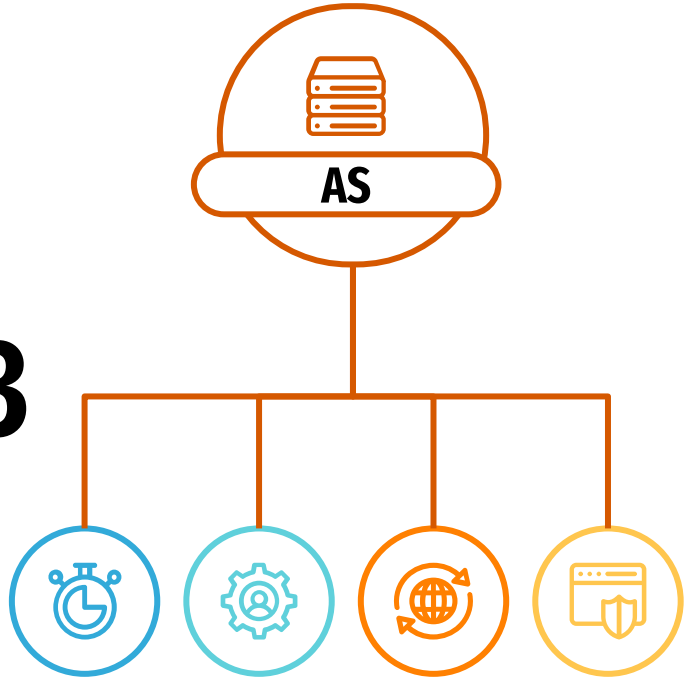
Estilos Arquitectonicos 3

MSc. Marco Tulio Aldana Prillwitz



FIUSAC
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Arquitectura de software

- La arquitectura de un sistema software es la definición de qué componentes constituyen ese sistema, sus responsabilidades y las relaciones de uso y dependencia entre ellos.

- Es, por tanto, completamente independiente de la tecnología que se utilice y no debería representar en ningún momento el *framework*, la base de datos o la forma de interactuar con el usuario.

Arquitectura Multi Tier

- Una arquitectura *multi tier* (o n-tier) hace referencia a una arquitectura en la que se expone la separación de sistema en varias capas físicas. Es decir, los distintos componentes están en máquinas separadas.

- En este tipo de arquitectura tenemos por ejemplo la de [cliente – servidor] o la de [presentación – negocio – datos]. Sin embargo, al referirse más a la distribución física del código ejecutable que a la relación lógica de sus componentes se aleja de lo que estamos tratando en aquí.

Arquitectura Multi Tier

Arquitectura Multitier (Distribuida)

- Interfase de usuario
- Administración de las transacciones

- Lógica del negocio
- Caché
- Administración de las transacciones
- Transparencia en la localización de los datos
- Balance de carga

