Async APIs in funcX

Kir Nagaitsev

Basic Demo Function

```python
import time
from funcx.sdk.client import FuncXClient
from funcx.utils.errors import TaskPending

def double_delayed(x):
   import time
   # simulate a function that takes a bit of time
   time.sleep(1)
   return x * 2

fxc = FuncXClient()

# tutorial endpoint
ep_id = '4b116d3c-1703-4f8f-9f6f-39921e5864df'
func_id = fxc.register_function(double_delayed)
```

Existing funcX Model

```python
x = 50
task_id = fxc.run(x, endpoint_id=ep_id, function_id=func_id)

while True:
   try:
       # HTTP task query
       result = fxc.get_result(task_id)
       print(result)
       break
   except TaskPending:
       # task is still pending, continue waiting
       print('Task pending')
   time.sleep(1)
```

Existing funcX Model

```
x = 50
task_id = fxc.run(x, endpoint_id=ep_id, function_id=func_id)

while True:
    try:
        # HTTP task query
        result = fxc.get_result(task_id)
        print(result)
        break
    except TaskPending:
        # task is still pending, continue waiting
        print('Task pending')
    time.sleep(1)
```

Expected Output:

Existing funcX Model

```
x = 50
task_id = fxc.run(x, endpoint_id=ep_id, function_id=func_id)

while True:
    try:
        # HTTP task query
        result = fxc.get_result(task_id)
        print(result)
        break
    except TaskPending:
        # task is still pending, continue waiting
        print('Task pending')
    time.sleep(1)
```

Expected Output:

```
Task pending
```

Existing funcX Model

```
x = 50
task_id = fxc.run(x, endpoint_id=ep_id, function_id=func_id)

while True:
    try:
        # HTTP task query
        result = fxc.get_result(task_id)
        print(result)
        break
    except TaskPending:
        # task is still pending, continue waiting
        print('Task pending')
    time.sleep(1)
```

```
Expected Output:

Task pending

100
```

Async API introduces WebSockets under the hood!

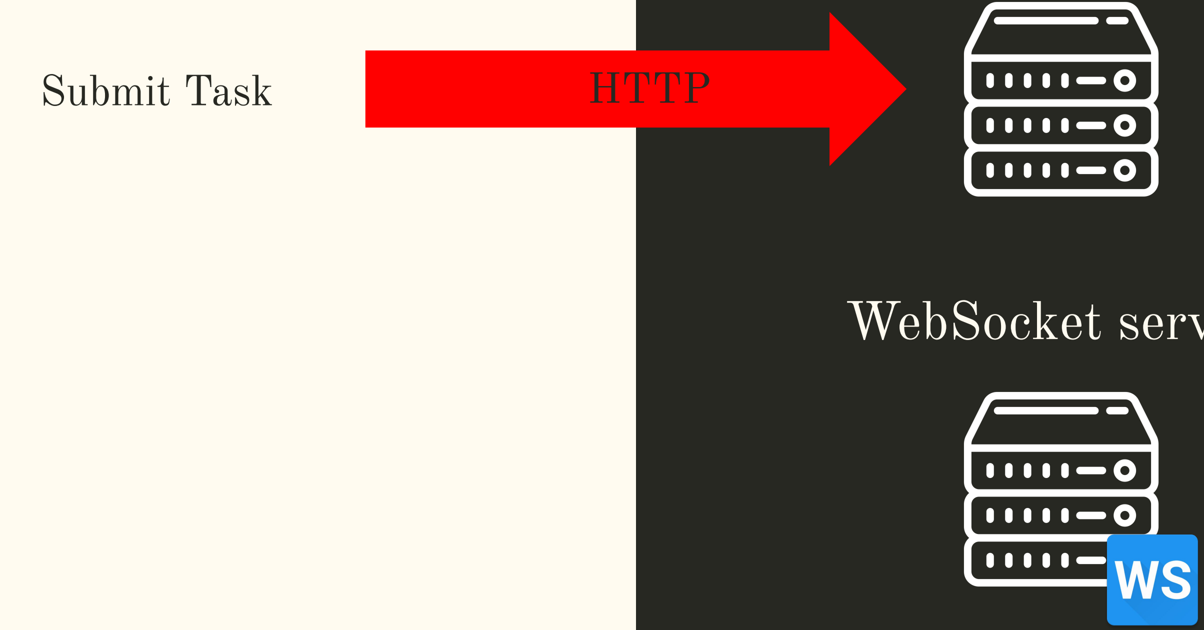
funcX SDK
(Async API model)

