Using **Envoy on its own** versus using **Envoy with Istio** involves key differences in deployment, management, and functionality. Here's a breakdown of the major differences:

**1. Control Plane Management**

* **Envoy on its own**:
  + Envoy is a standalone **data plane** component. When used alone, you must manually configure Envoy’s settings, including service discovery, traffic routing, load balancing, retries, timeouts, and circuit breaking. The configuration is usually done via static files or APIs like xDS.
  + You need to build your own control plane to manage Envoy instances or rely on third-party tools to automate these tasks.
  + You have full control over Envoy’s setup but need to manage all aspects, which can be complex for large-scale systems.
* **Envoy with Istio**:
  + Istio provides a **control plane** that automatically manages Envoy sidecars deployed across your services. Istio’s control plane (Pilot, Mixer, Citadel) dynamically configures Envoy sidecars using the xDS protocol, which abstracts away much of the manual effort required to manage the data plane.
  + Istio takes care of service discovery, load balancing, traffic management, policy enforcement, security (mTLS), and observability, all out-of-the-box.
  + Using Istio reduces the operational burden by providing centralized control over Envoy instances, especially for larger systems.

**2. Service Mesh Functionality**

* **Envoy on its own**:
  + Envoy is a powerful proxy with built-in features such as:
    - Traffic routing and load balancing
    - Circuit breaking and retries
    - Rate limiting
    - Observability (metrics, logs, and traces)
    - Advanced layer 7 (L7) handling for HTTP/2, gRPC, WebSockets
  + While you get the raw power of Envoy, you need to build and configure a system around it to fully implement service mesh features like security policies, global traffic management, and observability.
  + Manual work is required for service mesh-like behavior, such as mutual TLS (mTLS), security policies, and access controls.
* **Envoy with Istio**:
  + Istio turns Envoy into a fully-managed **service mesh** by automatically injecting Envoy sidecars into every service pod.
  + Istio enhances Envoy’s native capabilities with:
    - **Traffic Control**: Advanced routing, traffic splitting, A/B testing, canary deployments, and shadowing.
    - **Security**: Out-of-the-box mutual TLS (mTLS), identity-based access control, and certificate management.
    - **Observability**: Centralized logging, distributed tracing (e.g., Jaeger), and detailed metrics (e.g., Prometheus) with minimal setup.
    - **Policy Enforcement**: Fine-grained control over authorization and rate limiting.
  + Istio’s service mesh makes it easier to implement consistent security, traffic policies, and observability across multiple services in a Kubernetes environment without needing to configure each Envoy instance manually.

**3. Operational Complexity**

* **Envoy on its own**:
  + You are responsible for managing and deploying Envoy proxies across your environment. This includes manual setup, configuration, and updates.
  + You need to define and maintain your own system for managing configurations, scaling Envoy instances, and service discovery.
  + More operational complexity, but potentially more flexibility if you only need specific proxying capabilities.
* **Envoy with Istio**:
  + Istio automates much of the operational overhead of deploying and managing Envoy proxies.
  + Istio provides a unified configuration and management interface, allowing you to configure and control all Envoy proxies centrally, using high-level CRDs (e.g., VirtualService, DestinationRule).
  + Istio’s complexity lies in its initial setup and the learning curve of using Istio’s extensive features, but it simplifies operations for large, microservices-based environments.

**4. Security**

* **Envoy on its own**:
  + Envoy provides the ability to enforce security policies, like rate limiting and access control, but you must configure them manually.
  + You need to implement mutual TLS (mTLS) and certificate rotation yourself if required, which can be complicated.
* **Envoy with Istio**:
  + Istio provides automated **mTLS** between services, enforcing strong identity-based access control with built-in certificate management and rotation.
  + Security policies are centrally defined and applied across all services, ensuring consistent security behavior across your mesh.

**5. Deployment Model**

* **Envoy on its own**:
  + You can deploy Envoy as a standalone proxy or sidecar, but you need to handle service discovery, routing, and failover manually or via external systems.
  + This is suitable for smaller systems or custom deployments where you need precise control over your proxy and service communication.
* **Envoy with Istio**:
  + Istio injects Envoy as a **sidecar proxy** into each service pod, meaning all service-to-service communication is routed through the Envoy proxy automatically.
  + Istio’s architecture is designed for Kubernetes environments and microservices, providing seamless integration and management at scale.

**Summary of Key Differences:**

| **Aspect** | **Envoy on its Own** | **Envoy with Istio** |
| --- | --- | --- |
| **Control Plane** | Manual configuration | Automated by Istio control plane |
| **Service Mesh Features** | Manual setup for mesh-like features | Fully managed service mesh with mTLS, observability, and traffic control |
| **Security** | Requires manual configuration | Built-in mTLS, certificate management, and security policies |
| **Operational Complexity** | High manual effort | Simplified with Istio’s centralized management |
| **Traffic Management** | Manually configured | Advanced traffic routing and control via CRDs |
| **Observability** | Requires third-party tools | Built-in observability, logging, and tracing with Istio |
| **Deployment Model** | Standalone or custom sidecar deployments | Automated sidecar injection via Istio |

**When to Use Each:**

* **Envoy on its own** is ideal for simpler, smaller-scale environments where you need specific proxy features without the complexity of a full service mesh.
* **Envoy with Istio** is better suited for managing large-scale, microservices-based systems where centralized control, security, observability, and traffic management are critical.

In general, if you need a fully-featured service mesh and don't want to manage everything manually, using Envoy with Istio is the better choice. However, if you only need some of Envoy's proxying features without the overhead of Istio, running Envoy standalone may suffice.

