FaaSioFlex: Plataforma FaaS para la Ejecución y Gestión Dinámica de Funciones

Escalabilidad y Elasticidad en un Entorno Descentralizado Basado en Docker

Chunlu Hu Juan Amilcar Tito Tito Carlos Caramagna
José Javier Gutiérrez Gil

2024

Chapter 1

Introducción a FaaSioFlex

FaaSioFlex es una plataforma de Function-as-a-Service (FaaS) que permite a los desarrolladores y usuarios gestionar y ejecutar funciones en un entorno escalable y flexible. Su diseño modular facilita el registro y la implementación de funciones en contenedores **Docker**, ofreciendo una solución ágil para el desarrollo y despliegue de aplicaciones basadas en microservicios.

La plataforma se integra con componentes clave como **Dex** para la autenticación, **NATS** para la mensajería y colas, y **APISIX** como enrutador y proxy inverso, garantizando una comunicación eficiente entre servicios. FaaSioFlex no solo permite a los usuarios registrar sus funciones, sino que también optimiza la asignación de recursos, asegurando **escalabilidad** y **elasticidad** en la ejecución de tareas. La implementación completa en **Golang** proporciona un rendimiento robusto y una fácil integración con otros sistemas.

El servicio FaasioFlex se implementa utilizando contenedores Docker, lo que permite un entorno aislado y escalable para la ejecución de funciones. Cada función registrada por un usuario se ejecuta en su propio contenedor independiente, garantizando que las operaciones de cada función no interfieran entre sí. FaasioFlex gestiona la comunicación entre módulos utilizando NATS, que actúa como cola de mensajes para coordinar la ejecución de funciones. Esto incluye la coordinación de eventos del sistema, como la creación de nuevas funciones y la escalabilidad y elasticidad de las instancias de cada función en ejecución, asegurando una comunicación eficiente entre los componentes del sistema. Los usuarios, después de registrarse en el sistema, tienen la capacidad de registrar sus propias funciones, las cuales pueden ser invocadas por otros usuarios. Cada instancia de función corre en un contenedor Docker aislado, lo que permite un manejo eficiente de los recursos y una mejor escalabilidad.

Con un enfoque en la facilidad de uso y la flexibilidad, FaaSioFlex se destaca como una solución innovadora para quienes desean aprovechar al máximo el potencial de la computación sin servidor basada en microservicios y cloud computing en sus proyectos.

1.1 Definición del Alcance y Funcionalidades

Funcionalidades Clave del Proveedor de Servicios FaaSioFlex

El proyecto para crear un proveedor de servicios FaaS tiene como objetivo ofrecer una plataforma que permita a los desarrolladores ejecutar funciones en la nube de forma escalable y eficiente, eliminando la necesidad de gestionar los servidores subyacentes. La plataforma proporcionará las siguientes funcionalidades principales:

- **Deployment del Servicio FaaS**: Describe cómo se implementará el despliegue

de funciones, incluyendo la escalabilidad automática y el soporte para múltiples lenguajes de programación.

- **Independencia de Funciones**: En el entorno FaaSioFlex, cada función se puede empaquetar en su propia imagen Docker. Esto significa que cada vez que se invoca una función, se puede crear una nueva instancia de esa imagen para ejecutarla. Esto proporciona un alto grado de aislamiento entre funciones, ya que cada una tiene su propio entorno de ejecución.
- **Recursos Eficientes**: Utilizar Docker para ejecutar funciones permite que los recursos de la infraestructura se utilicen de manera más eficiente. Dado que las imágenes Docker son ligeras y se pueden iniciar rápidamente, los tiempos de arranque de las funciones se reducen, mejorando así la experiencia del usuario.
- **Ejecución de Funciones a Demanda**: Los usuarios podrán registrar funciones que estarán disponibles para ser invocadas por otros usuarios en cualquier momento. Cuando se realiza una invocación, la función comenzará a consumir los recursos necesarios para su ejecución. El propio servicio FaaS gestionará automáticamente la asignación de estos recursos, asegurando su uso óptimo de acuerdo con la demanda en cada momento.

  Para facilitar la invocación de estas funciones, se implementará una API que permitirá desencadenar la ejecución de cada función. Las invocaciones podrán realizarse de diversas maneras, lo que proporcionará una integración flexible tanto con el entorno como con otros servicios:

  - **En respuesta a eventos específicos**: Por ejemplo, la función puede ejecutarse automáticamente cuando ocurre un evento, como una solicitud HTTPS, un mensaje en una cola de mensajería (NATS), o un cambio en una base de datos.
  - **Invocación directa**: Los usuarios pueden llamar a la función directamente a través de la API en cualquier momento, lo que es útil para situaciones donde se necesita una respuesta inmediata o procesamiento bajo demanda.

  Para la implementación de la API se utiliza un **enfoque serverless**, donde cada función se despliega y se ejecuta como un servicio independiente. Ello permite:

  - **Escalabilidad**: Permite escalar automáticamente según la demanda en cada instante, lo que puede ser eficiente en costos.
  - **Elasticidad**: Asegura que la capacidad de recursos puede ser incrementada o decrementada dinámicamente en respuesta a las fluctuaciones de la demanda, optimizando así los recursos disponibles.