- Administración de los datos

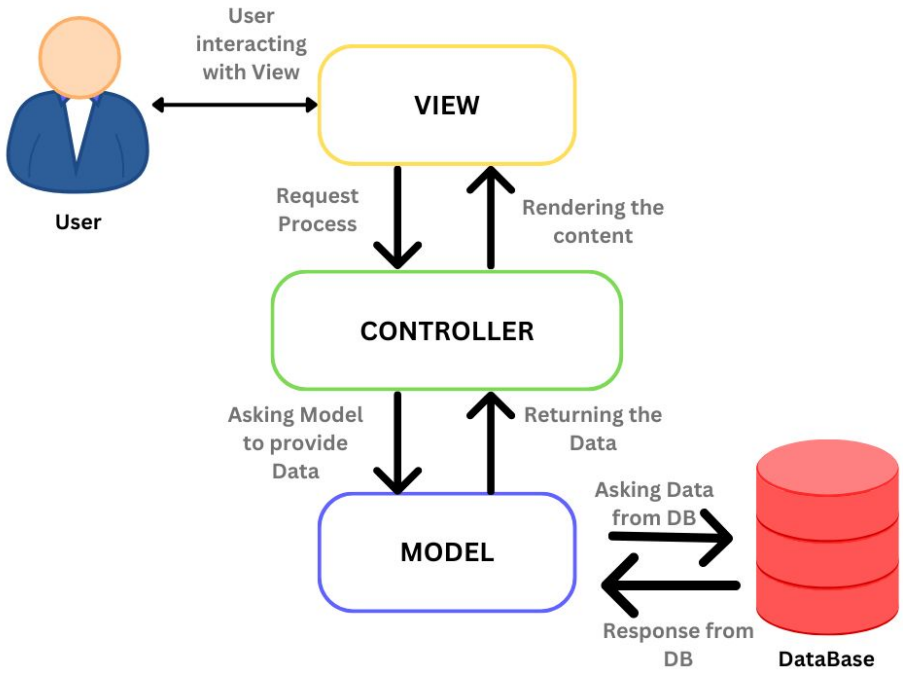
Ventajas de la Arquitectura Multi Tier

- Separación de responsabilidades : Las responsabilidades se separan por capa y son específicas
- Escalabilidad : Cada capa puede escalar de forma independiente
- Reusabilidad : Los componentes se puede reutilizar entre las diferentes capas o en ellas mismas
- Mantenimiento simplificado : Los cambios no afectan todas las capas a la misma vez

Desventajas de la Arquitectura Multi Tier

- Complejidad adicional por la separación de las capas
- Overhead de comunicación
- Requerimiento adicionales de recursos
- Latencia
- Dificultad en la depuración y monitoreo

Arquitectura MVC



Arquitectura Model Controller View

- Patrón diseño arquitectónico utilizado en el desarrollo de aplicaciones de software especialmente web.
- El objetivo principal de MVC es separar las preocupaciones de una aplicación en tres componentes principales : el modelo, la vista y el controlador.
- Cada uno de estos componentes tiene un rol específico y se encarga de tareas diferentes en la aplicación.

Arquitectura Model Controller View

- Modelo (Model)
    - Representa los datos de la aplicación y su lógica de negocio subyacente.
    - Se encarga de almacenar, manipular y procesar los datos.
    - No tiene conocimiento de la interfaz de usuario ni de cómo se muestran los datos.
    - Es independiente de la vista y el controlador.

Arquitectura Model Controller View

- Vista (View)
    - La vista es la capa de presentación de la aplicación.
    - Se encarga de mostrar los datos al usuario de una manera comprensible.
    - No contiene lógica de negocio, solo se encarga de la presentación.
    - Se comunica con el modelo para obtener los datos que se van a mostrar, pero no modifica directamente el modelo.
    - Puede haber múltiples vistas para un mismo modelo, mostrando los datos de diferentes maneras según las necesidades del usuario.
    - Vistas pueden ser páginas HTML, páginas de correo electrónico.

Arquitectura Model Controller View

- Controlador (Controller)
  - El controlador actúa como intermediario entre el modelo y la vista.
  - Se encarga de manejar las solicitudes del usuario, interpretarlas y tomar las acciones correspondientes.
  - Recibe las entradas del usuario a través de la vista y las envía al modelo para su procesamiento.
  - Decide qué vista debe ser mostrada al usuario en función de las acciones del usuario y del estado del modelo.
  - Es el componente que maneja la lógica de negocio de la aplicación.
  - El controlador coordina la interacción entre el modelo y la vista.

