DISTRIBUTED AND CLOUD COMPUTING

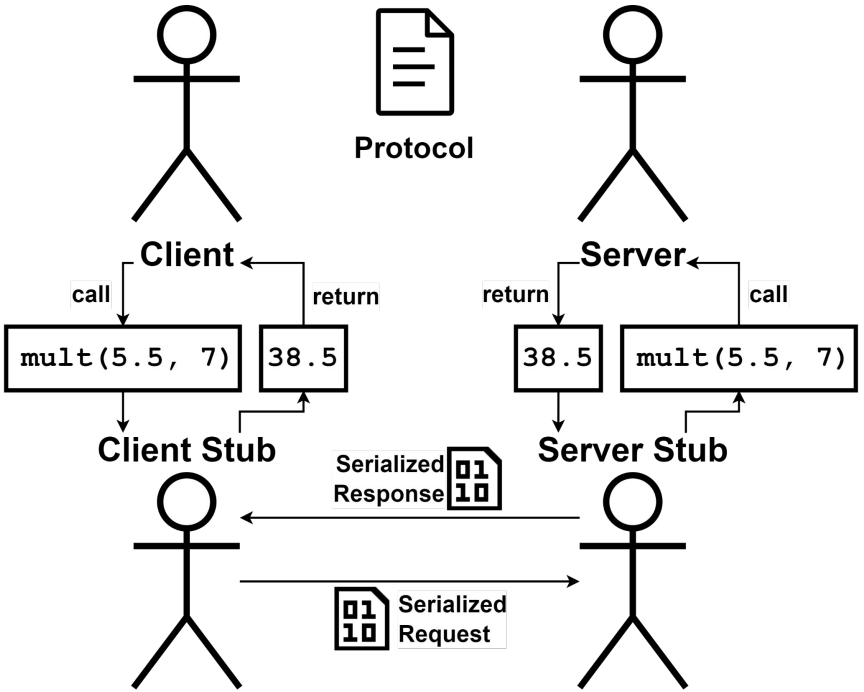
LAB 8: REVERSE PROXY + OTHER API ARCHITECTURES

(Module: RPC & RESTFUL API)

RPC Stub as A Proxy

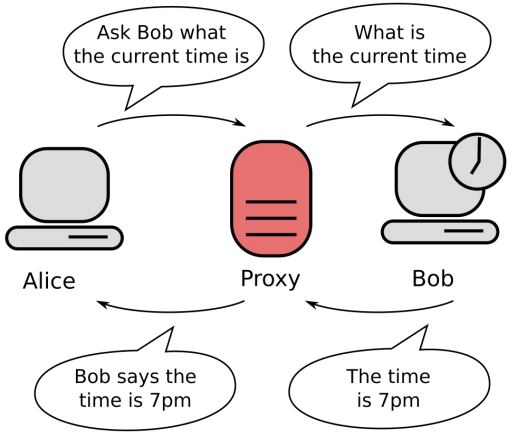
- Client Stub
  a. Handles serialization / deserialization of data for the client.
  b. Handles remote server connection and data transmission for the client.
- Server Stub
  a. Works similarly, but for the server.

- RPC Stubs act like **proxies**.
- The concept of a proxy is broad, but in computer networking, a **forward proxy** and a **reverse proxy** each have specific, distinct functions.



