

# Arquitecturas de Software

En este resumen enumeramos una variedad de Arquitecturas útiles y ampliamente utilizados. Este catálogo no pretende ser exhaustivo, de hecho, tal catálogo no sería muy extenso. Más bien está destinado a ser representativo. Mostramos arquitecturas de elementos de tiempo de ejecución (como intermediario o cliente-servidor) y de elementos de tiempo de diseño (como capas). Para cada arquitectura enumeramos el contexto, el problema y la solución. Como parte de la solución, describimos brevemente los elementos, las relaciones y las restricciones de cada arquitectura.

Para empezar, podemos definir la arquitectura de software como una ciencia que pretende estudiar cómo se definen organizan y unen los componentes de software. Como se comunican esas partes entre sí. Y las restricciones que regulan el funcionamiento del sistema.

Las arquitecturas se pueden clasificar por el tipo dominante de elementos que muestran: las arquitecturas de módulos muestran módulos, las arquitecturas de componentes y conectores (C&C) muestran componentes y conectores, y las arquitecturas de asignación muestran una combinación de elementos de software (módulos, componentes y conectores) y elementos no software.

Tabla: 1 Clasificación de Arquitecturas Según Bass

Módulos	<ul style="list-style-type: none"><li>● Arquitectura en capas</li></ul>
Componentes y conectores (C&C)	<ul style="list-style-type: none"><li>● Intermediario (<i>Broker</i>)</li><li>● Cliente - Servidor</li><li>● Modelo - Vista - Controlador (Model View Controller)</li><li>● Tuberías y Filtros (Pipes &amp; Filters)</li><li>● De igual a igual (Peer -To -Peer)</li><li>● SOAP</li><li>● Repositorio (Shared Data)</li><li>● Multi Nivel (Multi-tier Pattern)</li></ul>
Arquitecturas de asignación (Allocation)	<ul style="list-style-type: none"><li>● Multi Nivel (Multi-tier Architecture)</li><li>● Map-Reduce</li></ul>
Otros	<ul style="list-style-type: none"><li>● Publicador Suscriptor (publish-subscribe)</li><li>● Arquitectura basada en la Nube</li></ul>

## Arquitecturas basadas en Módulos

Tipo Módulo	Arquitectura en capas, tipo
Contexto	Todos los sistemas complejos experimentan la necesidad de desarrollar y evolucionar partes del sistema de manera independiente. Por esta razón, los desarrolladores del sistema necesitan una separación clara y bien documentada de las preocupaciones, para que los módulos del sistema puedan desarrollarse y mantenerse de manera independiente.
Problema	El software debe segmentarse de tal manera que los módulos se puede desarrollar y evolucionar por separado con poca interacción entre las partes, soportando portabilidad, modificabilidad y reutilización.
Solución	<p>Para lograr esta separación de preocupaciones, la arquitectura en capas divide el software en unidades llamadas capas. Cada capa es una agrupación de módulos que ofrece un conjunto cohesivo de servicios.</p> <p>Debe haber restricciones en la relación de uso permitido entre las capas: las relaciones deben ser unidireccionales.</p> <p>Las capas particionan completamente un conjunto de software, y cada partición se expone a través de una interfaz pública. Las capas se crean para interactuar de acuerdo con una estricta relación de orden. Si (A, B) está en esta relación, decimos que la implementación de la capa A puede usar cualquiera de las instalaciones públicas proporcionadas por la capa B. En algunos casos, los módulos en una capa pueden ser necesarios para usar directamente los módulos en una capa inferior no adyacente; normalmente solo se permiten los usos de la siguiente capa inferior. Este caso de software en una capa superior que utiliza módulos en una capa inferior no adyacente se denomina puente de capa. Si se producen muchos casos de puente de capa, es posible que el sistema no cumpla con sus objetivos de portabilidad y modificabilidad que las capas estrictas ayudan a lograr. Los usos ascendentes no están permitidos en esta arquitectura.</p> <p>Esta arquitectura define unas agrupaciones de módulos que ofrecen un conjunto cohesivo de servicios y una relación unidireccional de uso permitido (<i>allowed-to-use</i>) entre las capas. Se muestra gráficamente apilando cajas que representan capas una encima de la otra.</p>
Debilidades	<ul style="list-style-type: none"><li>• La adición de capas agrega un costo y complejidad iniciales a un sistema.</li><li>• Las capas contribuyen a una penalización de rendimiento.</li></ul>

Gráficamente	<table><tr><td>Capa de presentación</td></tr><tr><td>Capa de lógica de negocios</td></tr><tr><td>Capa de persistencia de datos</td></tr><tr><td>Capa de base de datos.</td></tr></table>	Capa de presentación	Capa de lógica de negocios	Capa de persistencia de datos	Capa de base de datos.
Capa de presentación					
Capa de lógica de negocios					
Capa de persistencia de datos					
Capa de base de datos.					
Restricciones	Cada pieza de software se asigna a exactamente una capa. Debe haber al menos dos capas (pero generalmente hay tres o Más). Las relaciones de uso permitido no deben ser circulares (es decir, una la capa no puede usar una capa arriba).				
Ejemplos	Sistemas de comunicación basados en el modelo ISO/OSI. Sistema Operativo <b>THE</b> , construido en Technische Hogeschool Eindhoven en Holanda por E. W. Dijkstra (1968) y sus estudiantes				