  - **Desarrollo Ágil**: Facilita el desarrollo rápido y la implementación continua.
  - **Mantenimiento Reducido**: FaaSioFlex se encargará de la infraestructura subyacente, lo que reducirá la carga de mantenimiento para el desarrollador.

- **Escalabilidad Automática**: La plataforma ajustará automáticamente el número de instancias en ejecución de cada función según la demanda, garantizando un uso eficiente de los recursos y optimizando costos. Cuando la demanda de una función específica aumenta, el proveedor FaaS puede iniciar múltiples instancias de la imagen Docker correspondiente para manejar el tráfico. Esto permite que la infraestructura escale automáticamente en función de la carga de trabajo, asegurando que se satisfagan las solicitudes de los usuarios sin necesidad de intervención manual.
- **Integración con Servicios de Terceros**: Se ofrecerán conectores nativos para integrar las funciones con servicios externos, como bases de datos, almacenamiento en la nube, colas de mensajería y otros servicios de API.
- **Gestión de Versiones y Despliegue de Funciones**: Los desarrolladores podrán gestionar diferentes versiones de sus funciones y realizar despliegues continuos sin interrumpir el servicio.
- **Soporte para Múltiples Lenguajes de Programación**: La plataforma será compatible con varios lenguajes de programación, como Python, JavaScript, Go, y otros, para permitir a los desarrolladores trabajar con el lenguaje de su elección.
- **Independencia de Funciones**: Cada función se puede empaquetar en su propia imagen Docker. Esto significa que cada vez que se invoca una función, se puede crear una nueva instancia de esa imagen para ejecutarla. Esto proporciona un alto grado de aislamiento entre funciones, ya que cada una tiene su propio entorno de ejecución.
- **Control de Seguridad Interna Automática**: La plataforma implementará controles de seguridad interna automatizados que incluirán auditoría de acciones, prevención de pérdida de datos (DLP), y análisis de comportamiento de usuarios.
  - **Auditoría de Acciones:** La plataforma registrará automáticamente todas las acciones realizadas por los usuarios, incluyendo el registro de funciones, despliegues, ejecuciones y cualquier acceso a datos sensibles. Estos registros se mantendrán en un sistema seguro para cumplir con los estándares de auditoría y facilitar la identificación de actividades sospechosas.
  - **Prevención de Pérdida de Datos (DLP):** Se integrarán mecanismos de

DLP para monitorear y proteger la información sensible que pueda ser manipulada o transferida a través de las funciones. Esto incluirá políticas de cifrado y restricciones sobre dónde se pueden almacenar y enviar los datos. La plataforma identificará automáticamente la información sensible y tomará acciones preventivas, como bloquear o alertar sobre intentos de acceso no autorizado.

  - **Análisis de Comportamiento de Usuarios:** La implementación de herramientas de análisis de comportamiento permitirá detectar anomalías en las acciones de los usuarios. Estas herramientas utilizarán algoritmos de aprendizaje automático para identificar patrones de uso normales y generar alertas si se detectan comportamientos que se desvían significativamente de la norma.
  - **Notificaciones Automáticas:** Ante la detección de actividades sospechosas o violaciones de políticas de seguridad, el sistema enviará notificaciones automáticas a los administradores para que tomen las medidas necesarias de forma proactiva. Esto asegurará una respuesta rápida ante posibles incidentes de seguridad.
  - **Control de Vulnerabilidades:** Se implementarán evaluaciones regulares de vulnerabilidades en el servicio para identificar y mitigar riesgos potenciales. Esto incluirá escaneos automáticos y revisiones de seguridad para garantizar que las funciones y servicios se mantengan protegidos contra amenazas conocidas. Además, se llevarán a cabo pruebas de penetración para evaluar la resistencia del sistema ante ataques maliciosos.
  - **Detección de Intrusos:** Se implementarán sistemas de detección de intrusos (IDS) que monitorizarán continuamente el tráfico de red y las actividades del sistema para identificar patrones anómalos que puedan indicar un intento de intrusión. Estas soluciones analizarán los datos en tiempo real y generarán alertas cuando se detecten actividades sospechosas.
  - **Detección de Anomalías:** Utilizando algoritmos avanzados de aprendizaje automático, la plataforma será capaz de identificar anomalías en los patrones de uso y tráfico, lo que permitirá detectar posibles brechas de seguridad o comportamientos maliciosos. Las anomalías pueden incluir accesos no autorizados, intentos de acceso fuera de horario habitual o actividades inusuales en el uso de recursos.

- **Mecanismos de Recuperación y Respaldo**: Implementación de procesos para asegurar la disponibilidad continua de datos y la recuperación en caso de fallo.

  - **Replicación**: Creación de copias exactas de datos en tiempo real para asegurar la disponibilidad continua.
  - **Backup**: Realización de copias de datos en intervalos específicos, protegiendo contra pérdidas de datos y permitiendo la restauración a estados anteriores.

