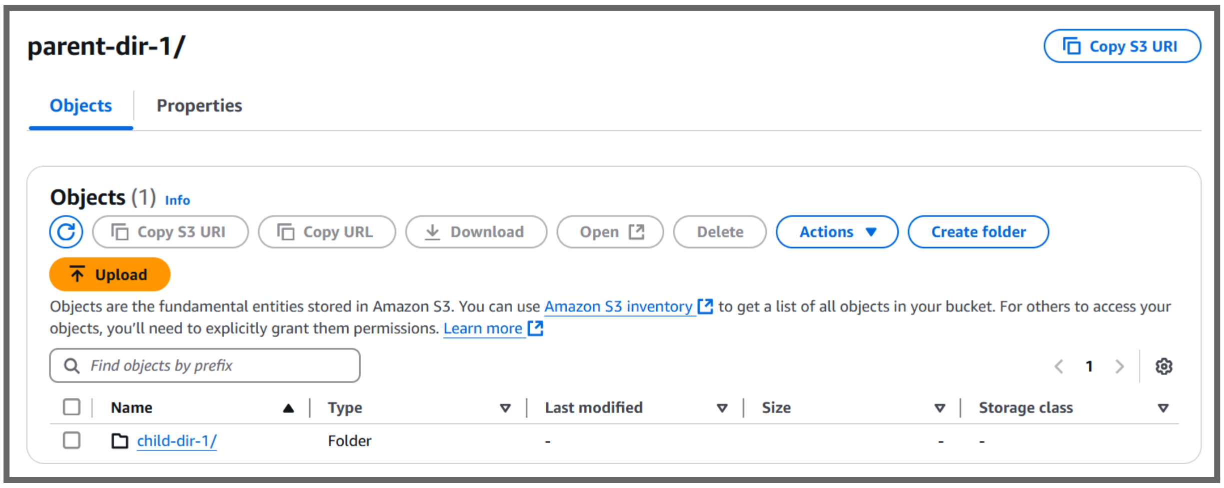
Objects (2) Info

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

Name	Type	Last modified	Size	Storage class
parent-dir-1/	Folder	-	-	-
parent-dir-2/	Folder	-	-	-