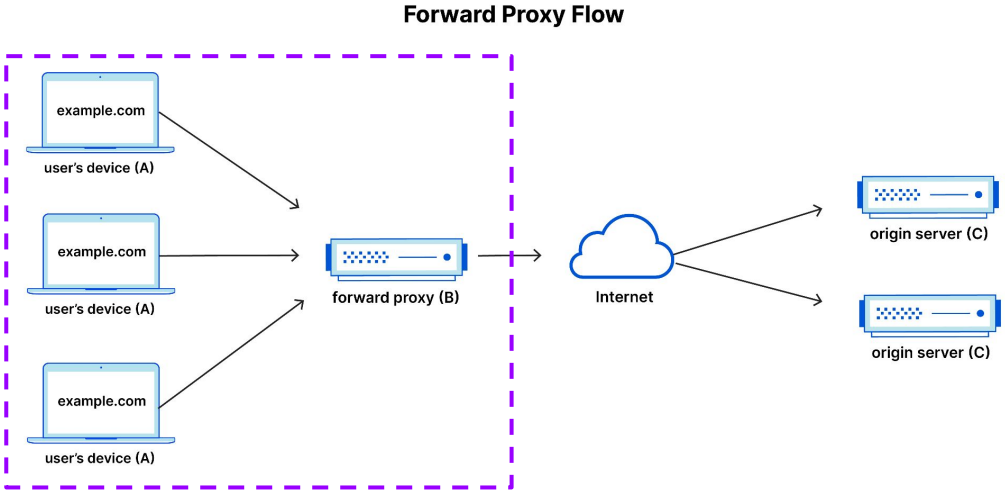
Forward Proxy

- “A forward proxy, often called a proxy, proxy server, or web proxy, is a server that sits in front of a group of client machines. When those computers make requests to sites and services on the Internet, the proxy server intercepts those requests and then communicates with web servers on behalf of those clients, like a middleman.”



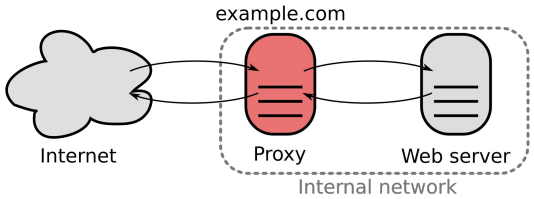
Why Proxy?

1. Access Control & Security
2. Privacy & Anonymity
3. Content Caching & Filtering
4. Monitoring & Logging
5. Transparency



https://en.wikipedia.org/wiki/Proxy_server
https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/

Reverse Proxy

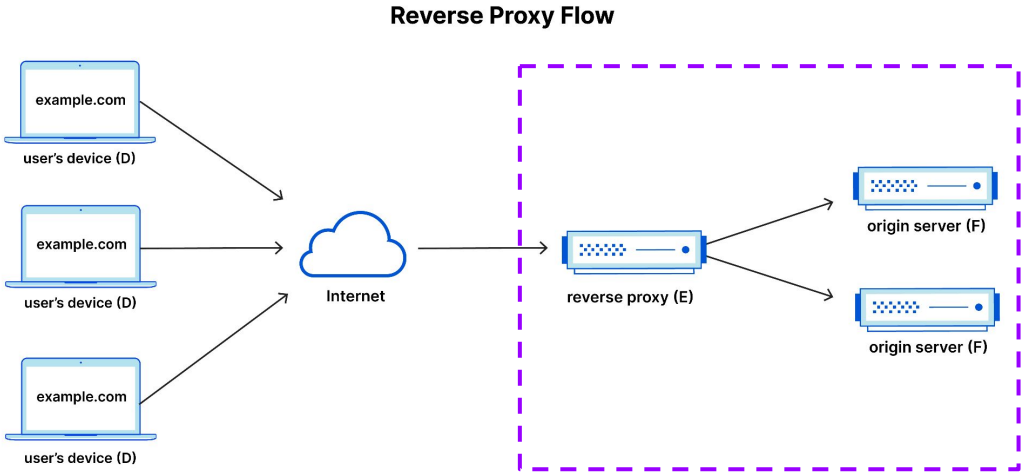


- "A reverse proxy is a server that sits in front of one or more web servers, intercepting requests from clients."
- It works for the servers compared to the forward proxy.
- Forward & Reverse Proxy are coexisting twins.
- Example: Load Balancers, CDNs, API Gateways, Web Application Firewalls (WAFs), etc.

Why Reverse Proxy?

1. **Load Balancing**
2. Access Control & Security
3. Content Caching & Filtering
4. Monitoring & Logging
5. SSL Encryption Offloading

