Arquitectura de referencia TRATEME

Versión 1.0

Fecha 21/08/2018

Preparado para:

Raúl Iván Mazo

Preparado por:

Richard Ruiz
Jhon Medina
Jean Paul Galeano
Andrés López

1 Descripción

Tras realizar un análisis detallado de la LPS (Línea de Productos de Software) salen a la luz diferentes condiciones que se deben de cumplir para que exista un correcto funcionamiento en cada aplicación dado el contexto y categoría del software que se desea implementar. En esta línea de negocio en particular se hace necesario que cada cliente cuente con unas características específicas que le permitirán hacer uso de las aplicaciones como lo es una conexión a internet estable y una dirección IP publica y estática la cual será agregada a una WHITE-LIST previo al uso de la aplicación que desee utilizar, además de esto se hace necesario que este cliente a nivel interno pueda transar con el estilo arquitectónico REST, ya que las aplicaciones de esta LPS son *aplicaciones orientadas a servicios* las cuales entregan un bajo acoplamiento y una serie de características que dado el contexto de negocio facilitaran realizar el acceso a una la plataforma remota donde las aplicaciones expondrán las funcionalidades requeridas en formato JSON que proveerá la comunicación a través de canales HTTP, los cuales por medio de JWT (JSON WEB Token) facilitaran la gestión de la seguridad brindando autenticación y validación entre los diferentes componentes de la LPS.

Para lograr cumplir con este objetivo se usará tanto el estilo arquitectónico orientado a microservicios que facilita la construcción de servicios independientes lo que permitirá que cada uno sea desarrollados en el lenguaje de programación más apto para generar un mejor desempeño permitiendo que estos componentes hagan parte de un sistema completamente funcional y el estilo arquitectónico REST (Representational State Transfer) esta tecnología funciona con HTTP lo cual facilita la comunicación entre lenguajes lo que las hace mantenibles y escalables.

Teniendo en cuenta lo anterior se platea realizar un despliegue distribuido basado en una estrategia *Cliente-Servidor.*

Para los aspectos técnicos de la aplicación es importante tener en cuenta una serie de factores que pueden afectar el correcto funcionamiento de las API, es por este que su método de construcciones debe de facilitar la contenerización, la administración de dicha contenerización y el escalamiento horizontal. Para lograr estas metas se a optado por desarrollar las aplicaciones en lenguajes orientaos a objetos tales como java, .Net y NodeJS que permiten la extensión del funcionamiento de los microservicios, esto a su vez permitirá un alojamiento en múltiples sistemas operativos dado las ventajas entregadas por este tipo de aplicaciones.

2 Definición de la arquitectura de referencia.

1. Identificar la variabilidad en los tipos de aplicaciones

- Las aplicaciones deben de tener acceso a internet.
- Las aplicaciones deben poder transar con el estilo arquitectónico REST
- Las aplicaciones deben de contar con una IP estática, para validar la conexión de la aplicación hacia los microservicios por medio WHITE LIST.
- Las aplicaciones deben tener la capacidad de generación de JWT (JSON WEB TOKEN).
- Las aplicaciones deben de tener la capacidad de generar mensajes en formato JSON.
- Los componentes de dominio estarán orientadas a familias de aplicaciones orientadas a API REST, diseñadas para soportar comunicaciones entre componentes poco acoplados.

2. **Identificar la variabilidad en el estilo arquitectónico**

- Se usará el estilo arquitectónico orientado a Microservicios combinado con el estilo arquitectónico REST (Representational State Transfer) .
- La comunicación está orientada a aplicaciones API REST.
- Las API estarán expuestas remotamente en Internet.

3. **Identificar la variabilidad en las estrategias de despliegue**

- Las aplicaciones deben de soportar despliegues distribuidos.
  Nota: Afecta la latencia de respuesta hacia la aplicación final, pero existen estrategias que ayudan a mejorar por ejemplo Arquitecturas orientadas a cache.
- Las aplicaciones deben diseñarse siendo tolerables a fallos totales.
- Las aplicaciones deben soportar grandes volúmenes de transacciones, facilidad, flexibilidad, elasticidad.
- Las aplicaciones deben manejar algún patrón de transaccionalidad distribuida.
- Las aplicaciones se deben desplegar cliente – servidor. Capa Cliente (Servidor web), Capa de Aplicación (Servidor de Aplicaciones).

4. **Identificar la variabilidad en los aspectos técnicos**

- Las aplicaciones deben poder transar con REST API.
- Las aplicaciones deben tener canal de comunicación (Red, Puertos, Formato, etc) habilitados para poder transar contra los componentes de domino.
- Las aplicaciones deben de soportar la tecnología suficiente de implementación API y deben tener las capacidades suficientes para cubrir la capacidad de contenerizarse y escalamiento horizontal, despliegue automático, y administrador de contenedores.
- Las aplicaciones deben poder utilizar cualquier distribución de Linux.

5. **Construir el modelo de variabilidad de la arquitectura de referencia**

6. Documentar la(s) arquitectura(s) de referencia

Las vistas son adicionadas como adjuntos a este documento