- **Mejora Continua:** La plataforma adoptará un enfoque de mejora continua, analizando periódicamente los informes de seguridad, las auditorías y las métricas de rendimiento. Se implementarán mejoras basadas en los hallazgos para optimizar la seguridad y la eficiencia del servicio. Este enfoque permitirá adaptar rápidamente las políticas y controles a medida que evolucione el panorama de amenazas y las necesidades de los usuarios.

1.2 Diseño y Experiencia del Usuario (UX/UI)

Interfaz Intuitiva

El diseño de la interfaz de usuario está enfocado en proporcionar una experiencia sencilla y eficiente, tanto para desarrolladores experimentados como para principiantes. Los elementos clave del diseño incluyen:

- **Panel de Control Centralizado**: Una consola unificada que permite a los usuarios gestionar sus funciones, ver métricas de rendimiento, configurar eventos disparadores y acceder a la documentación de la API.
- **Despliegue en un Solo Clic**: Simplificación del proceso de despliegue, permitiendo a los desarrolladores subir código, configurar parámetros de ejecución y activar funciones en la nube con un solo clic.
- **Editor de Código Integrado**: El panel incluirá un editor de código en línea con resaltado de sintaxis y autocompletado, permitiendo a los desarrolladores editar sus funciones directamente desde la interfaz.
- **Desarrollo de API**: La interfaz de usuario se construye sobre una API estandarizada, que facilita el registro de funciones que pueden ser invocadas por los usuarios. Esta API asegura que, al ser invocada a través del medio deseado, se produzca la ejecución de la función correspondiente. De esta manera, se optimiza tanto la disponibilidad como el rendimiento de las funciones, ajustándose a la demanda. La implementación de FaaSioFlex es tan flexible que permite el uso del servicio FaaS exclusivamente a través de esta API; incluso sin un frontend, el servicio funcionará sin problemas, ofreciendo a los desarrolladores la libertad de integrarlo en cualquier entorno que deseen.

Personalización del Usuario

Para ofrecer una experiencia más adaptada a las necesidades individuales, se incluirán las siguientes opciones de personalización:

- **Configuración de Notificaciones**: Los usuarios podrán definir alertas personalizadas para eventos específicos, como errores en la ejecución de funciones o escalados automáticos.
- **Ajuste de Parámetros de Ejecución**: Los desarrolladores tendrán la capacidad de personalizar los límites de memoria, tiempo de ejecución y número máximo de instancias para cada función.
- **Temas y Preferencias de la Consola**: La interfaz será personalizable con diferentes temas (claro, oscuro, etc.) y opciones de diseño para adecuarse a las preferencias de cada usuario.

1.3 Aspectos Técnicos

Multiplataforma y Disponibilidad en Diversos Dispositivos

El proveedor de servicios FaaS se diseñará para ser accesible y usable desde una variedad de dispositivos, asegurando que los usuarios puedan gestionar sus funciones sin inconvenientes. Los elementos clave del diseño incluyen:

- **Arquitectura en Contenedores**: La implementación se llevará a cabo de tal manera que el proveedor de servicios FaaS se ejecutará dentro de una imagen Docker. Cada función será independiente, ejecutándose en su propia instancia de Docker. Esto permite un alto grado de aislamiento y facilita la escalabilidad, ya que, en caso de necesidad, se pueden iniciar múltiples instancias de la misma función para gestionar el tráfico. Así, se garantiza que las funciones puedan ser gestionadas y desplegadas de manera eficiente y flexible en un entorno multiplataforma.
- **Creación de una API para el Usuario**: El proveedor de servicios FaaS será accesible a través de una aplicación web responsive ~~y aplicaciones móviles nativas para iOS y Android~~. Los usuarios podrán gestionar sus funciones desde cualquier dispositivo con conexión a internet, asegurando la disponibilidad continua del servicio.
- **Compatibilidad con Diversos Navegadores**: La aplicación web será compatible con todos los navegadores modernos, garantizando una experiencia de usuario consistente.
- ~~**Aplicaciones Móviles Nativas**: Se desarrollarán aplicaciones móviles con interfaces~~

~~adaptadas para optimizar la gestión desde dispositivos con pantallas más pequeñas.~~