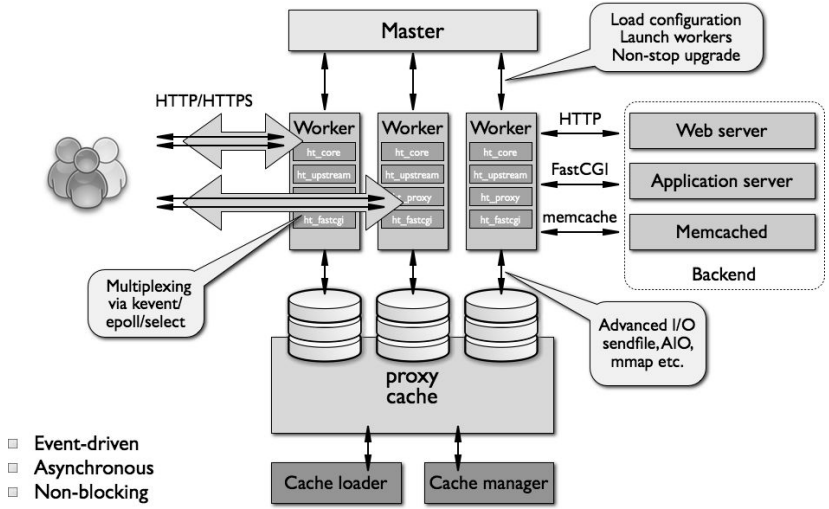
DDos attack



https://en.wikipedia.org/wiki/Proxy_server
https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/

Nginx as A Reverse Proxy

- “nginx ("engine x") is an HTTP web server, reverse proxy, content cache, load balancer, TCP/UDP proxy server, and mail proxy server.”
- one of the most popular tools for **load balancing** purpose
- simple configuration file format
- clean support with Docker



nginx.conf

rpc_rest > 2_others > 0_reverse_proxy > nginx > nginx.conf

```
# https://nginx.org/en/docs/http/load_balancing.html
# placeholder: event handlers
events {}

# handle HTTP requests
http {
  # define a group of backend servers that will handle requests
  upstream flask_servers {
    # Default load balancing method is round-robin.
    # To test other methods, uncomment one of the following:
    # least_conn;   # forward to the server with the least number o
    # ip_hash;      # same ip-hash to the same server (session main
    server flask_server1:5000;
    server flask_server2:5000;
    server flask_server3:5000;
  }

  # virtual server config
  server {
    listen 80;

    # process requests matching the URI pattern (`/` means root + e
    location / {
      # forward requests to the upstream group
      proxy_pass http://flask_servers;
      # note back info of the original client
      # $host: https://nginx.org/en/docs/http/ngx_http_core_module.
      # $remote_addr: https://nginx.org/en/docs/http/ngx_http_core_
      # $proxy_add_x_forwarded_for: https://nginx.org/en/docs/http/
      # $scheme: https://nginx.org/en/docs/http/ngx_http_core_modul
      # X-Forwarded-For records a list of hopped IPs, compared to X
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
    }
  }
}
```

https://nginx.org
https://aosabook.org/en/v2/nginx.html

If the “docker build” failed to retrieve image metadata, try pulling in advance: **docker pull python:3-slim**

TASK: Reverse Proxy - Nginx

Load balance requests to 3 identical Flask servers.

> Reference codebase: `revproxy_nginx`

1. Set up Python (Miniconda is recommended).
2. Check `flask_app/app.py` for the Flask server implementation. It contains 1 simple API at the base URL that returns a Hello World message marking the server’s id/name.
3. Check `flask_app/Dockerfile` for the Flask server image specification. Build the image:
   - `docker build -t flask_app .`

TIPS: **docker image prune** to clean previous builds.

app.py

rpc_rest > 2_others > 0_reverse_proxy > flask_app > app.py > ...

```python
from flask import Flask
import os

app = Flask(__name__)

@app.route('/')
def hello():
    server_name = os.getenv('SERVER_NAME', 'Unknown Server')
    return f'Hello from {server_name}!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=int(os.getenv('FLASK_PORT', 5000)))
```

Dockerfile

rpc_rest > 2_others > 0_reverse_proxy > flask_app > Dockerfile > ...