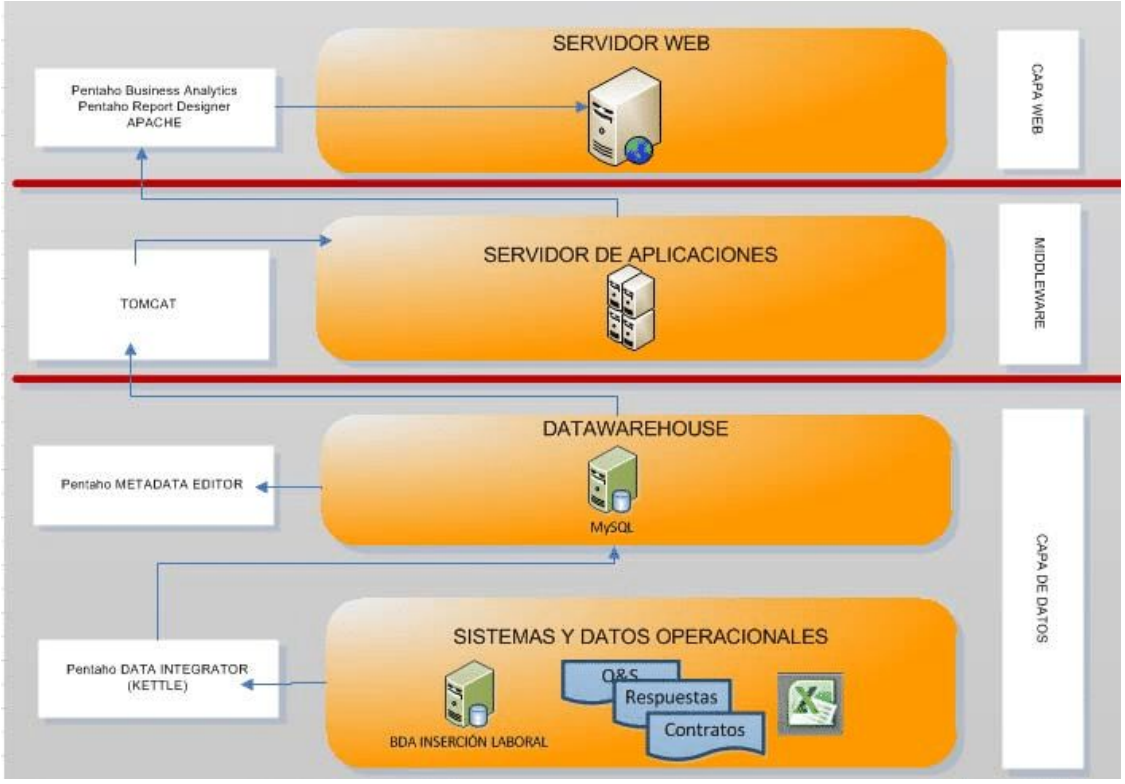
Arquitectura Multilayer

- Por su parte una arquitectura *multi layer* (o n-layer) refleja la separación lógica en capas de un sistema software. En este contexto una capa es simplemente un conjunto de clases, paquetes o subsistemas que tienen unas responsabilidades relacionadas dentro del funcionamiento del sistema.

- Estas capas están organizadas de forma jerárquica unas encima de otras y las dependencias siempre van hacia abajo. Es decir, que una capa concreta dependerá solamente de las capas inferiores, pero nunca de las superiores.

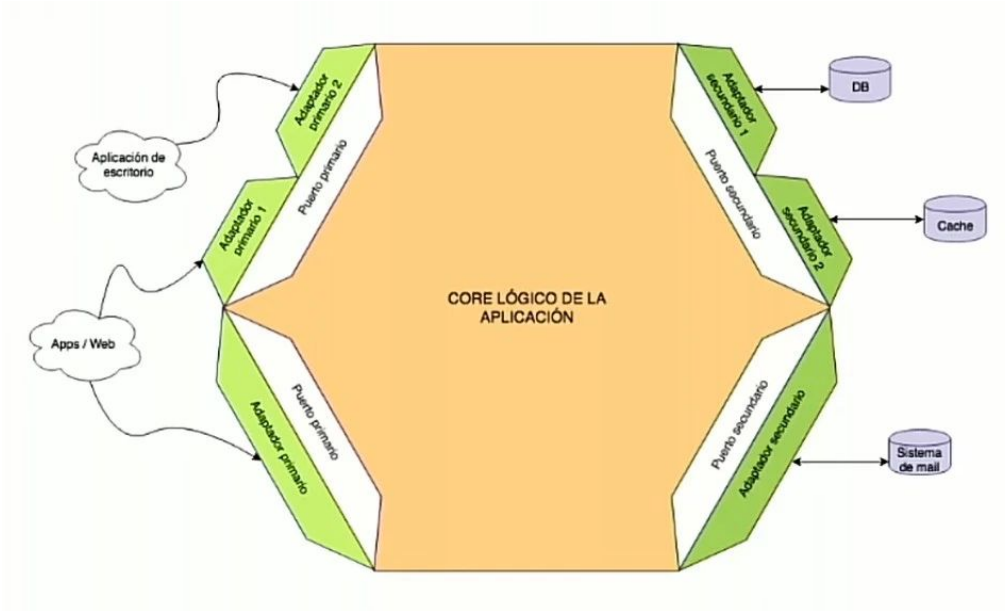
- En backend lo más común suele ser tener [servicio – negocio – acceso a datos], aunque a veces podríamos también encontrarnos una capa superior de presentación si esta se está manejando también a nivel de backend o una capa con controladores REST.