Submit Task

funcX web service

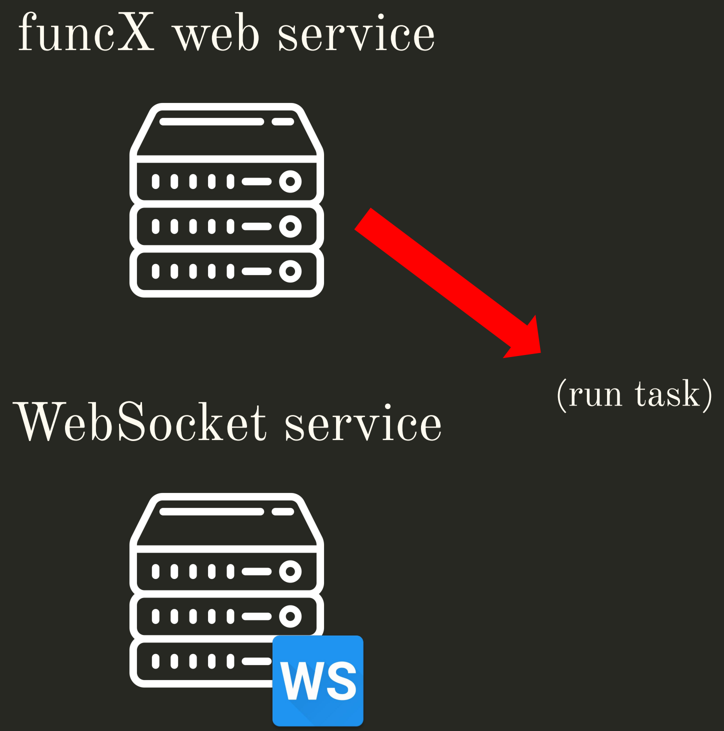
WebSocket service

funcX SDK (Async API model)



funcX SDK (Async API model)

