Given that **Kapistio** is built using **UnityHub** and **AWS EKS (Elastic Kubernetes Service)**, your delivery method will largely depend on how your users will interact with your tool and the infrastructure it requires. Let’s break down the options, considering that UnityHub is primarily a **local desktop application** and AWS EKS is a **cloud-based orchestration platform** for managing containerized applications. Here are the most relevant delivery methods for your product:

**1. Software as a Service (SaaS)**

* **What is it?**
  + In this model, Kapistio would be delivered as a fully managed service. Your customers interact with your application through a web interface (SaaS front-end) and you handle all the back-end infrastructure, including UnityHub and AWS EKS integration, on your end.
* **Use Case:**
  + **Kapistio** users only interact with the tool via a web interface, and all UnityHub processing (e.g., rendering, 3D modeling) happens in the cloud on AWS infrastructure (EKS). Users don’t need to install any software locally.
* **Advantages:**
  + Easier for customers to adopt (no installation required).
  + Automatic updates and maintenance are handled by you.
  + Can scale the back-end infrastructure (AWS EKS) to meet demand.
* **Challenges:**
  + You will need to host and manage both the UnityHub engine and AWS EKS infrastructure.
  + Pricing might need to reflect the continuous operational costs (AWS compute, storage, etc.).
* **Best For:**
  + Customers looking for a completely cloud-based solution without having to manage any infrastructure.

**2. Hybrid Delivery (On-Premises + Cloud)**

* **What is it?**
  + In a **hybrid delivery model**, part of the tool (UnityHub) is delivered for **on-premises/local use**, while **AWS EKS** manages the cloud-based components. The user would install the UnityHub client locally to handle interactions like rendering or 3D model manipulation, but the heavy-lifting processing (e.g., data uploads, model processing) happens on the cloud via AWS EKS.
* **Use Case:**
  + Users need **local rendering** or other tasks performed by UnityHub, but you still offload **data processing** and other services (e.g., scaling, distributed processing) to AWS EKS in the cloud.
* **Advantages:**
  + Combines the flexibility of cloud resources with the power of local computing.
  + Users get the benefits of using UnityHub locally for responsive tasks while AWS EKS handles cloud-based scaling for processing.
* **Challenges:**
  + Requires maintaining both a local application (UnityHub) and a cloud infrastructure.
  + Syncing between local (on-prem) and cloud resources can introduce complexity.
* **Best For:**
  + Scenarios where local computing (UnityHub) is needed for responsive interactions and large-scale data processing happens on AWS EKS.

**3. Container-Based Delivery (ECS/EKS)**

* **What is it?**
  + This delivery method involves deploying your software in **containers**, which could be orchestrated using **AWS EKS** (Elastic Kubernetes Service). You package your application (UnityHub and related services) in Docker containers, and customers can deploy them into their own EKS or ECS environments.
* **Use Case:**
  + If your customers need control over the infrastructure, they can run the Kapistio containerized applications on their own AWS accounts using **EKS**.
* **Advantages:**
  + Portability: Your customers can deploy Kapistio wherever they want (EKS, ECS, on-premise Kubernetes clusters).
  + Scalability: Containers are easily scalable in AWS, and customers can dynamically scale workloads.
* **Challenges:**
  + Customers will need to be familiar with container orchestration and Kubernetes.
  + You must ensure your application is correctly containerized, and all dependencies are managed within the container.
* **Best For:**
  + Customers who are already using container-based environments and want control over their AWS EKS setup but still leverage the UnityHub functionality in containers.

**4. Amazon Machine Image (AMI)**

* **What is it?**
  + In this model, you deliver a pre-configured **Amazon Machine Image (AMI)** that contains both **UnityHub** and your backend processing (or EKS interaction). Customers can launch an EC2 instance using your AMI with everything pre-installed and configured.
* **Use Case:**
  + If your product needs to run on dedicated EC2 instances, with UnityHub installed and AWS EKS interactions pre-configured for certain workloads, the customer can launch an EC2 instance with your AMI.
* **Advantages:**
  + Ready-to-go solution with UnityHub and EKS integration.
  + No container or Kubernetes orchestration knowledge is required for your customer.
* **Challenges:**
  + AMI is typically tied to EC2 instances, which limits flexibility in environments (compared to SaaS or containers).
  + You need to maintain AMIs for updates.
* **Best For:**
  + Customers who want to quickly deploy Kapistio in their AWS environment using EC2 instances without worrying about complex configurations.

**5. Bring Your Own License (BYOL)**

* **What is it?**
  + The **BYOL** model lets customers bring their own existing UnityHub licenses and deploy Kapistio (or a part of it) using either AMIs or containers. This means the user already has the licenses required to use UnityHub and simply deploys your application in AWS EKS or EC2.
* **Use Case:**
  + If your customers already own licenses for UnityHub or prefer to manage licenses separately, this model allows them to use their existing licenses while leveraging your product.
* **Advantages:**
  + Flexibility for customers to use their own licenses, reducing licensing costs on your end.
  + Customers can deploy the application in the cloud (EKS/EC2) while using their licenses locally.
* **Challenges:**
  + License management is up to the customer, and you need to ensure smooth integration with existing licenses.
* **Best For:**
  + Customers who want control over licensing while deploying and scaling on AWS infrastructure.

**6. On-Premises Delivery (Local UnityHub Deployment)**

* **What is it?**
  + In this model, you deliver UnityHub for local installation and usage while the data-processing backend (if needed) can be done locally or integrated with AWS services such as EKS via APIs or connectors.
* **Use Case:**
  + If your customers need to run the full Kapistio suite, including UnityHub, on their own hardware without cloud involvement (though cloud interaction can still happen via APIs or hybrid models).
* **Advantages:**
  + Full control for customers over their infrastructure.
  + Useful for customers who deal with sensitive data or need to comply with specific regulations that require local deployments.
* **Challenges:**
  + Customers must manage and maintain the entire infrastructure, which can limit scalability compared to cloud-based options.
* **Best For:**
  + Customers with strict security or compliance requirements who prefer local deployments but may occasionally use AWS resources for specific tasks.

**Delivery Options Summary:**

| **Delivery Method** | **UnityHub** | **AWS EKS** | **Best Use Case** |
| --- | --- | --- | --- |
| **SaaS** | Fully managed in the cloud | Fully managed in the cloud | Customers want a hands-off, cloud-based experience |
| **Hybrid** | Local deployment for user interaction | Cloud-based processing on EKS | Customers need local computing with cloud processing |
| **Container-Based (EKS)** | UnityHub containerized and deployed on EKS | Fully containerized in EKS | Customers want control over cloud deployment |
| **Amazon Machine Image** | Pre-configured UnityHub in an AMI for EC2 | Interacts with EKS for backend processing | Customers prefer EC2-based instances with custom setup |
| **BYOL** | Customer-provided UnityHub license | Deployable on EKS or EC2 | Customers already own UnityHub licenses |
| **On-Premises** | Fully local installation of UnityHub | Optional integration with AWS EKS via APIs | Customers need full control over infrastructure |

**Conclusion:**

The most suitable delivery method for Kapistio depends on your customers’ needs. If they need local use of UnityHub but benefit from cloud-based scaling and processing, a **Hybrid Delivery** model may be ideal. However, if you want to provide a fully cloud-based experience without customers managing infrastructure, **SaaS** is a great option.