Arquitectura Multilayer



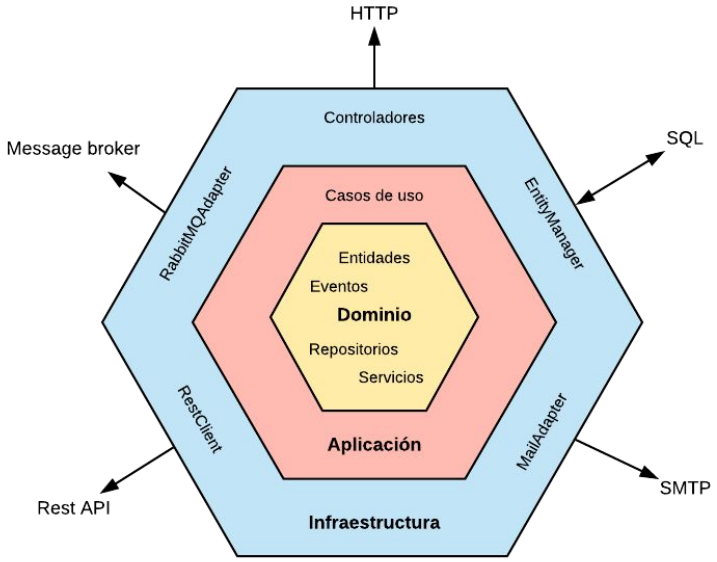
Arquitectura Hexagonal

- Es una **arquitectura** de software en la que se busca es separar el core lógico de la aplicación, dejarlo en el centro totalmente aislado del exterior del cliente y de otras interacciones. En la misma tenemos adaptadores, puertos, el core lógico y adaptadores y puertos secundarios.



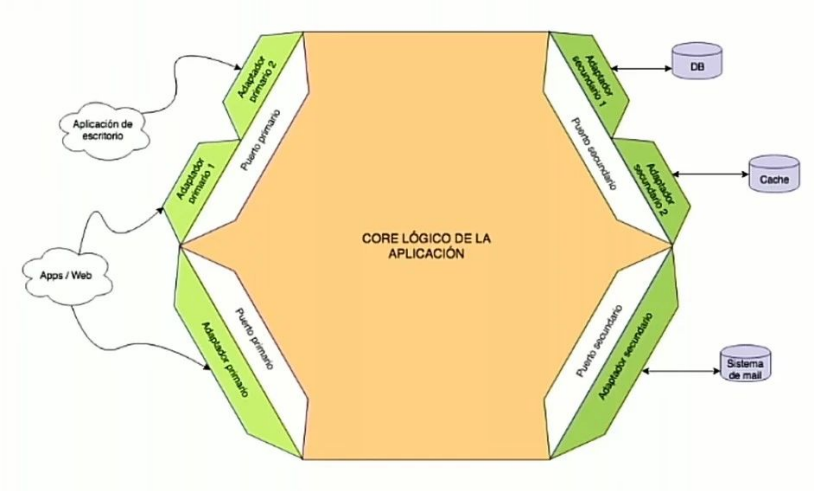
Arquitectura Hexagonal

- Propone que nuestro dominio sea el núcleo de las capas y que este no se acople a nada externo. En lugar de hacer uso explícito y mediante el principio de inversión de dependencias nos acoplados a contratos (interfaces o puertos) y no a implementaciones concretas.