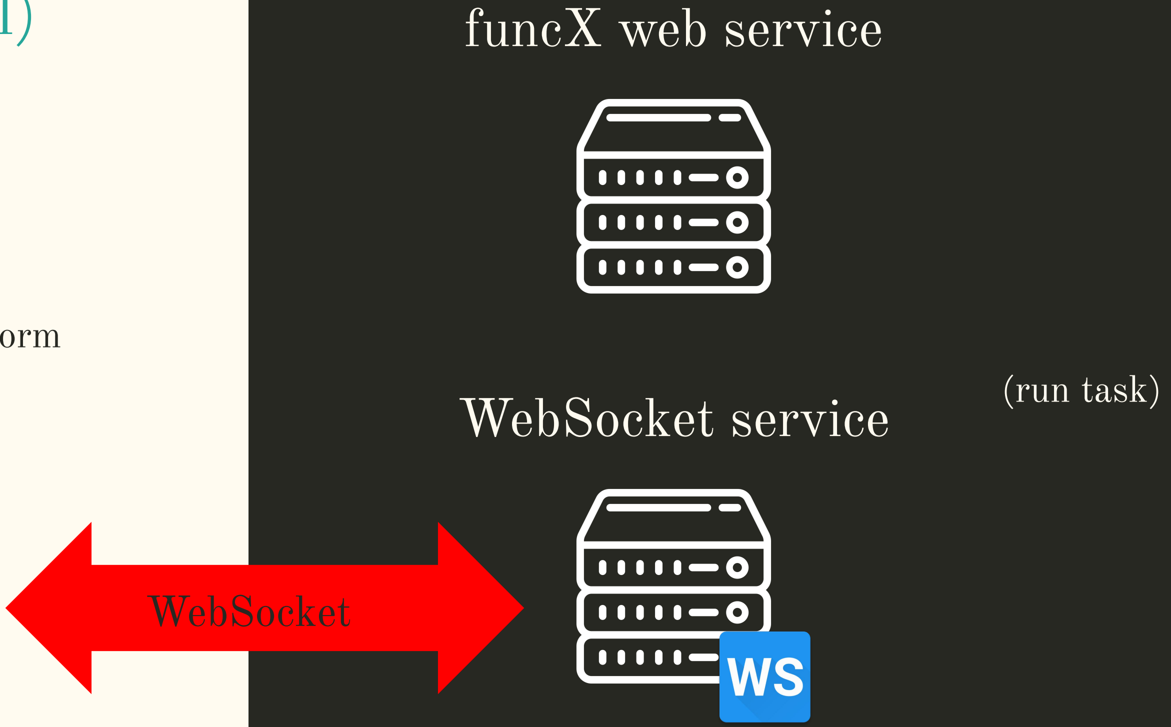
Submit Task



funcX SDK (Async API model)

Submit Task

Internally: Outstanding tasks? Form WebSocket connection

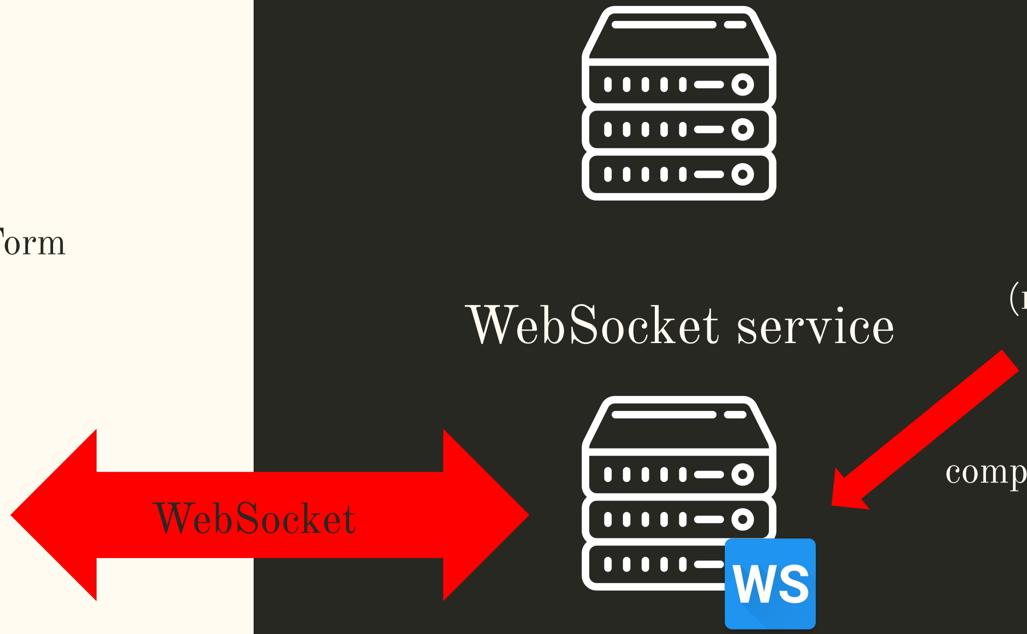


funcX SDK (Async API model)

Submit Task

Internally: Outstanding tasks? Form WebSocket connection

funcX web service



funcX SDK (Async API model)

