AWS Persistent Storage

1. Block Storage

Block storage works at the operating system level and is commonly used for storing data that requires frequent reads/writes, such as databases, applications, and virtual machines.

Amazon EBS (Elastic Block Store)

- What it is: A durable, high-performance storage service designed for Amazon EC2 instances.
- How it works:
    - You create an EBS volume, attach it to an EC2 instance, and format it with a file system.
    - The volume behaves like a physical hard drive but exists independently of the instance.
    - If an instance is stopped or terminated, the EBS volume remains intact.
- Features:
    - Supports SSD (gp3, gp2, io1, io2) and HDD (st1, sc1) storage.
    - Can be encrypted for security.
    - Snapshots allow backup and recovery by storing copies in Amazon S3.
    - Can be resized without data loss.

Instance Store (Temporary Block Storage for EC2 Instances)

- What it is: High-speed storage that comes pre-attached to an EC2 instance but is temporary.
- How it works:
    - Storage is physically attached to the instance.
    - Data is lost if the instance is stopped, hibernated, or terminated.

- Use case: Suitable for temporary data like cache, buffers, or scratch space where persistence is not required.
- Limitations:
    - Data is not persistent beyond instance shutdown.
    - Cannot be detached or moved between instances.

2. Object Storage

Object storage treats data as objects rather than files or blocks, making it ideal for storing large amounts of unstructured data like images, videos, backups, and logs.

Amazon S3 (Simple Storage Service)

- What it is: A highly scalable, durable, and cost-effective object storage service.
- How it works:
    - Data is stored as objects inside buckets (like folders).
    - Each object has metadata and a unique identifier for retrieval.
    - Objects can be accessed via API, AWS CLI, or web browser.
- Features:
    - Stores and retrieves any amount of data from anywhere.
    - Durability: 99.999999999% (11 nines) reliability.
    - Lifecycle policies allow automatic data transitions (e.g., moving old files to lower-cost storage like S3 Glacier).
    - Used for backups, disaster recovery, data lakes, and media storage.

3. File Storage

File storage provides a shared file system accessible by multiple EC2 instances. It is useful for applications that need concurrent access to the same data, such as content management systems or distributed applications.

Amazon EFS (Elastic File System) (Linux Instances Only)

- What it is: A managed, scalable, and elastic file system for Linux-based applications.
- How it works:
    - Multiple EC2 instances can mount the same EFS file system at the same time.
    - Uses the NFS (Network File System) protocol for access.
    - Automatically scales based on usage.
- Features:
    - Supports both Standard and Infrequent Access (IA) storage classes.
    - Good for shared workloads, container storage, and analytics.

Amazon FSx

- What it is: A managed service providing high-performance file systems.
- How it works:
    - Unlike EFS (which is for Linux), FSx supports Windows and high-performance workloads.
    - Offers different file systems for different use cases:
        - FSx for Windows File Server (SMB, NTFS) → Windows applications.
        - FSx for Lustre → High-performance computing (HPC).
        - FSx for NetApp ONTAP → Enterprise workloads with advanced features.
        - FSx for OpenZFS → Cost-effective storage for Linux workloads.
- Use cases:
    - Running Windows-based applications.
    - High-performance computing (HPC) requiring fast data processing.
    - Enterprise file storage needs.