- **Entorno de despliegue:** El objetivo final es que nuestro servicio FaaSioFlex se despliegue en Kimodo. Kimodo es una plataforma que simplifica la gestión y orquestación de aplicaciones en la nube. Proporciona herramientas para facilitar el despliegue, escalado y monitoreo de microservicios, lo que la convierte en una opción ideal para entornos de desarrollo y producción. Kimodo ayuda a gestionar la complejidad de la infraestructura subyacente, permitiendo a los desarrolladores centrarse en el desarrollo de aplicaciones.
- **Entorno de Desarrollo:** Los desarrolladores implementan FaasioFlex en sus máquinas locales utilizando Docker. Esto permite pruebas rápidas y ajustes en el código sin necesidad de un entorno de producción.
  - **Local:** Los colaboradores que utilicen Windows o Mac deben usar Docker Desktop para tener un servidor de ejecución de contenedores.
  - **Contenedores:** Cada función se ejecuta en un contenedor independiente, asegurando que las dependencias y configuraciones no se vean afectadas por otras funciones.
  - **Orquestación:** Docker Compose se utiliza para definir y ejecutar múltiples servicios, facilitando la gestión de las dependencias entre ellos, como la comunicación entre APISIX, DEX y NATS.
  - **Integración con GitHub:** El repositorio del código Go y otros componentes necesarios se mantendrá en un repositorio del grupo de trabajo en GitHub. Es esencial integrar el entorno local de desarrollo con GitHub para subir lo que se está implementando en el repositorio. **SÓLO SE PODRÁ SUBIR AL REPOSITORIO AQUELLO QUE SE HAYA TESTADO EN EL ENTORNO LOCAL Y QUE FUNCIONE CORRECTAMENTE EN LA INTEGRACIÓN CON EL RESTO DE MÓDULOS.**
  - **Configuración del Repositorio:** Se configurará el repositorio de tal manera que, cuando se modifique un archivo en una aplicación o módulo asociado al proyecto, este módulo se recompilará y se creará una nueva imagen Docker que se almacenará en Docker Hub. Por lo tanto, es imperativo que todo lo que se suba al repositorio haya sido probado de forma integral con el resto de módulos y no produzca fallos.
- **Integración de Kimodo con GitHub** Para integrar Kimodo con GitHub, se pueden seguir los siguientes pasos:

1. **Conectar Kimodo con GitHub:** En la interfaz de Kimodo, se debe proporcionar acceso a la cuenta de GitHub donde se encuentra el repositorio del proyecto. Esto se puede hacer a través de las configuraciones de integración de servicios.
2. **Configurar Webhooks:** Crear un webhook en el repositorio de GitHub para que Kimodo reciba notificaciones sobre eventos, como pushes o pull requests. Esto permite que Kimodo despliegue automáticamente nuevas versiones de la aplicación.
3. **Definir Pipelines:** En Kimodo, se pueden definir pipelines de despliegue que especifiquen cómo debe construirse y desplegarse la aplicación cada vez que se detecte un cambio en el repositorio de GitHub.
4. **Monitorear Despliegues:** Utilizar las herramientas de monitoreo de Kimodo para observar el estado de los despliegues y asegurarse de que las versiones más recientes de la aplicación se estén ejecutando correctamente.

Integración de Modelos de Ejecución y Seguridad

El proyecto incluirá la implementación de modelos de ejecución seguros para garantizar la integridad y aislamiento de las funciones.

- **Sandboxing y Aislamiento de Ejecución**: Las funciones se ejecutarán en entornos aislados para prevenir que el código de un usuario afecte los recursos de otros.Las funciones en el entorno FaaS se ejecutarán en entornos aislados mediante el uso de imágenes Docker. Al hacer esto, se garantiza que:
  - **Seguridad Aumentada:** Cada función opera dentro de un sandbox que limita su acceso a los recursos del sistema y a los datos de otras funciones, reduciendo el riesgo de que un código malicioso comprometa la integridad del sistema global.
  - **Flexibilidad:** Al empaquetar cada función en su propia imagen Docker, se facilita la creación de instancias de funciones de manera dinámica según la demanda. Esto no solo mejora la eficiencia en el uso de recursos, sino que también permite una gestión más sencilla de las actualizaciones y versiones de las funciones.
  - **Mantenimiento Simplificado:** El aislamiento de ejecución implica que cualquier fallo o problema dentro de una función no afectará a otras funciones o al sistema en su conjunto, lo que simplifica el proceso de mantenimiento y mejora la estabilidad de la plataforma.

- **Políticas de Seguridad y Autenticación**: Se implementarán controles de acceso, integración con sistemas de identidad (OAuth, OpenID), y gestión de permisos basados en roles (RBAC).
- **Optimización del Rendimiento y Cacheo**: La plataforma permitirá el uso de estrategias de cacheo de funciones para mejorar el rendimiento en casos de invocaciones repetidas.

1.4 Módulos de la Arquitectura del Servicio FaaS

El servicio FaaS está compuesto por los siguientes módulos clave:

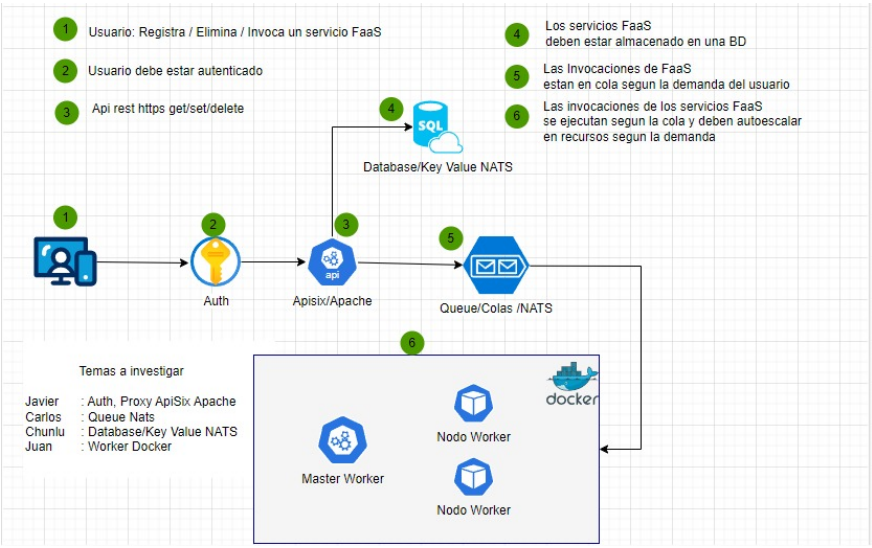

Figure 1.1: Arquitectura de FaaSioFlex vo

- **Gestor de Funciones (Function Manager)**: Encargado del registro, despliegue y gestión de las funciones de los usuarios. Cada función se empaqueta como un contenedor Docker y se gestiona a través de Docker Compose.
- **API Gateway (APISIX)**: Actúa como proxy de entrada, manejando las solicitudes externas y redirigiéndolas a las funciones registradas. Proporciona enrutamiento y balanceo de carga, además de verificar la autenticación mediante Dex.
- **Autenticación y Autorización (Dex)**: Gestiona la autenticación de usuarios y la generación de tokens OAuth2. Está integrado con APISIX para validar los accesos a las funciones.

- **Mensajería y Orquestación (NATS)**: Facilita la comunicación interna entre servicios mediante colas de mensajes. También se utiliza para coordinar el balanceo de carga y la escalabilidad.
- **Monitor de Escalabilidad (Auto-Scaler)**: Supervisa las métricas de uso del sistema y ajusta el número de contenedores de funciones en ejecución para asegurar la escalabilidad automática.
- **Registro de Funciones y Estado (Database/Registry)**: Base de datos ligera para almacenar información sobre las funciones registradas, sus configuraciones y el estado del sistema. Se utilizarña la basde de datos que viene con Nats.

1.5 Flujo del Sistema y Conexiones entre Módulos

Registro y Autenticación de Usuarios

Los usuarios se autentican a través de Dex, que proporciona tokens de autenticación basados en OAuth2. APISIX verifica estos tokens antes de permitir el acceso a los endpoints protegidos.

Registro y Gestión de Funciones

El Gestor de Funciones permite que los usuarios registren sus funciones como imágenes Docker. La información de cada función, incluyendo la configuración de recursos, se almacena en la base de datos. Esto facilita la gestión y el acceso a las funciones desplegadas en la plataforma.

Enrutamiento de Solicitudes

APISIX redirige las solicitudes a las funciones correspondientes en función del identificador único de cada una. Para el balanceo de carga, APISIX se integra con NATS, distribuyendo las solicitudes entre las instancias activas de forma eficiente.

Despliegue y Escalabilidad de Funciones

El Gestor de Funciones utiliza Docker Compose para iniciar o detener contenedores según la demanda. Un monitor de escalabilidad ajusta el número de instancias en función de métricas de rendimiento como la latencia y el uso de CPU. Además, se implementan estrategias de escalabilidad horizontal y vertical.

Mensajería y Orquestación con NATS

NATS gestiona la comunicación entre módulos y actúa como cola de mensajes para coordinar la ejecución de las funciones. Esto incluye la coordinación de eventos del sistema, como la creación de nuevas funciones o la escalabilidad, asegurando una

comunicación eficiente entre componentes.

Gestión de la Base de Datos

La base de datos almacena el estado de las funciones, incluyendo configuraciones y métricas de uso, lo que permite un seguimiento y gestión efectiva de las operaciones.

1.6 Estrategias de Escalabilidad y Asignación de Recursos

Escalado Basado en Métricas

Un monitor de métricas supervisa el uso de CPU, memoria y latencia para ajustar el número de instancias. Por ejemplo, se pueden añadir más instancias si la CPU supera el 80%, asegurando así un rendimiento óptimo.

Auto-Scaling Horizontal y Vertical

- **Escalado Horizontal**: Aumenta o disminuye el número de contenedores según la carga.
- **Escalado Vertical**: Ajusta los recursos (CPU/memoria) asignados a los contenedores según las necesidades de cada función.