## Arquitecturas basadas en Componentes y conectores (C&C)

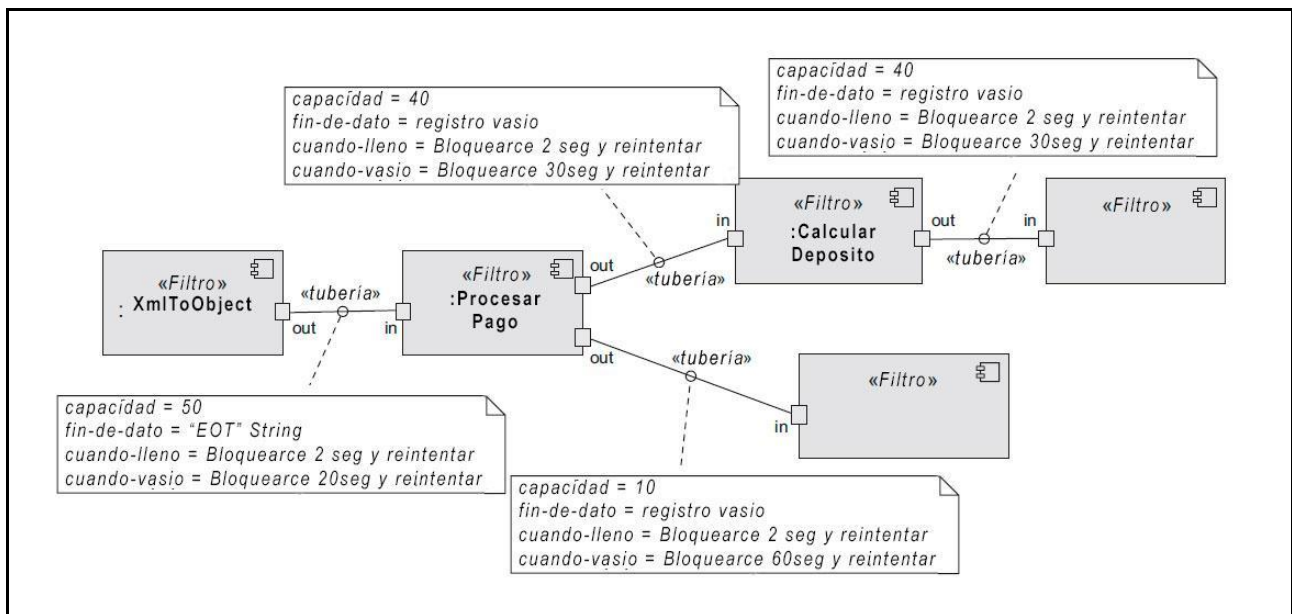
Tipo C&C	Intermediario (Broker)
Contexto	Muchos sistemas se construyen a partir de una colección de servicios distribuidos en múltiples servidores. La implementación de estos sistemas es compleja porque debemos preocuparnos por la forma en que los sistemas interactúan, cómo se conectan entre sí y cómo intercambian información y por cómo será la disponibilidad de los servicios que brinda.
Problema	¿Cómo estructuramos el software distribuido para que los usuarios del servicio no necesiten conocer la naturaleza y la ubicación de los proveedores de servicios, lo que facilita el cambio dinámico de los enlaces entre los usuarios y los proveedores?
Solución	<p>La arquitectura de intermediario separa a los usuarios de servicios (clientes) de los proveedores de servicios (servidores) mediante la inserción de un intermediario. Cuando un cliente necesita un servicio, consulta a un intermediario a través de una interfaz de servicio. El agente luego reenvía la solicitud de servicio del cliente a un servidor, que procesa la solicitud. El resultado del servicio se comunica desde el servidor al agente, que luego devuelve el resultado (y cualquier excepción) al cliente solicitante. De esta manera, el cliente puede ignorar la identidad, la ubicación y las características del servidor.</p> <p>Debido a esta separación, si un servidor deja de estar disponible, el agente puede elegir dinámicamente un reemplazo. Si un servidor se reemplaza con un servicio diferente y compatible, nuevamente, el intermediario es el único componente que necesita conocer este cambio, por lo que el cliente no se ve afectado.</p>

Debilidades	<ul style="list-style-type: none"> <li>• Agregan complejidad los intermediarios y posiblemente los proxies deben diseñarse e implementarse, junto con los protocolos de mensajería.</li> <li>• Agregan un nivel de direccionamiento indirecto entre un cliente y un servidor, lo que agrega latencia a su comunicación.</li> <li>• La depuración de los corredores de comunicación puede ser difícil porque están involucrados en entornos altamente dinámicos donde las condiciones que conducen a una falla pueden ser difíciles de replicar.</li> <li>• El intermediario sería un punto de ataque obvio, desde una perspectiva de seguridad.</li> <li>• Un agente mal diseñado, puede ser un punto de falla único para un sistema grande y complejo.</li> <li>• Los corredores pueden potencialmente ser cuellos de botella para la comunicación</li> </ul>
Elementos	<p><b>Cliente:</b> que requiere un servicio</p> <p><b>Servidor:</b> que provee un servicio</p> <p><b>Intermediario:</b> que localiza un servidor apropiado para cumplir con la solicitud de un cliente, reenvía la solicitud al servidor y devuelve los resultados al cliente</p>
Gráficamente	<pre> graph LR     subgraph Cliente         direction TB         C1[Smartphone]         C2[PC]     end     subgraph Intermediario         direction TB         I1[Server]     end     subgraph Servidor         direction TB         S1[Cloud]     end     Cliente --&gt; Intermediario     Intermediario -.-&gt; Servidor </pre>
Ejemplo	Los sistemas Proxy

Tipo C&C	Modelo Vista Controlador
Contexto	El software de interfaz de usuario suele ser la parte más frecuentemente modificada de una aplicación interactiva. Por este motivo, es importante mantener las modificaciones del software de la interfaz de usuario separadas del resto del sistema.
Problema	¿Cómo se puede mantener la funcionalidad de la interfaz de usuario separada de la funcionalidad de la aplicación y aun así responder a las aportaciones de los usuarios o a los cambios en los datos de la aplicación subyacente? ¿Y cómo se pueden crear, mantener y coordinar múltiples vistas de la interfaz de usuario cuando cambian los datos de la aplicación subyacente?
Solución	<p>La arquitectura modelo-vista-controlador (MVC) separa la funcionalidad de la aplicación en tres tipos de componentes:</p> <ul style="list-style-type: none"> <li>• Un modelo, que contiene los datos de la aplicación.</li> <li>• Una vista, que muestra parte de los datos subyacentes e interactúa con el usuario.</li> </ul>