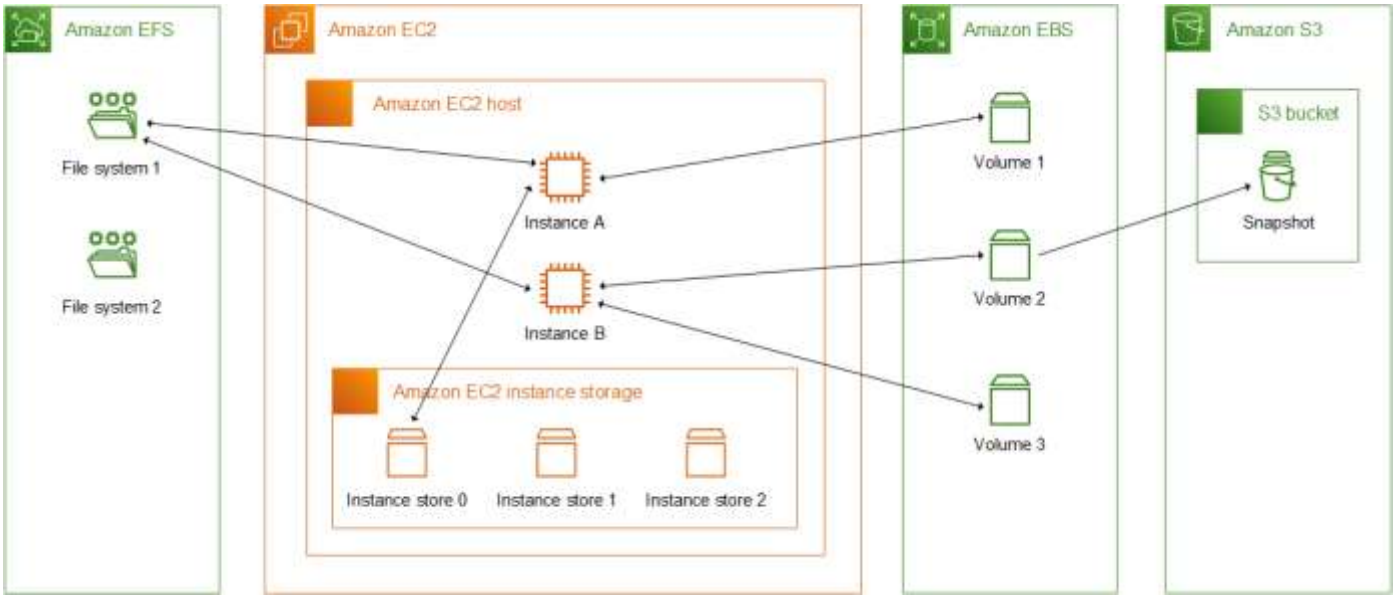
4. File Caching

File caching improves performance by providing fast access to frequently used data stored in remote locations.

Amazon File Cache

- What it is: A high-speed, temporary cache that accelerates access to remote file systems.

- How it works:
    - Speeds up workloads by caching remote data close to compute resources.
    - Reduces latency for applications accessing files over a network.
- Use cases:
    - Applications needing low-latency access to large datasets.
    - Hybrid cloud environments where data is stored in multiple locations.



Links:

[Storage options for your Amazon EC2 instances - Amazon Elastic Compute Cloud]

[Amazon EBS]

[Instance store temporary block storage for EC2 instances]

[Amazon EFS]

[Amazon FSx]

[Use Amazon File Cache with Amazon EC2 instances]

Hunter’s additions:

How do we actually connect these?

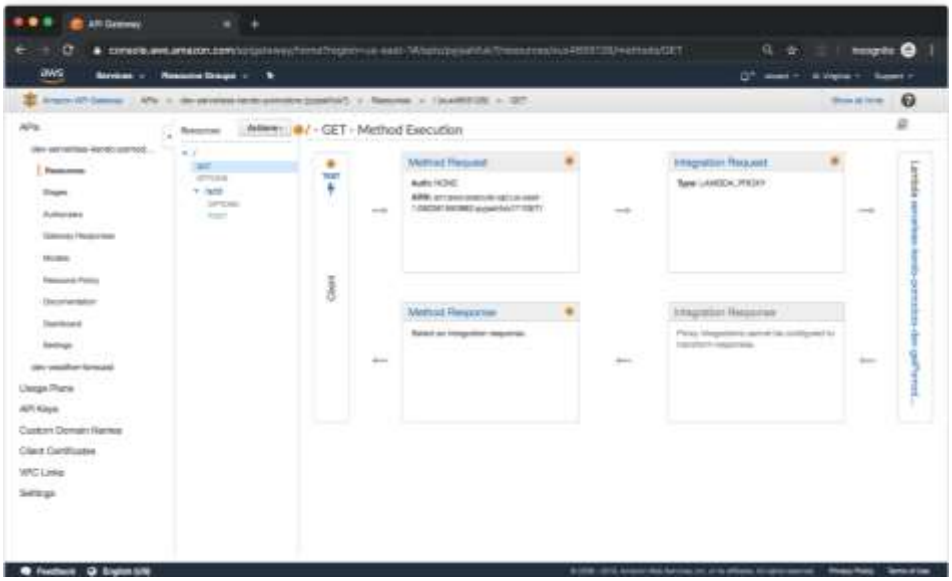
AWS API Gateways:

“Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud.”

API gateways are the connection between different services/containers and other connected applications. They handle processing all API calls. This can range from anything from monitoring, access control, and traffic management.

“API Gateway acts as a ‘front door’ for applications…”

AWS has a management console which gives a web interface for managing APIs.



As a part of using Amazon APIs, you have to use what are known as Lambda functions. You use a Lambda function for the backend of your API. Lambda runs your code only when needed and scales automatically.

RESTful API is what we will be covering in class (and thus will likely use for the project). Representational State Transfer (REST) is a software architecture that imposes conditions on how an API should work. REST was initially created as a guideline to manage communication on a complex network like the internet. You can use REST-based architecture to support high-performing and reliable communication at scale.

Code on demand: In REST architectural style, servers can temporarily extend or customize client functionality by transferring software programming code to the client. For example, when you fill a registration form on any website, your browser immediately highlights any mistakes you make, such as incorrect phone numbers. It can do this because of the code sent by the server.

Here is Amazon's walk-through of how to set up a REST API console:

Step 1: Create a Lambda function

You use a Lambda function for the backend of your API. Lambda runs your code only when needed and scales automatically, from a few requests per day to thousands per second.

For this exercise, you use a default Node.js function in the Lambda console.

To create a Lambda function

1. Sign in to the Lambda console at https://console.aws.amazon.com/lambda.
2. Choose **Create function**.
3. Under **Basic information**, for **Function name**, enter **my-function**.
4. For all other options, use the default setting.
5. Choose **Create function**.