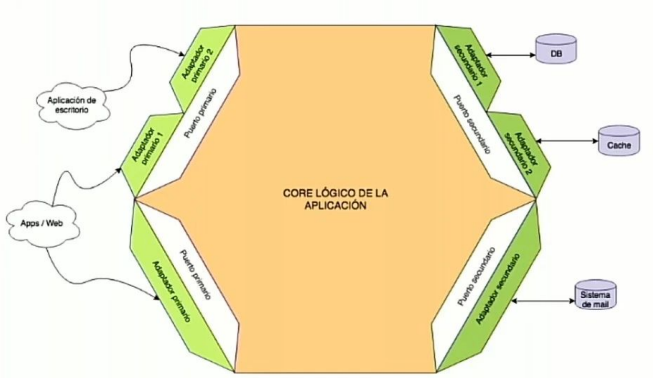
Arquitectura Hexagonal

- Adaptadores primarios : Son los controladores que se comunican con el cliente, con el exterior, y reciben las peticiones. Estos adaptadores usan y no implementan los puertos primarios para acceder al core lógico de la aplicación.
- Adaptadores secundarios : Son la implementación de los puertos secundarios que acceden a la base de datos, a bases de datos de caché y a otros micro servicios o sistemas en red.



Arquitectura Hexagonal

- Puertos primarios : Los puertos primarios serían la capa de servicio, la capa de lógica y negocio, donde haríamos toda nuestra infraestructura, en la que trabajaríamos con objeto de dominio.
- Puertos secundarios : Los puertos secundarios serían las interfaces a implementar por los adaptadores para conectarse de frente en base de datos.
- Lo que buscamos con todo esto es separar el core y los puertos de los adaptadores, así tenemos la aplicación encapsulada en el interior.

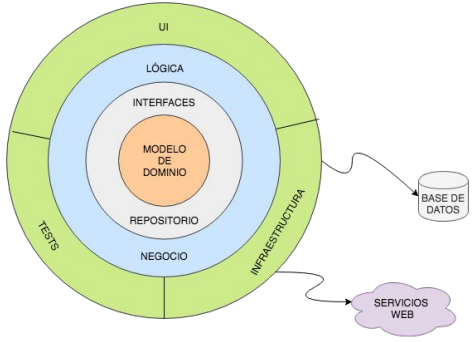


Arquitectura Onion

- Arquitectura multicapa construida por Jeffrey Palermo en torno a un modelo de dominio independiente de todo lo demás.
- Las dependencias van hacia el centro, por lo que todo depende de ese modelo de dominio. A su alrededor se organizan varias capas, estando en las más cercanas las interfaces de repositorio, es decir, las que definen el comportamiento del almacenamiento de los datos pero no lo implementan.
- En las capas siguientes está la lógica de negocio que usa estas interfaces y que en tiempo de ejecución tendrá las implementaciones apropiadas. Alrededor del núcleo de modelo puede haber un número variable de capas, pero siempre debe cumplirse que las interfaces estén más cerca que las clases que las utilizan. Con esto ya tenemos creado el *core* lógico de nuestra aplicación, que no tiene absolutamente ningún detalle de infraestructura.

Arquitectura Onion

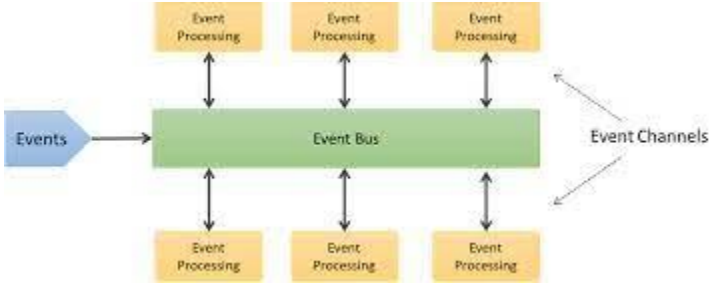
- Por último, en la capa más exterior es donde estarán todos los detalles de comunicación con el exterior (tanto de interfaz con el usuario como el almacenamiento) y los tests de integración.
- Las clases que se presentan aquí implementarán las interfaces que se definen en las capas inferiores, pudiendo cambiar por tanto las implementaciones dependientes de la tecnología sin que las capas inferiores se enteren.
- Lo que conseguimos de esta manera es una arquitectura que habla de cómo está montado el sistema y no de los terceros que se comunican con él.