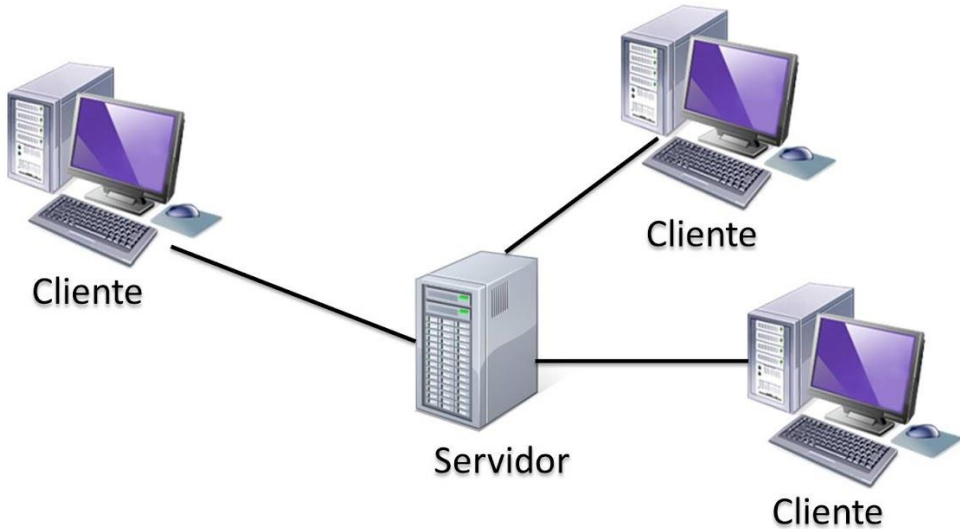
	<ul style="list-style-type: none"> <li>• Un controlador, que media entre el modelo y la vista y administra las notificaciones de cambios de estado.</li> </ul> <p>MVC no es apropiado para todas las situaciones. El diseño y la implementación de tres tipos distintos de componentes, junto con sus diversas formas de interacción, pueden ser costosos y este costo puede no tener sentido para interfaces de usuario relativamente simples. Además, la coincidencia entre las abstracciones de MVC y los kits de herramientas de la interfaz de usuario comercial no es perfecta. La vista y el controlador dividen la entrada y la salida, pero la separación de estas funciones en la implementación a menudo no es tan clara y esto puede resultar en una discrepancia conceptual entre la arquitectura y el kit de herramientas de la interfaz de usuario.</p>
Debilidades	<p>La complejidad puede no valer la pena para interfaces de usuario simples. Es posible que las abstracciones del modelo, la vista y el controlador no sean adecuados para algunos kits de herramientas de interfaz de usuario.</p>
Elementos	<ul style="list-style-type: none"> <li>• El <b>modelo</b> es una representación de los datos o el estado de la aplicación, y contiene (o proporciona una interfaz para) la lógica de la aplicación.</li> <li>• La <b>vista</b> es un componente de la interfaz de usuario que produce una representación del modelo para el usuario o permite alguna forma de entrada del usuario, o ambos.</li> <li>• El <b>controlador</b> gestiona la interacción entre el modelo y la vista, traduciendo las acciones del usuario en cambios al modelo o cambios en la vista.</li> </ul>
Gráficamente	<pre> graph TD     Vista[Vista] --&gt; Controlador[Controlador]     Controlador --&gt; Modelo[Modelo]     Modelo --&gt; Vista     Vista -.-&gt; Controlador     Modelo -.-&gt; Controlador </pre>
Ejemplos	<p>La arquitectura MVC se usa ampliamente en bibliotecas de interfaz de usuario como las clases:</p> <ul style="list-style-type: none"> <li>• Java's Swing classes.</li> <li>• Microsoft's ASP.NET framework.</li> <li>• Adobe's Flex software development kit.</li> <li>• Nokia's Qt framework.</li> </ul> <p>Por lo tanto, es común que una sola aplicación contenga muchas instancias de <b>MVC</b> (a menudo una por objeto de interfaz de usuario).</p>

Tipo C&C	Tubería y Filtro
Contexto	Se requieren muchos sistemas para transformar flujos de elementos de datos discretos, desde la entrada hasta la salida. Muchos tipos de transformaciones ocurren repetidamente en la práctica, por lo que es deseable crearlas como partes independientes y reutilizables.
Problema	<p>Dichos sistemas deben dividirse en componentes reutilizables y poco acoplados con mecanismos de interacción simples y genéricos. De esta manera se pueden combinar de forma flexible entre sí.</p> <p>Los componentes, que son genéricos y están ligeramente acoplados, se pueden reutilizar fácilmente y siendo éstos independientes, pueden ejecutarse en paralelo.</p>
Solución	<p>La arquitectura de interacción en la arquitectura de tubería y filtro se caracteriza por transformaciones sucesivas de flujos de datos. Los datos llegan a los puertos de entrada de un filtro, se transforman y luego se pasan a través de su (s) puerto (s) de salida a través de una tubería al siguiente filtro.</p> <p><b>Un solo filtro puede:</b> consumir datos de, o producir datos a, uno o más puertos.</p>
Debilidades	<ul style="list-style-type: none"> <li>• Esta arquitectura generalmente no es una buena opción para un sistema interactivo, ya que no permite los ciclos (que son importantes para la retroalimentación del usuario).</li> <li>• Tener una gran cantidad de filtros independientes puede agregar cantidades sustanciales de sobrecarga computacional, porque cada filtro se ejecuta como su propio hilo o proceso.</li> <li>• Los sistemas de tubería y filtro pueden no ser apropiados para cálculos de larga ejecución, sin la adición de algún tipo de funcionalidad de punto de control / restauración, ya que la falla de cualquier filtro (o tubería) puede causar que toda la tubería falle.</li> </ul>
Elementos	<p><b>Filtro</b>, que es un componente que transforma los datos leídos en su (s) puerto (s) de entrada en datos escritos en su (s) puerto (s) de salida. Los filtros pueden ejecutarse simultáneamente entre sí. Los filtros pueden transformar incrementalmente los datos; es decir, pueden comenzar a producir una salida tan pronto como comiencen a procesar la entrada. Las características importantes incluyen tasas de procesamiento, formatos de datos de entrada / salida y la transformación ejecutada por el filtro.</p> <p><b>Tubería</b>, que es un conector que transporta datos desde los puertos de salida de un filtro a los puertos de entrada de otro filtro. Una tubería tiene una única fuente para su entrada y un solo objetivo para su salida. Una tubería conserva la secuencia de elementos de datos y no altera los datos que pasan por ella. Las características importantes incluyen el tamaño del búfer, el protocolo de interacción, la velocidad de transmisión y el formato de los datos que pasan a través de una tubería.</p>
Gráficamente	