The default Lambda function code should look similar to the following:

```
export const handler = async (event) => {
  const response = {
    statusCode: 200,
    body: JSON.stringify('The API Gateway REST API console is great!'),
  };
  return response;
};
```

You can modify your Lambda function for this exercise, as long as the function's response aligns with the format that API Gateway requires.

Replace the default response body (Hello from Lambda!) with The API Gateway REST API console is great!. When you invoke the example function, it returns a 200 response to clients, along with the updated response.

Step 2: Create a REST API

Next, you create a REST API with a root resource (/).

To create a REST API

1. Sign in to the API Gateway console at https://console.aws.amazon.com/apigateway.
2. Do one of the following:
    - To create your first API, for **REST API**, choose **Build**.
    - If you've created an API before, choose **Create API**, and then choose **Build** for **REST API**.
3. For **API name**, enter **my-rest-api**.
4. (Optional) For **Description**, enter a description.
5. Keep **API endpoint type** set to **Regional**.
6. Choose **Create API**.

Step 3: Create a Lambda proxy integration

Next, you create an API method for your REST API on the root resource (/) and integrate the method with your Lambda function using a proxy integration. In a Lambda proxy integration, API Gateway passes the incoming request from the client directly to the Lambda function.

To create a Lambda proxy integration

1. Select the / resource, and then choose **Create method**.
2. For **Method type**, select ANY.
3. For **Integration type**, select **Lambda**.
4. Turn on **Lambda proxy integration**.
5. For **Lambda function**, enter **my-function**, and then select your Lambda function.
6. Choose **Create method**.

Step 4: Deploy your API

Next, you create an API deployment and associate it with a stage.

To deploy your API

1. Choose **Deploy API**.
2. For **Stage**, select **New stage**.

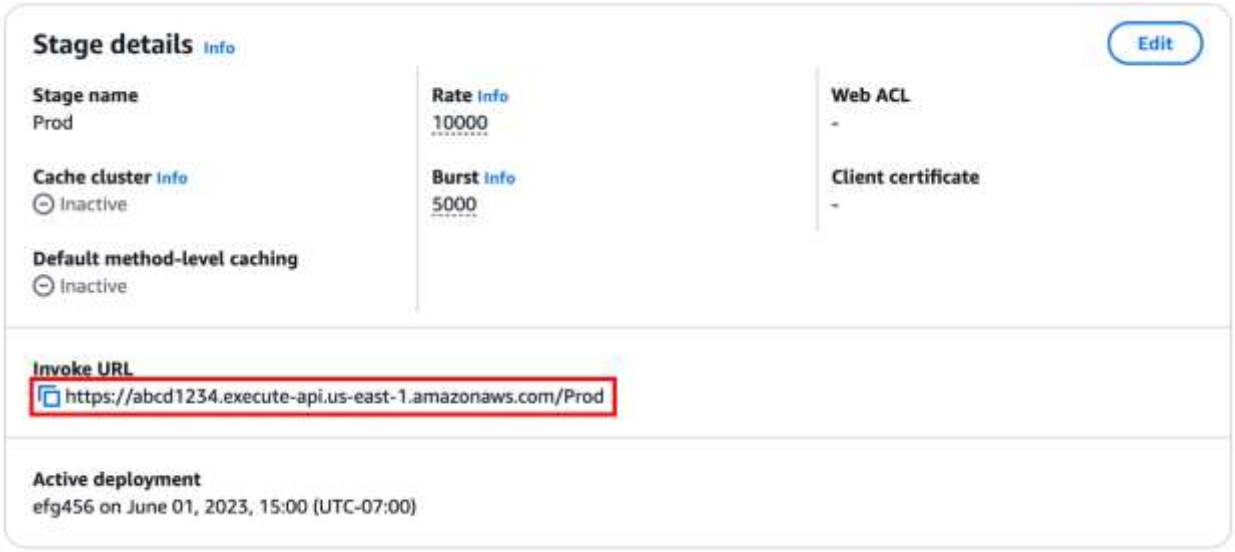
3. For **Stage name**, enter **Prod**.
4. (Optional) For **Description**, enter a description.
5. Choose **Deploy**.

Now clients can call your API. To test your API before deploying it, you can optionally choose the **ANY** method, navigate to the **Test** tab, and then choose **Test**.

Step 5: Invoke your API

To invoke your API

1. From the main navigation pane, choose **Stage**.
2. Under **Stage details**, choose the copy icon to copy your API's invoke URL.



3. Enter the invoke URL in a web browser.

The full URL should look like https://*abcd123*.execute-api.*us-east-2*.amazonaws.com/Prod.

Your browser sends a GET request to the API.

4. Verify your API's response. You should see the text "The API Gateway REST API console is great!" in your browser.

Links:

https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html#api-gateway-overview-aws-backbone

http://docs.aws.amazon.com/apigateway/latest/developerguide/getting-started.html

https://docs.aws.amazon.com/apigateway/latest/developerguide/getting-started-rest-new-console.html

https://aws.amazon.com/what-is/restful-api/