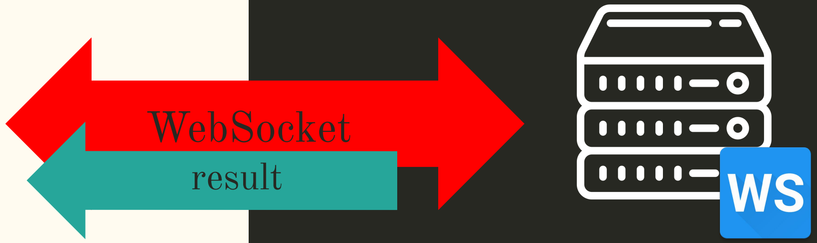
Submit Task

Internally: Outstanding tasks? Form WebSocket connection

Get Result

funcX web service

WebSocket service

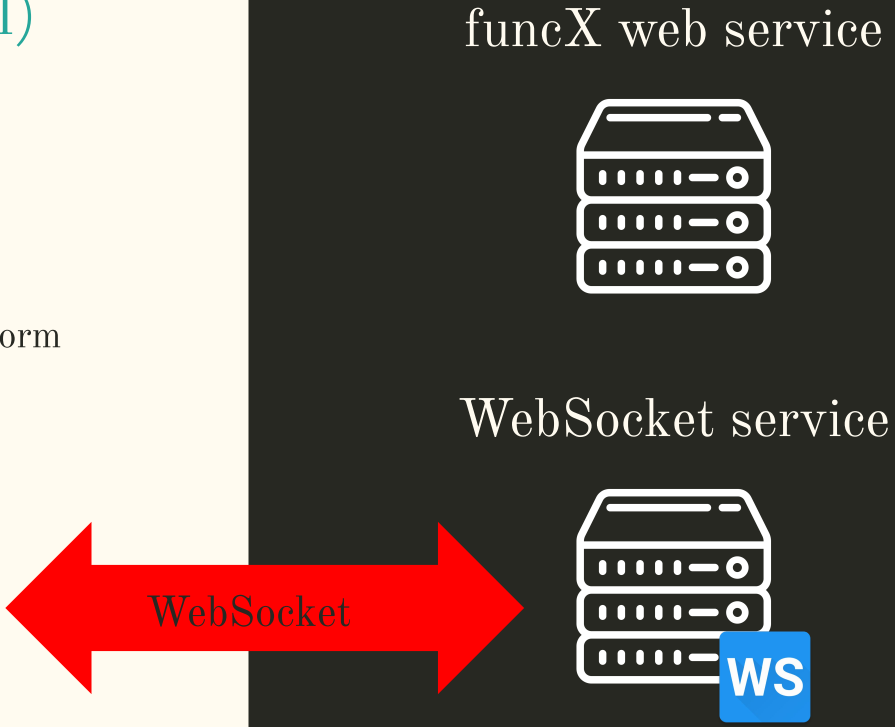


funcX SDK (Async API model)

Submit Task

Internally: Outstanding tasks? Form WebSocket connection

Get Result



funcX SDK (Async API model)

Submit Task

Internally: Outstanding tasks? Form WebSocket connection

Get Result

Internally: No more outstanding tasks? Close WebSocket connection

funcX web service

WebSocket service

Basic Async API Example

```python
from funcx.sdk.client import FuncXClient
from double_delayed import double_delayed

fxc = FuncXClient(asynchronous=True)
# tutorial endpoint
ep_id = '4b116d3c-1703-4f8f-9f6f-39921e5864df'
func_id = fxc.register_function(double_delayed)

async def task():
    x = 50
    result = await fxc.run(x, endpoint_id=ep_id, function_id=func_id)
    print(result)

fxc.loop.run_until_complete(task())
# If running in Jupyter notebook, just do: await task()
```

Basic Async API Example

```python
from funcx.sdk.client import FuncXClient
from double_delayed import double_delayed

fxc = FuncXClient(asynchronous=True)
# tutorial endpoint
ep_id = '4b116d3c-1703-4f8f-9f6f-39921e5864df'
func_id = fxc.register_function(double_delayed)

async def task():
    x = 50
    result = await fxc.run(x, endpoint_id=ep_id, function_id=func_id)
    print(result)

fxc.loop.run_until_complete(task())
# If running in Jupyter notebook, just do: await task()
```

```
Expected Output:
```

Basic Async API Example

```python
from funcx.sdk.client import FuncXClient
from double_delayed import double_delayed

fxc = FuncXClient(asynchronous=True)
# tutorial endpoint
ep_id = '4b116d3c-1703-4f8f-9f6f-39921e5864df'
func_id = fxc.register_function(double_delayed)

async def task():
   x = 50
   result = await fxc.run(x, endpoint_id=ep_id, function_id=func_id)
   print(result)

fxc.loop.run_until_complete(task())
# If running in Jupyter notebook, just do: await task()
```

```
Expected Output:

100
```

funcX Executor API is built on top of async interface

FuncXExecutor

```
class FuncXExecutor(concurrent.futures.Executor):

   def submit(self, function, *args, endpoint_id=None, container_uuid=None, **kwargs):
       ...

   (Runs async WebSocket code on a separate thread under the hood)
```