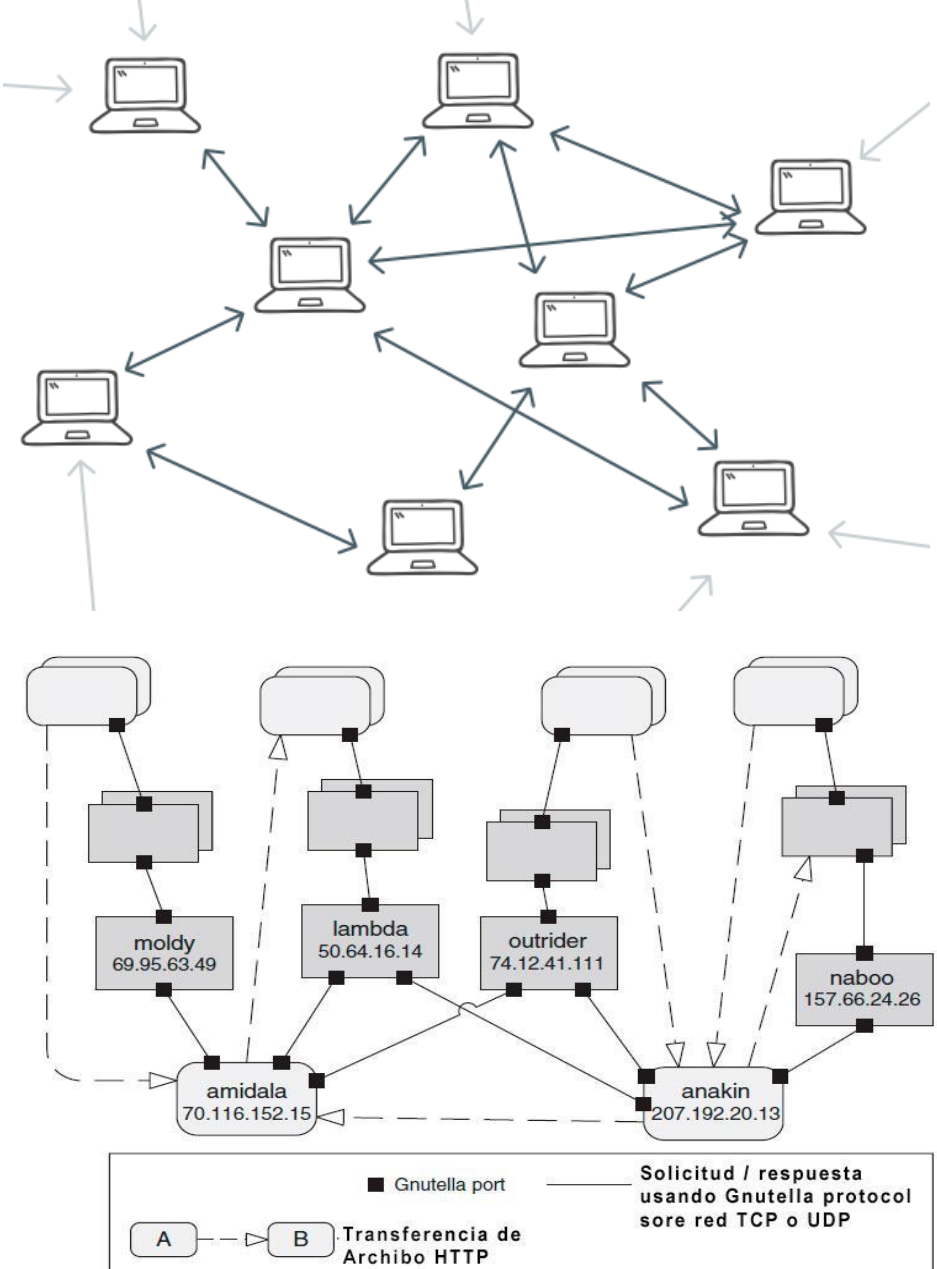
Ejemplos	<ul style="list-style-type: none"> <li>• Los sistemas que implementan la arquitectura de procesamiento de solicitudes del servidor web Apache.</li> <li>• Sistemas que utilizan la arquitectura map-reduce.</li> <li>• Yahoo! utiliza tuberías y filtros para analizar fuentes RSS.</li> </ul>
----------	--

Tipo C&C	Cliente Servidor
Contexto	Hay recursos y servicios compartidos que una gran cantidad de clientes distribuidos desean acceder, y para los cuales queremos controlar el acceso o la calidad del servicio.
Problema	Al administrar un conjunto de recursos y servicios compartidos, podemos promover la modificabilidad y la reutilización, eliminando los servicios comunes y teniendo que modificarlos en una única ubicación o en un pequeño número de ubicaciones. Se quiere mejorar la escalabilidad y la disponibilidad mediante la centralización del control de estos recursos y servicios, al tiempo que distribuimos los recursos en varios servidores físicos.
Solución	Los clientes interactúan solicitando servicios a el/los de servidor(es), que proporcionan un conjunto de servicios. Algunos componentes pueden actuar como clientes y servidores. Puede haber un servidor central o varios distribuidos. El tipo de conector principal para la arquitectura cliente-servidor es un conector de datos controlado por un protocolo de solicitud / respuesta ( <i>request/reply</i> ) utilizado para invocar servicios.
Debilidades	<p>Algunas de las desventajas de la arquitectura cliente-servidor son:</p> <ul style="list-style-type: none"> <li>• El servidor puede ser un cuello de botella en el rendimiento</li> <li>• Puede ser un punto único de falla.</li> <li>• Además, las decisiones sobre dónde ubicar la funcionalidad a menudo son complejas y costosas de cambiar después de que el sistema se haya puesto en operación.</li> </ul>
Elementos	<p><b>Cliente</b>, un componente que invoca servicios de un componente de servidor. Los clientes tienen puertos que describen los servicios que requieren. Servidor, un componente que proporciona servicios a los clientes.</p> <p>Los <b>servidores</b> tienen puertos que describen los servicios que proporcionan.</p>

	<p>Las características importantes incluyen información sobre la naturaleza de los puertos del servidor (como la cantidad de clientes que pueden conectarse) y las características de rendimiento (por ejemplo, las tasas máximas de invocación de servicio). Conector de solicitud / respuesta, un conector de datos que emplea un protocolo de solicitud / respuesta, utilizado por un cliente para invocar servicios en un servidor. Las características importantes incluyen si las llamadas son locales o remotas, y si los datos están encriptados.</p>
Gráficamente	 <p>El diagrama ilustra la arquitectura cliente-servidor. En el centro se encuentra un icono de un servidor, etiquetado como 'Servidor'. Alrededor del servidor, hay tres iconos de clientes (cada uno con una torre, un monitor, un teclado y un mouse), etiquetados como 'Cliente'. Líneas rectas representan las conexiones de red entre el servidor y cada uno de los tres clientes.</p>
Ejemplo	<p>La World Wide Web es el ejemplo más conocido de un sistema que se basa en la arquitectura cliente-servidor, permitiendo a los clientes (navegadores web) acceder a la información de los servidores a través de Internet mediante el Protocolo de transferencia de hipertexto (HTTP).</p> <p>HTTP es un protocolo de solicitud / respuesta. HTTP es sin estado; la conexión entre el cliente y el servidor finaliza después de cada respuesta del servidor.</p>