Arquitectura por Eventos

- Es un modelo y una arquitectura de software que sirve para diseñar aplicaciones. En un sistema como este, la captura, la comunicación, el procesamiento y la permanencia de los eventos son la estructura central de la solución. Esto difiere del modelo tradicional basado en solicitudes.

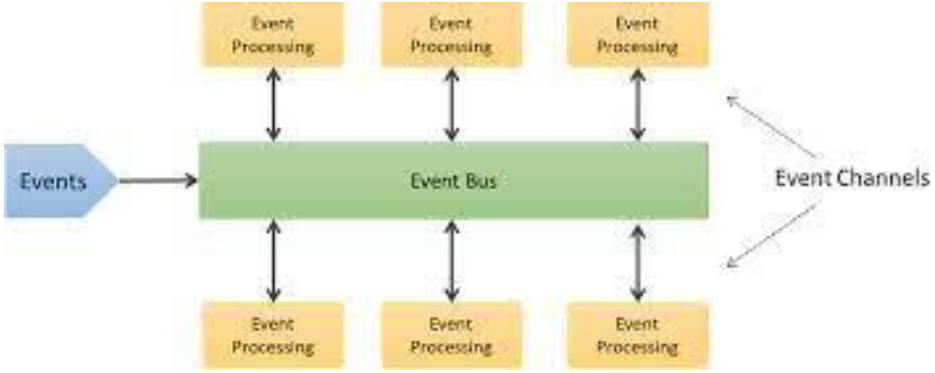
- Muchos diseños de aplicaciones modernas se basan en eventos, como los marcos de interacción con los clientes, que deben utilizar los datos de los clientes de forma inmediata. La arquitectura basada en eventos posibilita un acoplamiento mínimo, así que es una buena opción para las aplicaciones distribuidas y modernas.



Arquitectura por Eventos

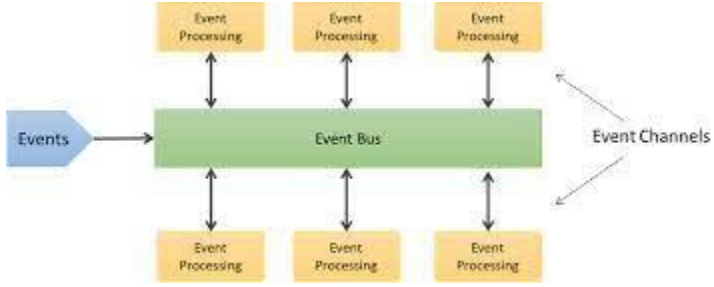
- Los eventos son aquellos sucesos o cambios significativos en el estado del hardware o el software de un sistema. Un evento y su notificación no son lo mismo: la segunda es un mensaje que el sistema envía para comunicar a otra parte del sistema que se produjo cierto evento.

- Los eventos pueden originarse por estímulos internos o externos. Pueden generarse con la actividad de un usuario, por ejemplo, cuando hace clic con el mouse o presiona una tecla; a partir de una fuente externa, como es el caso de un sensor; o provenir del sistema, cuando se carga un programa