1.7 Implementación con Docker y Docker Compose

Para la implementación del proveedor de servicios FaaS, se utiliza Docker y Docker Compose para orquestar los diferentes servicios que componen la plataforma. La configuración del archivo docker-compose.yml define los servicios clave: APISIX, DEX, NATS, Gestor de Funciones y Auto-Scaler. A continuación, se especifican las características principales de la configuración:

- **Definición de Servicios**: Se deben definir los contenedores para cada componente del sistema.
- **Límites y Solicitudes de Recursos**: Especificar los límites de recursos y las solicitudes para cada contenedor, para garantizar que el sistema pueda escalar eficientemente sin consumir más recursos de los necesarios.
- **Red y Dependencias**: Configurar la red para la comunicación entre contenedores y especificar dependencias para asegurar el orden de inicio correcto

Configuración del Archivo `docker-compose.yml`

La implementación del servicio FaasioFlex utiliza Docker Compose para gestionar múltiples componentes de manera eficiente: A continuación, se presenta un ejemplo simplificado del archivo `docker-compose.yml` utilizado para la implementación del

servicio FaasioFlex:

```
version: '3.8'
services:
  apisix:
    image: apache/apisix
    ports:
      - "80:9080"
    volumes:
      - ./config/apisix:/usr/local/apisix/conf
  dex:
    image: dexidp/dex
    ports:
      - "5556:5556"
    volumes:
      - ./config/dex:/etc/dex
  nats:
    image: nats
    ports:
      - "4222:4222"
  function_manager:
    build: ./function_manager
    environment:
      - NATS_URL=nats://nats:4222
  auto_scaler:
    build: ./auto_scaler
    depends_on:
      - nats
      - function_manager
```

Este archivo configura los componentes necesarios para el funcionamiento del servicio, permitiendo la escalabilidad y el manejo eficiente de funciones en un entorno de microservicios.

Integración con Dex para Autenticación

Dex se configura como un proveedor de identidad para la autenticación de usuarios, facilitando el uso de OAuth2 para la generación de tokens. La integración incluye:

- **Configuración de Dex con un Conector**: Dex se configura con un conector para autenticar usuarios a través de proveedores de identidad externos, como LDAP, GitHub o Google.

- **Validación de Tokens con APISIX**: APISIX valida los tokens de autenticación generados por Dex, garantizando que solo los usuarios autenticados puedan acceder a los servicios protegidos.

Gestión de Mensajería con NATS

NATS se utiliza para gestionar la comunicación entre los diferentes servicios del sistema y para la coordinación de la escalabilidad automática. Los aspectos principales son:

- **Comunicación entre Servicios**: NATS permite la comunicación asíncrona entre los componentes, facilitando la orquestación y el manejo de eventos.
- **Coordinación de Escalabilidad**: NATS también se emplea para enviar mensajes de escalabilidad entre el Auto-Scaler y el Gestor de Funciones, asegurando una respuesta dinámica a los cambios en la carga.

1.8 Conexiones y Comunicación

APISIX - Dex - API FaaS

- APISIX actúa como un proxy inverso que verifica la autenticación con Dex antes de enviar solicitudes a la API FaaS.
- La API FaaS es el punto de entrada para todas las operaciones, gestionando las funciones y las solicitudes de ejecución.

API FaaS - Gestor de Funciones

- La API FaaS comunica el registro, despliegue y eliminación de funciones al Gestor de Funciones.
- El Gestor de Funciones utiliza NATS para notificar a otros servicios sobre cambios en el estado de las funciones.

Gestor de Funciones - Auto-Scaler

- El Gestor de Funciones y el Auto-Scaler intercambian información a través de NATS.
- Las métricas de rendimiento y uso se envían al Auto-Scaler, que ajusta la cantidad de instancias de función según sea necesario.

Base de Datos - NATS

- La base de datos se actualiza en función de los eventos de NATS para mantener el estado y la configuración actualizados.

1.9 Configuración y Estrategias de Escalabilidad

Configurar el Gestor de Funciones para Interactuar con Docker Compose

- Definir servicios en `docker-compose.yml` con límites y solicitudes de recursos (como `cpu_shares`, `memory_limit`, etc.).
- Implementar un servicio Golang para gestionar los contenedores utilizando la API de Docker.

Integrar Dex y APISIX para la Autenticación

- Configurar Dex con un conector de identidad para la autenticación de usuarios.
- Configurar APISIX para validar tokens JWT generados por Dex.

Uso de NATS para la Orquestación y Mensajería

- NATS se encarga de la comunicación entre el Gestor de Funciones, el Auto-Scaler y otros módulos.
- Configurar la base de datos asociada para almacenar el estado de las funciones y las configuraciones de los contenedores.

Implementación del Monitor de Escalabilidad

- El Auto-Scaler monitorea las métricas y ajusta el número de instancias de las funciones utilizando NATS para desencadenar las acciones de escalado.
- Definir políticas de escalado específicas (basadas en el uso de CPU/memoria o el número de solicitudes activas).

Elasticidad

La elasticidad es una capacidad crítica de la plataforma FaaS para adaptarse dinámicamente a las variaciones de la carga de trabajo. Para implementar estrategias de elasticidad efectivas, se pueden considerar las siguientes prácticas:

- **Escalado Automático:** Utilizar el Auto-Scaler para ajustar automáticamente la cantidad de instancias de función según la carga. Esto permite una respuesta rápida a picos en el tráfico.
- **Definición de Umbrales:** Establecer umbrales de uso de recursos (como CPU y memoria) para activar el escalado. Por ejemplo, si el uso de CPU supera un 80%, se pueden añadir instancias adicionales.
- **Escalado Vertical y Horizontal:** Implementar tanto escalado vertical (aumentar

recursos de una instancia) como horizontal (añadir más instancias) según sea necesario.