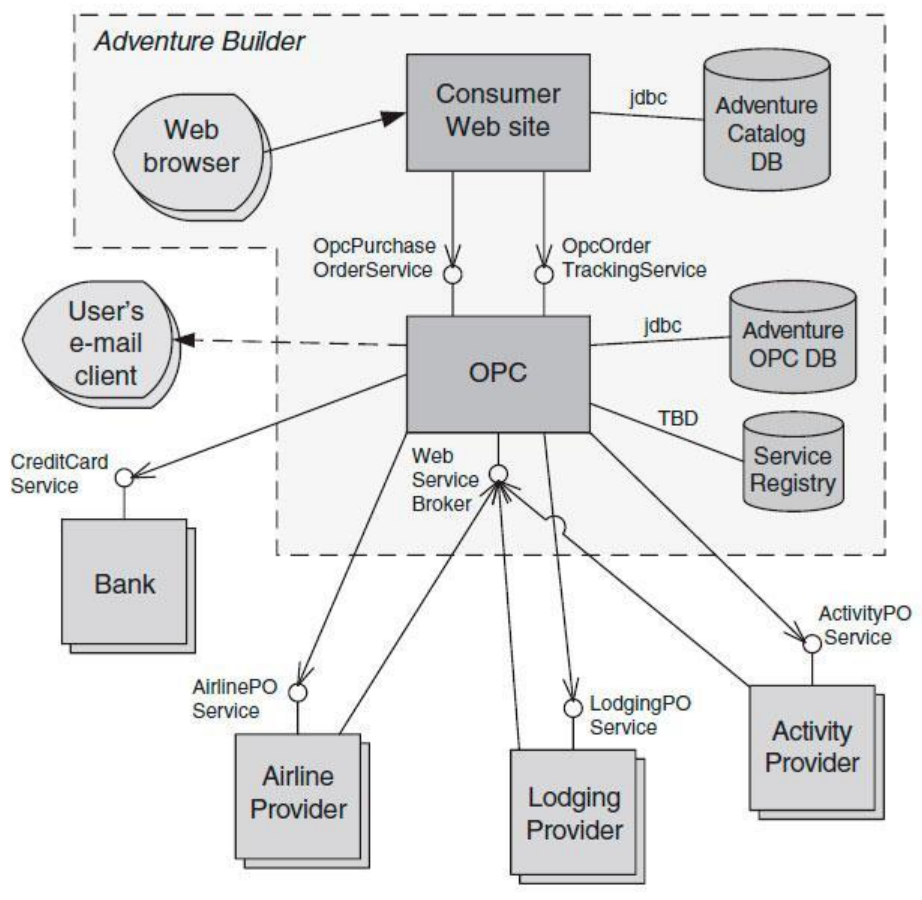
Tipo C&C	De igual a Igual (Peer-to-Peer   P2P)
Contexto	Las entidades computacionales distribuidas, cada una de las cuales se considera igual de importante en términos de iniciar una interacción y cada una de las cuales proporciona sus propios recursos, necesitan cooperar y colaborar para brindar un servicio a una comunidad distribuida de usuarios.
Problema	¿Cómo se puede conectar un conjunto de entidades computacionales distribuidas "iguales" a través de un protocolo común para que puedan organizar y compartir sus servicios con alta disponibilidad y escalabilidad?
Solución	<p>En la arquitectura de igual a igual (P2P), los componentes interactúan directamente como iguales. Todos los pares son "iguales" y ningún par o grupo de compañeros puede ser crítico para la salud del sistema. La comunicación de igual a igual suele ser una interacción de solicitud / respuesta sin la asimetría que se encuentra en la arquitectura cliente-servidor. Es decir, cualquier componente puede, en principio, interactuar con cualquier otro componente solicitando sus servicios. La interacción puede ser iniciada por cualquiera de las partes, es decir, en términos cliente-servidor, cada componente del par es un cliente y un servidor.</p> <p>A veces, la interacción es solo para reenviar datos sin la necesidad de una respuesta. Cada par proporciona y consume servicios similares y utiliza el mismo protocolo. Los conectores en los sistemas peer-to-peer implican interacciones bidireccionales, lo que refleja la comunicación bidireccional que puede existir entre dos o más componentes peer-to-peer.</p>
Nota	<p>Los pares primero se conectan a la red de igual a igual en la que descubren a otros pares con los que pueden interactuar, y luego inician acciones para lograr su cálculo cooperando con otros pares al solicitar servicios.</p> <p>Una arquitectura peer-to-peer puede tener nodos peer especializados (llamados supernodos) que tienen capacidades de indexación o enrutamiento y permiten que la búsqueda de un par regular llegue a un mayor número de peers.</p> <p>Normalmente, varios pares tienen capacidades superpuestas, como proporcionar acceso a los mismos datos o proporcionar servicios equivalentes. Por lo tanto, un par que actúa como cliente puede colaborar con varios peers que actúan como servidores para completar una tarea determinada.</p>
Elementos	<p><b>Peer</b>, que es un componente independiente que se ejecuta en un nodo de red. Los componentes pares especiales pueden proporcionar enrutamiento, indexación y capacidad de búsqueda de pares.</p> <p>El conector de solicitud / respuesta, que se utiliza para conectarse a la red de pares, buscar otros pares e invocar servicios de otros pares. En algunos casos, la necesidad de una respuesta es eliminada.</p>
Debilidades	<p>La gestión de la seguridad, la coherencia de los datos, la disponibilidad de datos / servicios, la copia de seguridad y la recuperación son todas más complejas. Es posible que los sistemas peer-to-peer pequeños no puedan lograr constantemente objetivos de calidad, como el rendimiento y la disponibilidad.</p>