Envoy is primarily a **reverse proxy** and **service proxy** designed for microservices architectures. Here's a breakdown of the types of proxying Envoy typically performs:

**1. Reverse Proxy**

* **How Envoy Uses It**:
  + Envoy acts as a reverse proxy by routing external requests from clients to backend services in a microservices environment. It handles the incoming requests, applies routing rules, and forwards them to the appropriate backend service.
  + **Common in microservice architectures**, where Envoy sits at the edge of the network or in front of services, controlling traffic flow and ensuring load balancing, SSL termination, and traffic management.
* **Features Envoy Offers as a Reverse Proxy**:
  + **Load balancing**: Distributes incoming requests across multiple services or instances.
  + **TLS Termination**: Offloads SSL/TLS decryption from backend services.
  + **Observability**: Provides extensive metrics, logs, and tracing for the traffic passing through it.
  + **Resilience**: Implements retries, circuit breaking, and fault injection for more reliable communication.
  + **Caching**: Can cache responses to improve performance.

**2. Service Proxy (Sidecar Proxy in Service Mesh)**

* **How Envoy Uses It**:
  + In a **service mesh** (such as Istio), Envoy operates as a **sidecar proxy**, which means it sits next to individual services or microservices within the same host. Each service communicates with Envoy, which handles service-to-service communication.
  + It ensures that all traffic between services (east-west traffic) passes through the Envoy sidecar, providing features like traffic management, security (e.g., mTLS), and monitoring.
* **Features as a Service Proxy**:
  + **Service Discovery**: Routes traffic based on dynamic service discovery.
  + **Load balancing**: Distributes traffic between instances of services.
  + **mTLS (mutual TLS)**: Encrypts service-to-service communication for security.
  + **Traffic Management**: Allows for advanced routing (e.g., canary releases, blue-green deployments).
  + **Telemetry**: Provides monitoring, logging, and tracing for intra-service communication.
  + **Rate Limiting** and **Quota Management**: Controls the rate of requests sent to services.

To estimate the **hourly cost** of using an **Amazon EKS stack** based on the Terraform configuration provided, we will break down the following key components:

**1. EKS Control Plane (Cluster)**

* **EKS Control Plane** has a flat rate of **$0.10 per hour** for each cluster. This charge is independent of the number of worker nodes running in the cluster.

**2. Managed Node Groups (EC2 Instances)**

You have an **EKS managed node group (mg\_5)** with the following configuration:

* **Desired size**: 2 nodes (minimum and maximum set to 2)
* **Instance types**: Mixed, with some instance types priced differently. I'll estimate costs using one of the most common instances: **m5.large**.
* **Instance pricing for m5.large** (2 vCPU, 8 GiB memory):
  + On-Demand: **$0.096 per hour** (per instance) in the **US East (N. Virginia)** region (pricing may vary by region).
  + Since there are **2 nodes** in the group:
    - Hourly cost: **$0.096 x 2 = $0.192 per hour**.

**3. VPC (Networking)**

* **VPC Charges**: There are no additional charges for using VPC itself, but there could be costs for:
  + **NAT Gateway**: Priced at **$0.045 per hour**.
  + **Data Transfer**: Not included here, but be aware of potential charges for data transfer out of AWS (e.g., internet access).

**4. Elastic Block Store (EBS) Volumes**

* Each EC2 instance will likely have **EBS volumes** for storage.
* By default, **Amazon Linux 2** instances use **8 GiB of EBS General Purpose (gp3)** storage.
* EBS pricing: **$0.08 per GiB per month**.
  + For two instances with 8 GiB each, the hourly cost is minimal:
  + **$0.08 per GiB per month** ≈ **$0.00011 per GiB per hour**.
  + **Total hourly cost for EBS**: **$0.00011 x 16 GiB** = **$0.00176 per hour**.

**5. Cluster Add-ons**

* **Cluster Add-ons**: kube-proxy, CoreDNS, and VPC CNI are part of the EKS cluster and do not incur additional charges beyond the control plane cost.

**6. CloudWatch Logs**

* CloudWatch is disabled in this configuration, so there are no additional costs for logging.

**7. Karpenter**

* The cost of Karpenter itself (the autoscaler) does not incur extra charges. However, the instances it scales would be reflected in the EC2 pricing.

**Cost Breakdown**

| **Service** | **Hourly Cost** | **Description** |
| --- | --- | --- |
| **EKS Control Plane** | **$0.10 per hour** | Flat rate for EKS cluster |
| **Managed Node Group (EC2)** | **$0.192 per hour** | 2 x m5.large instances ($0.096 per hour each) |
| **VPC NAT Gateway** | **$0.045 per hour** | Estimated cost for 1 NAT Gateway |
| **EBS Volumes (16 GiB total)** | **$0.00176 per hour** | Based on 8 GiB per node (gp3) |
| **CloudWatch Logs** | **$0** | Not enabled in this configuration |

**Total Hourly Cost:**

**$0.10 (EKS control plane) + $0.192 (EC2 instances) + $0.045 (NAT Gateway) + $0.00176 (EBS)** = **$0.338 per hour**

**Additional Considerations:**

* **Data Transfer**: Outbound traffic (e.g., internet traffic) is charged separately.
* **Different Instance Types**: If you use other instance types (e.g., m4.large, t3.large), the cost will vary.
* **Scaling**: Karpenter could scale up/down the number of instances, impacting the overall cost.

Let me know if you'd like a more specific estimate for different instance types or additional components!