AWS

- AWS products: S3, SQS, SNS, MediaConvert
- Terraform on AWS

AWS products: S3, SQS, SNS, MediaConvert

AWS offers a suite of integrated products for cloud storage, messaging, and media processing. The services mentioned are:

Amazon S3 (Simple Storage Service): A scalable and durable **object storage** service for storing and retrieving any amount of data from anywhere on the web. It is commonly used as the primary storage for input and output files for other AWS services like MediaConvert.

Amazon SQS (Simple Queue Service): A fully managed **message queuing** service that enables you to decouple and scale microservices, distributed systems, and serverless applications. It uses a polling model where consumers retrieve messages from the queue.

Amazon SNS (Simple Notification Service): A fully managed **pub/sub messaging** service that provides high-throughput, push-based delivery of messages to multiple subscribers (including SQS queues, Lambda functions, HTTP endpoints, and email addresses) simultaneously.

AWS Elemental MediaConvert: A file-based **video transcoding** service that allows you to convert video files into formats optimized for broadcast and multi-screen delivery (e.g., adaptive bitrate streaming for web/mobile) without managing any underlying infrastructure.

Common Use Case Scenario

These services often work together in decoupled, event-driven architectures (see figure below for a conceptual flow):

A user uploads a raw video file to an **Amazon S3** input bucket.

An S3 event notification triggers an **Amazon SNS** topic or directly invokes an AWS Lambda function when a new object is created.

The SNS topic (or Lambda function) initiates a transcoding job in **AWS Elemental MediaConvert**, providing the S3 input and output locations.

MediaConvert processes the file, and its job status changes (e.g., "job complete") trigger **Amazon CloudWatch** events.

These CloudWatch events can be routed to an **Amazon SQS** queue, where a downstream application can asynchronously poll for job completion status and perform post-processing tasks,

such as updating a database or generating a thumbnail from the transcoded output.

For more detailed information on each service, you can visit the official AWS documentation

Terraform on AWS

Terraform on AWS offers various approaches and considerations for managing infrastructure as code. These different ways relate to how you structure your configurations, manage state, and integrate with other tools.

1. Configuration Organization:

A single `main.tf` file or a small set of files defining all your AWS resources. This can be suitable for small, simple projects.

Breaking down your infrastructure into reusable modules. This promotes code reusability, organization, and consistency, especially for larger or more complex environments. You can create modules for common resource patterns like VPCs, EC2 instances with specific configurations, or application stacks.

Using workspaces or separate directories/repositories for different environments (e.g., dev, staging, production) to manage environment-specific configurations and variables.

2. State Management:

Storing the `terraform.tfstate` file locally on your machine. This is simple for individual use but not recommended for team environments due to potential conflicts and data loss.

Storing the state file in a shared, remote location like an S3 bucket with DynamoDB locking. This enables collaboration, prevents state corruption, and provides versioning and auditing capabilities.

3. Provider Usage:

The primary Terraform provider for interacting with AWS services. It offers extensive coverage and is widely used.

This provider leverages the AWS Cloud Control API to automatically generate support for new AWS services and features more quickly

than the standard provider, addressing potential coverage gaps.

4. Integration and Automation:

Integrating Terraform into your CI/CD pipeline (e.g., Jenkins, GitLab CI, GitHub Actions) to automate infrastructure provisioning, testing, and deployment.

Utilizing HashiCorp's managed services for enhanced collaboration, remote state management, policy enforcement (Sentinel), and cost optimization features.

While not a direct "way" of using Terraform, you can use AWS CDK to define your cloud application resources using familiar programming languages and then generate Terraform HCL state files for provisioning with Terraform.

5. Authentication and Permissions:

Using access keys and secret keys, typically stored as environment variables or in a `~/.aws/credentials` file.

Leveraging IAM roles for EC2 instances or instance profiles for control servers to provide temporary, fine-grained permissions to Terraform for interacting with AWS. This is a more secure approach than hardcoding credentials.