Gráficamente	 <p>The diagram is divided into two parts. The top part shows a general peer-to-peer network topology with several laptop icons connected by bidirectional arrows, representing a decentralized network. The bottom part shows a specific Gnutella network structure. It features several nodes, each represented by a stack of documents and a box with a name and IP address: moldy (69.95.63.49), lambda (50.64.16.14), outrider (74.12.41.111), naboo (157.66.24.26), amidala (70.116.152.15), and anakin (207.192.20.13). These nodes are connected by solid lines, indicating Gnutella protocol connections. Dashed lines represent HTTP file transfers between nodes. A legend at the bottom explains the symbols: a black square represents a Gnutella port, a solid line represents a request/response using the Gnutella protocol over TCP or UDP, and a dashed line represents an HTTP file transfer between nodes A and B.</p>
<a href="#">Ejemplo</a>	<a href="http://www.bittorrent.org/bittorrentecon.pdf">BitTorrent</a> ( <a href="http://www.bittorrent.org/bittorrentecon.pdf">http://www.bittorrent.org/bittorrentecon.pdf</a> ) <a href="http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html">Gnutella</a> ( <a href="http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html">http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html</a> ) <a href="https://mice.cs.columbia.edu/getTechreport.php?techreportID=99&amp;format=pdf&amp;">Skype</a> ( <a href="https://mice.cs.columbia.edu/getTechreport.php?techreportID=99&amp;format=pdf&amp;">https://mice.cs.columbia.edu/getTechreport.php?techreportID=99&amp;format=pdf&amp;</a> )

Tipo C&C	Repositorio (Shared-Data)
Contexto	Varios componentes computacionales necesitan compartir y manipular grandes cantidades de datos. Estos datos no pertenecen únicamente a ninguno de esos componentes.
Problema	¿Cómo pueden los sistemas almacenar y manipular datos persistentes a los que acceden múltiples componentes independientes?
Solución	En la arquitectura de datos compartidos, la interacción está dominada por el intercambio de datos persistentes entre múltiples <i>accesores</i> de datos y al menos un almacén de datos compartidos. El intercambio puede ser iniciado por los

	<p><i>accesores</i> o el almacén de datos. El tipo de conector es lectura y escritura de datos. El modelo computacional general asociado con los sistemas de datos compartidos es que los <i>accesores</i> de datos realizan operaciones que requieren datos del almacén de datos y escriben los resultados en uno o más almacenes de datos. Esos datos pueden ser vistos y activados por otros usuarios que acceden a ellos. En un sistema de datos compartidos puros, los <i>accesores</i> de datos interactúan sólo a través de uno o más almacenes de datos compartidos.</p>
Elementos	<p><b>Almacén de datos compartidos.</b> Las preocupaciones incluyen los tipos de datos almacenados, las propiedades orientadas al rendimiento de los datos, la distribución de datos y la cantidad de <i>accesores</i> permitido.</p> <p><b>Componente de acceso a los datos.</b> Conector de lectura y escritura de datos. Una opción importante aquí es si el conector es transaccional o no, así como el idioma de lectura / escritura, los protocolos y la semántica.</p>
Debilidades	<ul style="list-style-type: none"> <li>• El almacén de datos compartidos puede ser un cuello de botella de rendimiento.</li> <li>• El almacén de datos compartidos puede ser un único punto de error.</li> <li>• Los productores y consumidores de datos pueden estar estrechamente acoplados.</li> </ul>
<p>Gráficamente</p> <p><b>Key:</b></p> <ul style="list-style-type: none"> <li>Windows GUI app (rectángulo)</li> <li>Programas (rectángulo)</li> <li>Web Apps (óvalo)</li> <li>Repositorio de datos (cilindro)</li> <li>Lectura (cilindro → rectángulo/óvalo)</li> <li>Escritura (rectángulo/óvalo → cilindro)</li> <li>Lectura Escritura (cilindro ↔ rectángulo/óvalo)</li> </ul>	
Ejemplos	<p>SVN (Subversion).</p> <p>Git Hub, GitLab,</p>

Tipo C&C	<b>Arquitectura Orientada a servicios SOAP</b>
Contexto	Los proveedores de servicios ofrecen (y describen) una cantidad de servicios y los consumidores los consumen. Los consumidores de servicios deben poder entender y utilizar estos servicios sin ningún conocimiento detallado de su implementación.
Problema	¿Cómo podemos apoyar la interoperabilidad de los componentes distribuidos que se ejecutan en diferentes plataformas y están escritos en diferentes lenguajes de implementación, proporcionados por diferentes organizaciones y distribuidos a través de Internet? ¿Cómo podemos ubicar los servicios y combinarlos (y combinarlos dinámicamente) en coaliciones significativas y al mismo tiempo lograr un rendimiento, seguridad y disponibilidad razonables?
Solución	<p>La arquitectura orientada a servicios (SOA) describe una colección de componentes distribuidos que proporcionan y / o consumen servicios.</p> <p>En una SOA, los componentes del proveedor de servicios y los componentes del consumidor de servicios pueden usar diferentes lenguajes y plataformas de implementación. Los servicios son en gran medida independientes: los proveedores de servicios y los consumidores de servicios generalmente se implementan de forma independiente y, a menudo, pertenecen a diferentes sistemas o incluso a diferentes organizaciones. Los componentes tienen interfaces que describen los servicios que solicitan de otros componentes y los servicios que proporcionan. Los atributos de calidad de un servicio se pueden especificar y garantizar con un acuerdo de nivel de servicio (SLA). En algunos casos, estos son legalmente vinculantes. Los componentes logran su cálculo solicitando servicios entre sí.</p> <p>Los tipos básicos de conectores utilizados en SOA son estos:</p> <ul style="list-style-type: none"> <li>• <b>SOAP</b> El protocolo estándar para la comunicación en la tecnología de servicios web. Los consumidores y proveedores de servicios interactúan intercambiando mensajes XML de solicitud / respuesta, generalmente sobre HTTP.</li> <li>• Transferencia de Estado Representacional (<b>REST</b>). Un consumidor de servicios envía solicitudes HTTP sin bloqueo. Estas solicitudes se basan en los cuatro comandos HTTP básicos (POST, GET, PUT, DELETE) para decirle al proveedor del servicio que cree, recupere, actualice o elimine un recurso.</li> <li>• <b>Mensajes asíncronos</b>, un intercambio de información de "disparar y olvidar" ("fire-and-forget"). Los participantes no tienen que esperar un acuse de recibo, ya que se supone que la infraestructura ha entregado el mensaje correctamente. El conector de mensajería puede ser punto a punto o publicación-suscripción.</li> </ul>
Elementos	<p>Componentes:</p> <ul style="list-style-type: none"> <li>• Service providers, which provide one or more services through published interfaces. Concerns are often tied to the chosen implementation technology, and include performance, authorization constraints, availability, and cost. In some cases these properties are specified in a service-level agreement.</li> <li>• Consumidores de servicios, que invocan servicios directamente oa</li> </ul>