- **Políticas de Desescalado:** Definir políticas para desescalar, asegurando que las instancias inactivas se eliminen de manera eficiente cuando la carga disminuya.
- **Monitoreo y Alertas:** Implementar sistemas de monitoreo para rastrear el rendimiento de la aplicación y generar alertas para los administradores cuando se detecten anomalías o sobrecargas.

1.10 Funcionalidades de Docker Compose y Kimodo

Docker Compose:

- **Definición de Servicios:** Permite a los desarrolladores definir múltiples servicios en un archivo `docker-compose.yml`, especificando las imágenes de contenedores, redes, volúmenes y configuraciones necesarias.
- **Ejecución Local:** Ideal para entornos de desarrollo locales, ya que permite iniciar, detener y escalar servicios de manera fácil y rápida en una única máquina.
- **Dependencias:** Facilita la gestión de las dependencias entre diferentes servicios, como la comunicación entre APISIX, DEX y NATS.

Kimodo:

- **Orquestación Automática:** Kimodo proporciona una solución de orquestación más avanzada que puede gestionar automáticamente el escalado y la elasticidad de los servicios en la nube. Esto significa que puede aumentar o disminuir la cantidad de instancias de un servicio en función de la carga del sistema.
- **Gestión de Microservicios:** Está diseñado para simplificar la implementación y monitoreo de aplicaciones en un entorno de producción, facilitando la gestión de microservicios y optimizando el uso de recursos.
- **Integración con Docker:** Aunque Kimodo maneja la orquestación, aún puede trabajar en conjunto con Docker y Docker Compose para los entornos de desarrollo locales.

Integración de Docker Compose con Kimodo

Para integrar ambas herramientas en tu flujo de trabajo, puedes seguir estos pasos:

Desarrollo Local:

- Durante la fase de desarrollo, puedes utilizar Docker Compose para definir y ejecutar tu aplicación localmente. Esto te permitirá realizar pruebas rápidas y

ajustes en tu código sin necesidad de desplegarlo en un entorno de producción.

- Asegúrate de que tu `docker-compose.yml` esté configurado correctamente para incluir todos los servicios necesarios, como APISIX, DEX y NATS.

Transición a Kimodo:

- Una vez que tu aplicación esté funcionando correctamente en el entorno local, puedes preparar el despliegue en Kimodo.
- Kimodo se encargará de la orquestación en la nube, gestionando automáticamente el escalado y la elasticidad de los servicios según la demanda. Esto implica que puedes definir las reglas de escalado y las configuraciones de servicio dentro de Kimodo.

Flujo de Trabajo:

- Cuando haces cambios en tu código y realizas un commit en tu repositorio de GitHub, este proceso puede incluir la compilación de nuevas imágenes Docker y su subida a Docker Hub.
- Al implementar en Kimodo, puedes utilizar estas imágenes ya compiladas para crear o actualizar las instancias de tus servicios.

Ejemplo de Flujo de Trabajo

Desarrollo Local:

- Utiliza Docker Compose para ejecutar y probar tu aplicación.
- Realiza cambios en el código y prueba localmente.

Integración y Pruebas:

- Haz un commit en tu repositorio de GitHub. Esto puede activar un pipeline CI/CD que compile la imagen Docker y la suba a Docker Hub.

Despliegue en Kimodo:

- Accede a Kimodo para gestionar la implementación de tus servicios utilizando las imágenes almacenadas en Docker Hub.
- Kimodo orquestará automáticamente la escalabilidad y la resiliencia de tu aplicación en el entorno de producción.

1.11 Implementación de la API en Golang

La API del Servicio FaaS implementada en Go gestionará las siguientes funcionalidades:

Registro de Funciones

- Endpoint para registrar funciones (`/functions/register`), donde los usuarios pueden enviar información sobre la imagen Docker, recursos y configuraciones.
- El backend se comunica con Docker para desplegar la función y actualiza el registro en la base de datos.

Ejecución de Funciones

- Endpoint para ejecutar una función (`/functions/execute`), que enrutará la solicitud al contenedor correspondiente.
- Utiliza NATS para la comunicación entre servicios.

Gestión de Funciones

- Endpoints para listar, actualizar y eliminar funciones.
- Cada operación genera un evento en NATS que otros servicios pueden consumir.