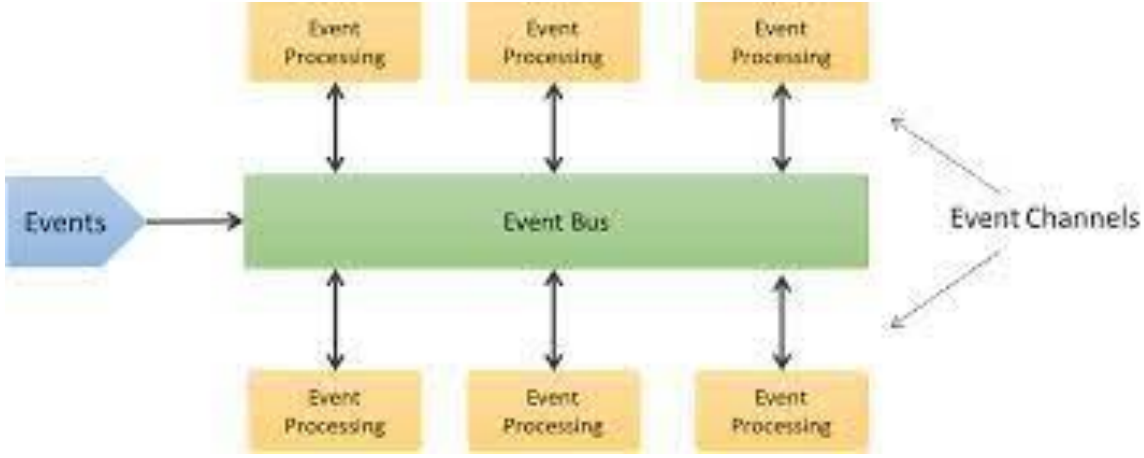
Arquitectura por Eventos

- Esta arquitectura está compuesta por productores y consumidores de eventos. El primero detecta los eventos y los representa como mensajes. No conoce al consumidor del evento ni el resultado que generará este último.
- Una vez que se detecta un evento, este se transmite del productor a los consumidores a través de canales de eventos, donde se procesa de manera asíncrona con una plataforma para este fin. Cuando se produce un evento, se debe informar a los consumidores, quienes podrían procesarlo o simplemente recibirlo.
- La plataforma de procesamiento ejecutará la respuesta adecuada para el evento y enviará la actividad a los consumidores correspondientes. Esta actividad downstream corresponde al lugar en el que se verá el resultado del evento.



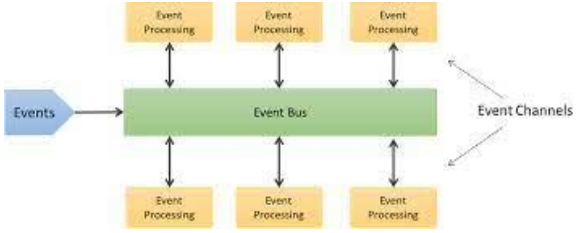
Arquitectura por Eventos

- **Modelo de publicación y suscripción**
  - Es una infraestructura de mensajería que se basa en suscripciones a un flujo de eventos. Con este modelo, una vez que se genera o publica un evento, este se envía a los suscriptores que necesitan estar informados al respecto.



Arquitectura por Eventos

- **Modelo de flujo de eventos**
  - Con este modelo, los eventos se escriben en un registro. Los consumidores no se suscriben a un flujo de eventos, sino que pueden leerlo desde cualquiera de sus partes y unirse a él en cualquier momento.
    - El **procesamiento de flujos de eventos** utiliza una plataforma de transmisión de datos, como Apache Kafka, para incorporar los eventos y procesar o transformar su flujo. Este procesamiento se puede utilizar para detectar patrones significativos en los flujos.
    - El **procesamiento de eventos simple** surge cuando un evento desencadena inmediatamente una acción en el consumidor.
    - El **procesamiento de eventos complejo** requiere que un consumidor de eventos procese una serie de ellos para detectar patrones.



Bibliografía

- "Software Architecture in Practice" (Arquitectura de Software en la Práctica), Len Bass, Paul Clements, Rick Kazman
- "Pattern-Oriented Software Architecture: A System of Patterns" (Arquitectura de Software Orientada a Patrones: Un Sistema de Patrones), Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal
- "Domain-Driven Design: Tackling Complexity in the Heart of Software" (Diseño Dirigido por Dominios: Abordando la Complejidad en el Corazón del Software), Eric Evans
- "Clean Architecture: A Craftsman's Guide to Software Structure and Design" (Arquitectura Limpia: Guía del Artesano para la Estructura y Diseño de Software), Robert C. Martin (Uncle Bob)
- "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions" (Patrones de Integración Empresarial: Diseño, Construcción e Implementación de Soluciones de Mensajería, Gregor Hohpe, Bobby Woolf
- "Microservices Patterns: With Examples in Java" (Patrones de Microservicios: Con Ejemplos en Java), Chris Richardson