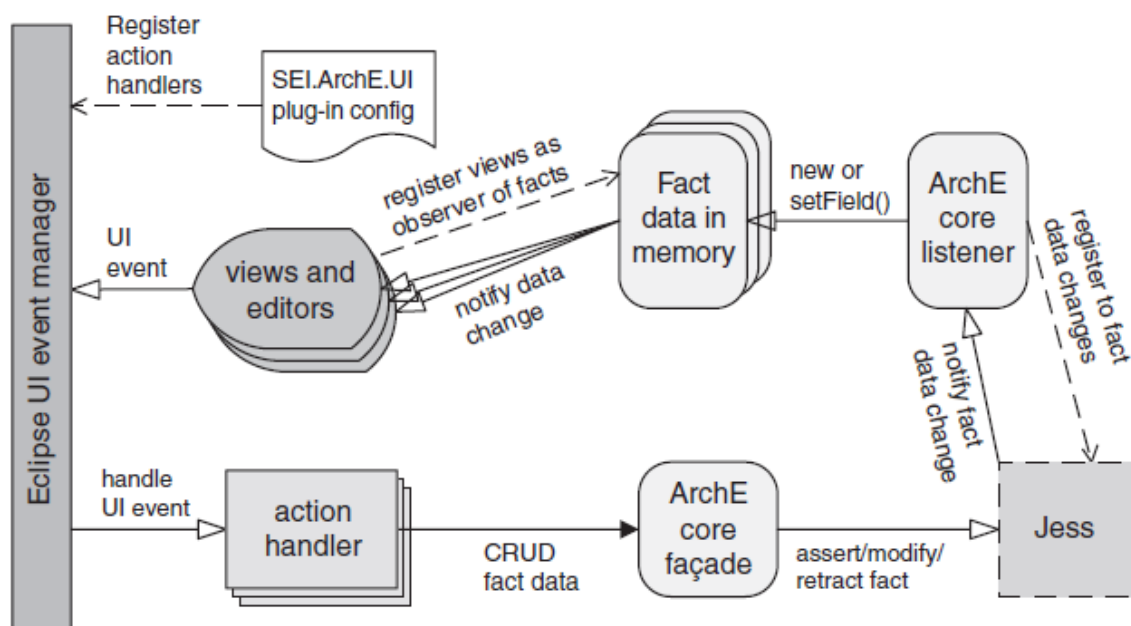
	<p>través de un intermediario.</p> <ul style="list-style-type: none"> <li>• Los proveedores de servicios también pueden ser consumidores de servicios.</li> <li>• ESB, que es un elemento intermediario que puede enrutar y transformar mensajes entre proveedores de servicios y consumidores.</li> <li>• Registro de servicios, que puede ser utilizado por los proveedores para registrar sus servicios y por los consumidores para descubrir servicios en tiempo de ejecución.</li> </ul> <p>Conectores:</p> <ul style="list-style-type: none"> <li>• Conector SOAP, que utiliza el protocolo SOAP para la comunicación sincrónica entre servicios web, generalmente a través de HTTP.</li> <li>• Conector REST, que se basa en las operaciones básicas de solicitud / respuesta del protocolo HTTP.</li> <li>• Conector de mensajería asíncrono, que utiliza un sistema de mensajería para ofrecer intercambios de mensajes asíncronos de punto a punto o de publicación / suscripción.</li> </ul>
Debilidades	<p>Los sistemas basados en SOA son típicamente complejos de construir. No controlas la evolución de los servicios independientes. Hay una sobrecarga de rendimiento asociada con el middleware, y los servicios pueden ser cuellos de botella de rendimiento y, por lo general, no ofrecen garantías de rendimiento.</p>
Gráficamente	 <p>El diagrama ilustra la arquitectura de un sistema basado en SOA (Service-Oriented Architecture) para un ejemplo llamado "Adventure Builder".</p> <ul style="list-style-type: none"> <li><b>Adventure Builder (Contenedor):</b> Encierra los componentes principales:       <ul style="list-style-type: none"> <li><b>Consumer Web site:</b> Interfaz de usuario que se conecta a la <b>Adventure Catalog DB</b> (base de datos) a través de JDBC.</li> <li><b>OPC (Orquestador de Procesos):</b> El núcleo central que coordina los servicios. Se conecta a la <b>Adventure OPC DB</b> (base de datos) y al <b>Service Registry</b> (registro de servicios) a través de JDBC y TBD (To Be Determined).</li> <li><b>Web Service Broker:</b> Actúa como intermediario entre el OPC y los proveedores de servicios.</li> </ul> </li> <li><b>Servicios y Proveedores:</b> <ul style="list-style-type: none"> <li><b>OPcPurchase OrderService:</b> Servicio que interactúa con el <b>Bank</b> a través de un <b>CreditCard Service</b>.</li> <li><b>OPcOrder TrackingService:</b> Servicio que interactúa con el <b>Activity Provider</b> a través de un <b>ActivityPO Service</b>.</li> <li><b>AirlinePO Service:</b> Servicio que interactúa con el <b>Airline Provider</b>.</li> <li><b>LodgingPO Service:</b> Servicio que interactúa con el <b>Lodging Provider</b>.</li> </ul> </li> <li><b>Interacción:</b> El <b>User's e-mail client</b> interactúa con el <b>OPC</b>. El <b>Consumer Web site</b> utiliza los servicios <b>OPcPurchase OrderService</b> y <b>OPcOrder TrackingService</b> para realizar operaciones.</li> </ul>

	<p><b>Key:</b></p> <ul style="list-style-type: none"> <li>Client-side application</li> <li>Java EE application</li> <li>External Web service provider</li> <li>Web services endpoint</li> <li>Data repository</li> <li>HTTP/HTTPS</li> <li>SOAP call</li> <li>Data access</li> <li>SMTP</li> <li>Scope of the application (not a component)</li> </ul>
Ejemplos	Proveen servicios SOPA entre otros: AMAZON WS AFIP WS