parent-dir-1/ Copy S3 URI

Objects | Properties

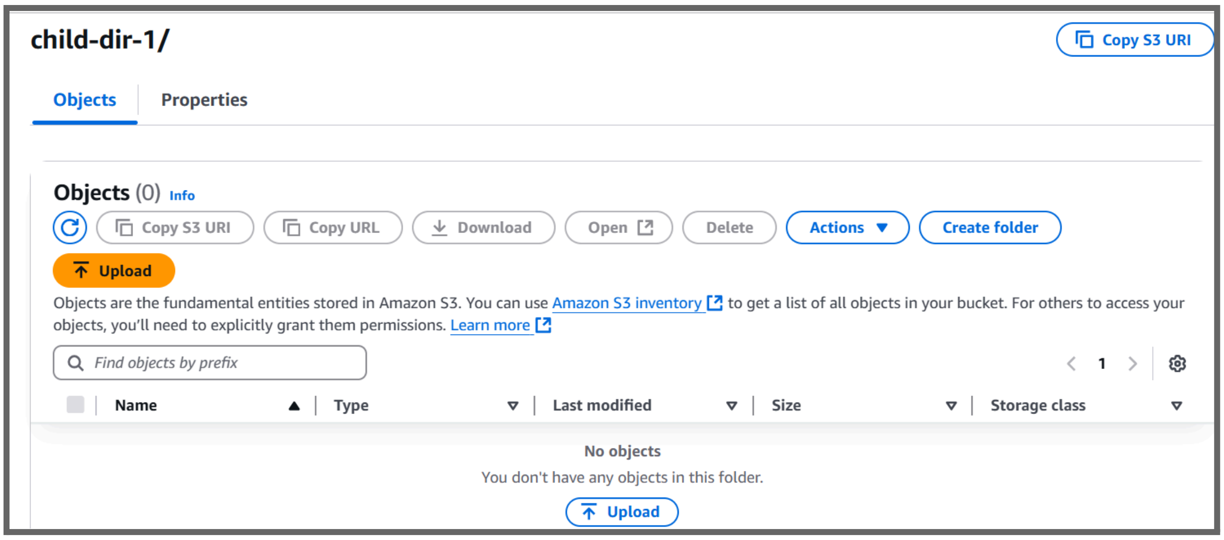
Objects (1) Info

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

Name	Type	Last modified	Size	Storage class
child-dir-1/	Folder	-	-	-



child-dir-1/

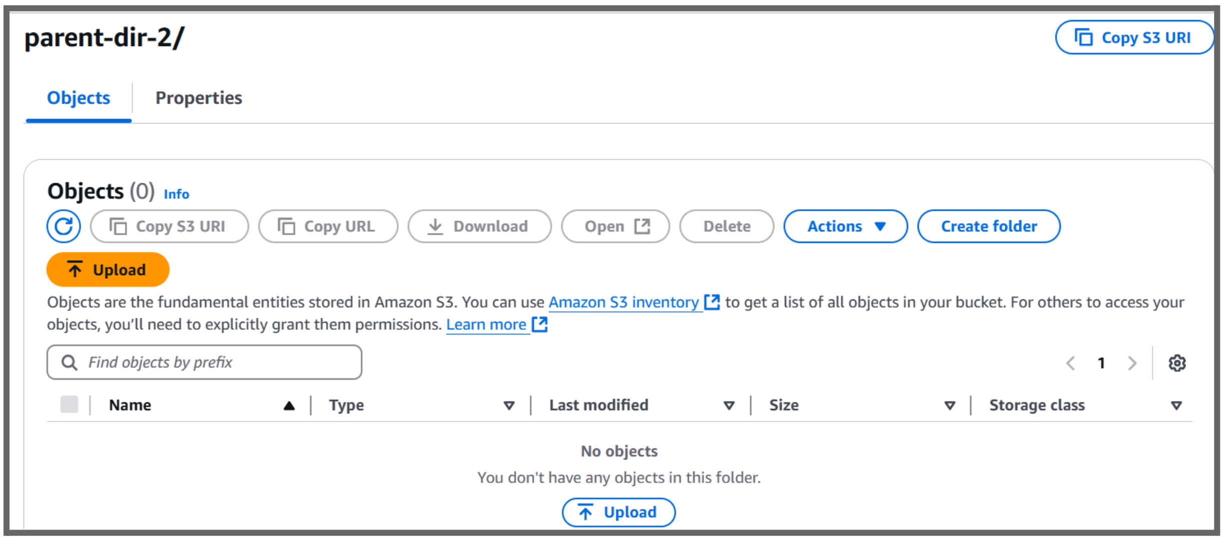
Objects (0)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

	Name	Type	Last modified	Size	Storage class

No objects

You don't have any objects in this folder.



parent-dir-2/

Objects (0)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

	Name	Type	Last modified	Size	Storage class

No objects

You don't have any objects in this folder.

API Responses as per the Demo Video:

```
"GET /list-bucket-content/ HTTP/1.1" 200 -
"GET /list-bucket-content/parent-dir-1 HTTP/1.1" 200 -
"GET /list-bucket-content/parent-dir-1/child-dir-1 HTTP/1.1" 200 -
"GET /list-bucket-content/parent-dir-2 HTTP/1.1" 200 -
"GET /list-bucket-content/parent-dir-2/new-dir HTTP/1.1" 404 -
"GET /list-bucket-content/parent-dir-1/child-dir-1/child-dor HTTP/1.1" 404 -
```

Note:

I have also shared a Live Working Demo (In 3 Parts) with a complete Explanation of the Assignment Workflow in the same Git Hub Repo.