Manual de Usuario de TinyFaaS MQTT

Este manual detalla cómo interactuar con el servidor **TinyFaaS MQTT**, un sistema de Functions as a Service (FaaS) que utiliza el protocolo MQTT para la comunicación. El manual está dividido en tres secciones principales: configuración, interacción del usuario (invocación) e interacción del administrador (gestión).

---

1. Configuración de Conexión

El servidor TinyFaaS opera sobre un broker MQTT, centralizando las operaciones en un tópico base.

Parámetro	Valor por Defecto	Descripción
Broker	broker.emqx.io	Dirección del broker MQTT al que debe conectarse.
Puerto	1883	Puerto estándar de conexión MQTT.
Tópico Base (Request)	faas	Prefijo para todos los comandos y peticiones.
Tópico Respuesta (Response)	faas/response	Prefijo para todos los resultados y errores devueltos por el servidor.

Autenticación	*(No utilizada por defecto)*	El servidor soporta opcionalmente MQTT_USERNAME y MQTT_PASSWORD.

---

2. Interacción del Usuario: Invocación de Funciones

La función principal del usuario es ejecutar código cargado previamente en el servidor. Esto se realiza mediante el tópico faas/invoke.

2.1. Invocación de una Función

Para ejecutar una función, se debe **publicar** un mensaje en el tópico específico de la función, incluyendo los argumentos necesarios en el *payload* JSON.

Acción	Tópico de Publicación (Request)	Payload (JSON)
Invocar	faas/invoke/{func_name}	{"args": [arg1, arg2, ...], "request_id": "opcional_uuid"}

- **func_name**: El nombre de la función a ejecutar (ej. calculadora).
- **args**: Una lista de valores que se pasarán a la función **main** dentro del código cargado.
- **request_id (Opcional)**: Un identificador único para rastrear la solicitud de inicio a fin.

2.2. Recepción de la Respuesta de Invocación

El resultado de la ejecución se recibe en un tópico dedicado. La respuesta incluirá el registro completo de la ejecución, ya sea un resultado exitoso o un error.

Tópico de Suscripción (Response)	Descripción
faas/response/invoke/{func_name}	Tópico para recibir el log de la ejecución de la función solicitada.

Ejemplo de Payload de Respuesta (Éxito):

JSON

```
{
   "id": "un_uuid_generado",
   "args": [10, 20],
   "result": 30,
   "status": "success",
   "time_start": "2025-10-09 12:40:00.123456",
   "time_end": "2025-10-09 12:40:00.678901",
   "request_id": "opcional_uuid"
}
```

---

3. Interacción del Administrador: Comandos de Gestión

Los administradores utilizan el tópico faas/admin/{comando} para administrar funciones, monitorear el sistema y realizar tareas de mantenimiento.

Resumen de Comandos Administrativos

Acción	Tópico de Publicación (Request)	Tópico de Suscripción (Response)

Cargar	faas/admin/upload/{func_name}	faas/response/admin/upload/{func_name}
Listar	faas/admin/list	faas/response/admin/list
Estado	faas/admin/status	faas/response/admin/status
Logs	faas/admin/logs/{func_name}	faas/response/admin/logs/{func_name}
Eliminar	faas/admin/delete/{func_name}	faas/response/admin/delete/{func_name}

3.1. Carga de una Nueva Función (Upload)

La carga de código requiere que el archivo de código (func.py) y opcionalmente los requerimientos (requirements.txt) se codifiquen en **Base64** antes de ser enviados en el *payload*.

Tópico de Publicación (Request)	Payload (JSON)
faas/admin/upload/{func_name}	{"code_b64": "...", "req_b64": "...", "request_id": "..."}

- **code_b64 (Obligatorio)**: El contenido del código de la función en formato Base64.
- **req_b64 (Opcional)**: El contenido del archivo de requerimientos en formato Base64.

Ejemplo de Request (Payload):

JSON

{
  "code_b64":
"ZGVmIG1haW4obmFtZSk6CiAgICByZXR1cm4gZiJIb2xhLCB7bmFtZX0gZGVzZGUgVGlueUZhQVMhIg==",
  "request_id": "upload_001"
}

Ejemplo de Response: {"status": "ok", "function": "saludo_simple", "request_id": "upload_001"}

3.2. Listado de Funciones

Se utiliza para obtener una lista de todos los nombres de las funciones cargadas.

Tópico de Publicación (Request)	Payload (JSON)
faas/admin/list	{}

Ejemplo de Response: {"functions": ["func1", "func2", "saludo_simple"], "request_id": "..."}

3.3. Obtener el Estado del Servidor (Status)

Proporciona información sobre el estado del servidor TinyFaaS, incluyendo el uso de recursos.

Tópico de Publicación (Request)	Payload (JSON)
faas/admin/status	{}

Ejemplo de Response (Fragmento):

JSON

```
{
   "status": "running",
   "cpu_usage_absolute": {"process_milicpu": "12.345678"},
   "memory_usage_absolute": {"process_rss_mb": "50.15 MB"},
   "timestamp": "2025-10-09 12:40:00",
   "request_id": "..."
}
```

3.4. Obtener los Logs de una Función

Permite recuperar el historial de ejecuciones (logs) persistidos para una función específica.

Tópico de Publicación (Request)	Payload (JSON)
faas/admin/logs/{func_name}	{}

Ejemplo de Response (Fragmento):

{"logs": [{"id": "...", "status": "success", ...}, {"id": "...", "status": "error", ...}], "request_id": "..."}

3.5. Eliminar una Función

Elimina la función del registro interno, borra sus logs y elimina el directorio de la función (incluyendo el código y el entorno virtual).

Tópico de Publicación (Request)	Payload (JSON)
faas/admin/delete/{func_name}	{}

Ejemplo de Response: {"status": "deleted", "function": "func_a_borrar", "request_id": "..."}

---

4. Manejo de Errores

Si se produce un error en el procesamiento de cualquier comando (invocación o administración), el servidor publicará un mensaje de error detallado en el tópico faas/response/error.

Tópico de Suscripción (Error)	Descripción
faas/response/error	Recibe mensajes de error para cualquier comando o invocación fallida.

Ejemplo de Payload de Error:

JSON
{"error": "Function not found", "topic": "faas/invoke/funcion_inexistente", "command": "funcion_inexistente"}