Tipo C&C	<b>Publicador Subscriptor (Publish-Subscribe)</b>
Contexto	Hay una cantidad de productores y consumidores independientes de datos que deben interactuar. La cantidad y naturaleza precisas de los productores y consumidores de datos no están predeterminadas o fijas, ni los datos que comparten.
Problema	¿Cómo podemos crear mecanismos de integración que apoyen la capacidad de transmitir mensajes entre los productores y consumidores de tal manera que no estén conscientes de la identidad de los demás, o potencialmente incluso de su existencia?
Solución	<p>En arquitectura de publicación-suscripción, los componentes interactúan a través de mensajes anunciados o eventos. Los componentes pueden suscribirse a un conjunto de eventos. El trabajo de la infraestructura de tiempo de ejecución de publicación-suscripción es asegurarse de que cada evento publicado se entregue a todos los suscriptores de ese evento. Así, la principal forma de conector en estas arquitecturas es un bus de eventos. Los componentes del editor colocan eventos en el bus al anunciarlos; el conector luego entrega esos eventos a los componentes del suscriptor que han registrado un interés en esos eventos. Cualquier componente puede ser tanto un editor como un suscriptor. Publicación-suscripción agrega una capa de direccionamiento indirecto entre los remitentes y los receptores. Esto tiene un efecto negativo en la latencia y la escalabilidad potencial, dependiendo de cómo se implementa. Por lo general, uno no desearía utilizar publicación-suscripción en un sistema que tenía que cumplir con los plazos de entrega en tiempo real, ya que introduce incertidumbre en los tiempos de entrega de los mensajes.</p> <p>Además, la arquitectura de publicación-suscripción sufre porque proporciona menos control sobre el orden de los mensajes, y la entrega de mensajes no está garantizada (porque el remitente no puede saber si un receptor está escuchando). Esto puede hacer que la arquitectura de publicación-suscripción</p>

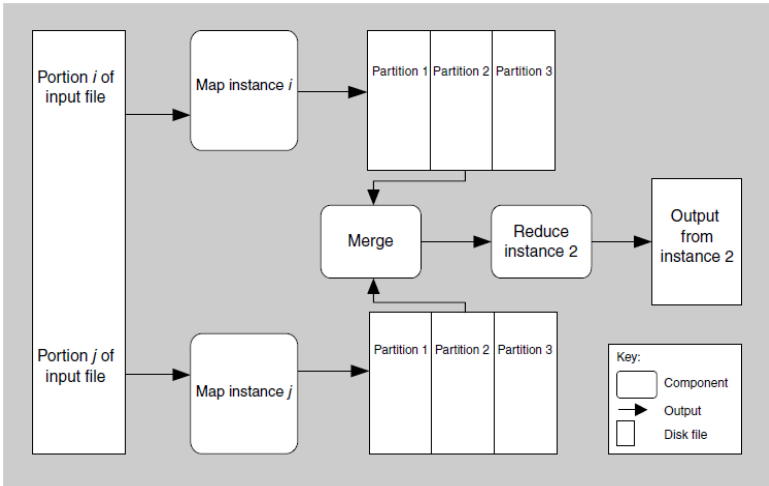
	sea inadecuado para interacciones complejas donde el estado compartido es crítico.
Elementos	Cualquier componente de C&C con al menos un puerto de publicación o suscripción. Se deben tener en cuenta los eventos que se publican y suscriben, y la granularidad de los eventos.
Debilidades	Típicamente, aumenta la latencia y tiene un efecto negativo en la escalabilidad y la previsibilidad del tiempo de entrega del mensaje. No se garantiza un menor control sobre el pedido de mensajes y la entrega de mensajes.
Ejemplos	<a href="#">Apache Kafka</a>

#### Gráficamente



#### Key:



Asignación	<b>Map-Reduce</b>
Contexto	<p>Las empresas tienen una necesidad urgente de analizar rápidamente los enormes volúmenes de datos que generan o acceden, a escala de petabyte. Los ejemplos incluyen registros de interacciones en un sitio de red social, documentos masivos o repositorios de datos, y pares de enlaces de &lt;source, target&gt; para un motor de búsqueda. Los programas para el análisis de estos datos deben ser fáciles de escribir, ejecutar de manera eficiente y ser resistentes con respecto a fallas de hardware.</p>
Solución	<p>Para muchas aplicaciones con conjuntos de datos muy grandes, basta con ordenar los datos y luego analizar los datos agrupados. El problema que resuelve la arquitectura map reduce consiste en realizar de manera eficiente y paralela una agrupación de un gran conjunto de datos y proporcionar un medio simple para que el programador especifique el análisis que se realizará.</p> <p>La arquitectura map reduce requiere tres partes: una infraestructura especializada se encarga de asignar software a los nodos de hardware en un entorno de computación masivamente paralelo y maneja la clasificación de los datos según sea necesario. Un nodo puede ser un procesador independiente o un núcleo en un chip de múltiples núcleos.</p> <p>El segundo y el tercero son dos funciones codificadas por el programador llamadas: mapear y reducir.</p>
Elementos	<p>El <b>mapa</b> es una función con varias instancias implementadas en varios procesadores que realiza las partes de extracción y transformación de el análisis.</p> <p><b>La Función Reducir</b> que se puede implementar como una sola instancia o como múltiples instancias en los procesadores para realizar la parte de carga de la extracción de la transformación de la transformación.</p> <p><b>La infraestructura</b> es el marco responsable de implementar el mapa y reducir las instancias, guardar los datos entre ellos, y detectar y recuperar errores.</p>
Gráficamente	 <pre> graph LR     P1[Portion i of input file] --&gt; M1[Map instance i]     P2[Portion j of input file] --&gt; M2[Map instance j]     M1 --&gt; P1_1[Partition 1]     M1 --&gt; P1_2[Partition 2]     M1 --&gt; P1_3[Partition 3]     M2 --&gt; P2_1[Partition 1]     M2 --&gt; P2_2[Partition 2]     M2 --&gt; P2_3[Partition 3]     P1_1 --&gt; M[Merge]     P1_2 --&gt; M     P1_3 --&gt; M     P2_1 --&gt; M     P2_2 --&gt; M     P2_3 --&gt; M     M --&gt; R[Reduce instance 2]     R --&gt; O[Output from instance 2] </pre> <p>Key:  <span style="border: 1px solid black; display: inline-block; width: 15px; height: 15px;"></span> Component  <span style="display: inline-block; width: 15px; height: 15px; border-bottom: 1px dashed black;"></span> Output  <span style="border: 1px dashed black; display: inline-block; width: 15px; height: 15px;"></span> Disk file </p>
Ejemplos	<a href="#">Apache Hadoop</a>



## **Bibliografía**

Software Architecture in Practice Third Edition, Len Bass, Paul Clements, Rick Kazman