Executor Example

```python
from funcx import FuncXClient
from funcx.sdk.executor import FuncXExecutor
...

fxc = FuncXClient()
fx = FuncXExecutor(fxc)

# tutorial endpoint
endpoint_id = '4b116d3c-1703-4f8f-9f6f-39921e5864df'

x = 50
future = fx.submit(double_delayed, x, endpoint_id=endpoint_id)
result = future.result()
print(result)
```

Executor Example

```
from funcx import FuncXClient
from funcx.sdk.executor import FuncXExecutor
...

fxc = FuncXClient()
fx = FuncXExecutor(fxc)

# tutorial endpoint
endpoint_id = '4b116d3c-1703-4f8f-9f6f-39921e5864df'

x = 50
future = fx.submit(double_delayed, x, endpoint_id=endpoint_id)
result = future.result()
print(result)
```

Expected Output:

Executor Example

```
from funcx import FuncXClient
from funcx.sdk.executor import FuncXExecutor
...

fxc = FuncXClient()
fx = FuncXExecutor(fxc)

# tutorial endpoint
endpoint_id = '4b116d3c-1703-4f8f-9f6f-39921e5864df'

x = 50
future = fx.submit(double_delayed, x, endpoint_id=endpoint_id)
result = future.result()
print(result)
```

Expected Output:

```
100
```

Executor Batching (Next SDK Release)

```
fxc = FuncXClient()
fx = FuncXExecutor(fxc, batch_enabled=True, batch_interval=1.0)
def run():
    futures = []
    for _ in range(50):
        x = random.randint(0, 100)
        future = fx.submit(double_delayed, x, endpoint_id=ep_id)
        futures.append(future)

    for future in futures:
        result = future.result()
        print(f'Result: {result}')

t = timeit.timeit(run, number=1)
print(f'Time: {round(t, 2)}s')
```

Executor Batching (Next SDK Release)

```
fxc = FuncXClient()
fx = FuncXExecutor(fxc, batch_enabled=True, batch_interval=1.0)
def run():
   futures = []
   for _ in range(50):
       x = random.randint(0, 100)
       future = fx.submit(double_delayed, x, endpoint_id=ep_id)
       futures.append(future)

   for future in futures:
       result = future.result()
       print(f'Result: {result}')

t = timeit.timeit(run, number=1)
print(f'Time: {round(t, 2)}s')
```

```
Expected Output:
```

Executor Batching (Next SDK Release)

```
fxc = FuncXClient()
fx = FuncXExecutor(fxc, batch_enabled=True, batch_interval=1.0)
def run():
   futures = []
   for _ in range(50):
       x = random.randint(0, 100)
       future = fx.submit(double_delayed, x, endpoint_id=ep_id)
       futures.append(future)

   for future in futures:
       result = future.result()
       print(f'Result: {result}')

t = timeit.timeit(run, number=1)
print(f'Time: {round(t, 2)}s')
```

```
Expected Output:

<Results>
```

Executor Batching (Next SDK Release)

```
fxc = FuncXClient()
fx = FuncXExecutor(fxc, batch_enabled=True, batch_interval=1.0)
def run():
    futures = []
    for _ in range(50):
        x = random.randint(0, 100)
        future = fx.submit(double_delayed, x, endpoint_id=ep_id)
        futures.append(future)

    for future in futures:
        result = future.result()
        print(f'Result: {result}')

t = timeit.timeit(run, number=1)
print(f'Time: {round(t, 2)}s')
```

```
Expected Output:

<Results>

Time: 8.47s
```

What’s Next?

- Robustness improvements: WebSocket connection loss, recovering tasks that were submitted but not received
- Task cancellation

Summary

- Existing HTTP query model works fine for fire-and-forget usage or long running tasks
- Async API is better for speed and complex async use-cases
- FuncX Executor is equally good for speed (thin layer built on top of async interface) and more user-friendly
- Both Async API and Executor allow you to forget about task_id
- Slides: https://github.com/Loonride/funcx-async-parslfest-2021

Questions?