1.12 Ejemplo de Configuración Simplificada del Archivo `docker-compose.yml`

A continuación se muestra un ejemplo básico de configuración del archivo `docker-compose.yml` para ilustrar la integración de los componentes mencionados:

```
version: '3.8'
services:
  apisix:
    image: apache/apisix
    ports:
      - "80:9080"
    volumes:
      - ./config/apisix:/usr/local/apisix/conf
    environment:
      - DEX_URL=http://dex:5556
  dex:
    image: dexidp/dex
    ports:
```

```
    - "5556:5556"
  volumes:
    - ./config/dex:/etc/dex
nats:
  image: nats
  ports:
    - "4222:4222"
function_manager:
  build: ./function_manager
  environment:
    - NATS_URL=nats://nats:4222
  depends_on:
    - nats
auto_scaler:
  build: ./auto_scaler
  environment:
    - NATS_URL=nats://nats:4222
  depends_on:
    - nats
    - function_manager
```

1.13 Consideraciones Adicionales

Cumplimiento Normativo y Protección de Datos

El proveedor de servicios FaaS debe cumplir con las normativas de protección de datos vigentes, como el GDPR o CCPA, para garantizar la privacidad de los datos del usuario.

- **Gestión de Datos Sensibles**: Las funciones que manejen datos sensibles deberán ser configuradas para cumplir con las políticas de cifrado y almacenamiento seguro.
- **Auditoría y Registro de Actividades**: La plataforma registrará las acciones realizadas por los usuarios para mantener un historial de auditoría que cumpla con los requisitos normativos.

Otros cumplimientos normativos para poder obtener supuestas certificaciones o auditorias sobre nuestro servicio FaaS atendiendo a la regularización territorial donde se encuentra el servicio, tanto a nivel local, nacional e internacional.

Estrategias de Monitoreo y Recuperación ante Desastres

La infraestructura de la plataforma deberá contar con mecanismos de monitoreo continuo y planes de recuperación ante fallos.

- **Monitoreo en Tiempo Real**: Se utilizarán herramientas para monitorear el estado de las funciones, latencia, errores y uso de recursos en tiempo real.
- **Copia de Seguridad y Restauración**: La plataforma implementará mecanismos para la realización de copias de seguridad periódicas y restauración de funciones en caso de incidentes.

Distribución del Trabajo

La **Distribución del Trabajo** en el desarrollo del servicio FaaS será clave para asegurar una implementación efectiva y eficiente. A continuación se detalla la asignación de tareas:

- **Arquitectura, Docker ty Docker Comppose**:
  - Diseñar la arquitectura general del servicio.
  - Configuración de Docker y Docker Compose para un despliegue eficiente.
  - Control de versiones utilizando GitLab para gestionar el desarrollo colaborativo.
- **Desarrollador de API (Go)**:
  - Implementación de la API FaaS y los endpoints necesarios para la funcionalidad del servicio.
  - Integración con NATS y la base de datos para la gestión eficiente de datos y comunicación.
- **Gestor de Funciones y Escalado (Go)**:
  - Desarrollo del Gestor de Funciones que facilite el despliegue y la gestión de las funciones.
  - Configuración de políticas de escalado automático y monitorización para asegurar el rendimiento óptimo.
- **Autenticación y Seguridad**:
  - Implementación de Dex para la autenticación de usuarios.
  - Integración con APISIX para la protección de rutas y asegurar la comunicación en el servicio.

1.14 Ventajas de Nuestro Servicio FaaS

Nuestra oferta como proveedor de servicios FaaS se distingue por varias ventajas clave:

- **Escalabilidad Dinámica:** Nuestro sistema de escalado automático permite que las aplicaciones respondan a la demanda de manera eficiente. Las funciones pueden ajustarse rápidamente a picos de tráfico o disminuir en momentos de baja carga, optimizando así el uso de recursos.
- **Flexibilidad de Desarrollo:** Los desarrolladores tienen la libertad de elegir entre múltiples lenguajes de programación para crear sus funciones, lo que les permite trabajar en el entorno con el que se sientan más cómodos. Además, pueden personalizar sus funciones según las necesidades específicas de sus aplicaciones.
- **Integración Sencilla:** La compatibilidad con servicios de mensajería como NATS y bases de datos garantiza que los desarrolladores puedan construir aplicaciones complejas sin complicaciones. Esto facilita la integración de distintos componentes en una arquitectura basada en microservicios.
- **Soporte y Monitoreo:** Ofrecemos herramientas de monitoreo integradas que permiten a los usuarios supervisar el rendimiento y optimizar sus funciones en tiempo real. Los desarrolladores pueden acceder a métricas detalladas para detectar problemas rápidamente y aplicar mejoras de forma proactiva.
- **Seguridad Robusta:** Implementamos medidas de seguridad sólidas, incluyendo autenticación mediante Dex, políticas de cifrado, y auditoría de acciones, para proteger los datos y la integridad del servicio. Nuestro enfoque garantiza un entorno seguro y conforme a estándares de la industria.

1.15 Consideraciones Finales

Para garantizar un rendimiento óptimo y evitar la saturación del sistema, es crucial ajustar las políticas de escalabilidad y la configuración de recursos de manera adecuada. Se recomienda revisar y ajustar las configuraciones periódicamente según los patrones de uso del sistema.

Con este enfoque, se busca ofrecer un servicio robusto y eficiente que satisfaga las necesidades de los desarrolladores y usuarios finales, alineándose con las tendencias actuales en la computación en la nube y los microservicios.