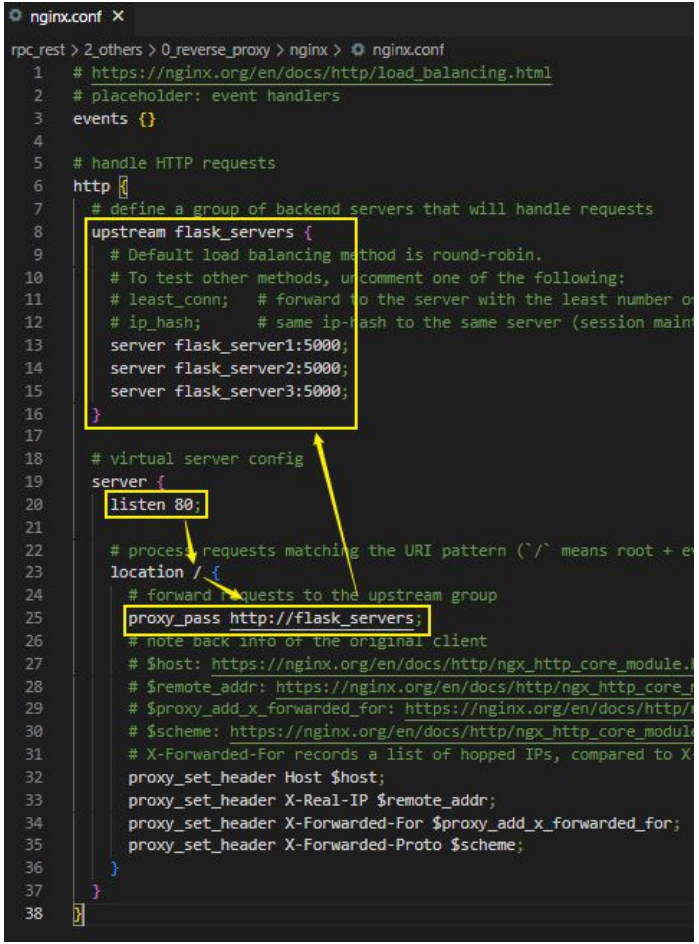
```
FROM python:3-slim

WORKDIR /app

COPY requirements.txt app.py ./
RUN pip3 install --no-cache-dir -r requirements.txt

CMD [ "python", "app.py" ]
```

TASK: Reverse Proxy - Nginx

Load balance requests to 3 identical Flask servers.

> Reference codebase: `revproxy_nginx`

4. Check `nginx/nginx.conf` that configures how nginx server intercepts & forwards the requests.
   a. Listen for new requests on port `80`.
   b. Process requests matching `/` pattern only.
   c. Forward requests to upstream server group.

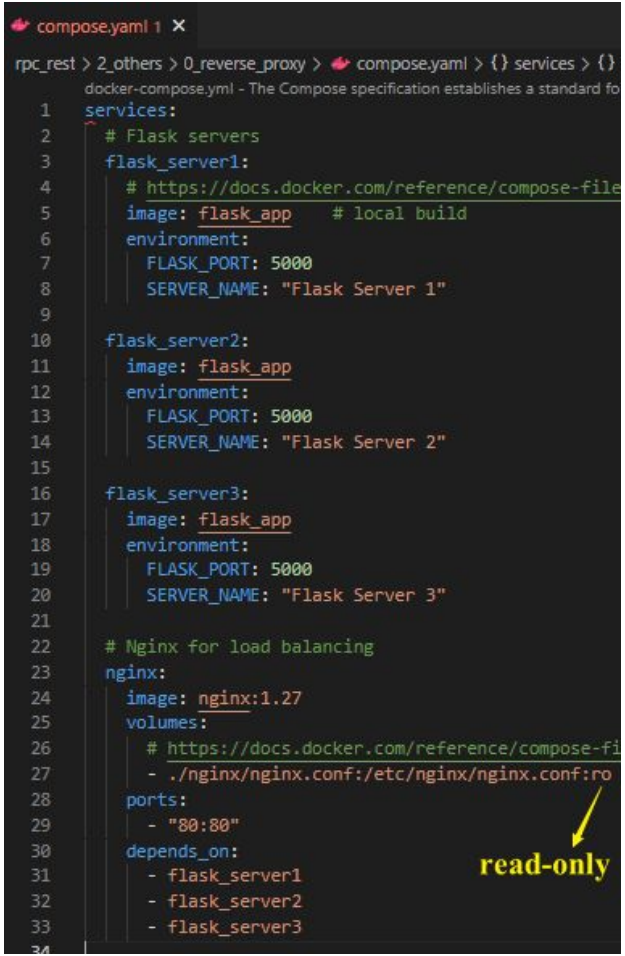


TASK: Reverse Proxy - Nginx

Load balance requests to 3 identical Flask servers.

> Reference codebase: `revproxy_nginx`

4. Check `nginx/nginx.conf` that configures how nginx server intercepts & forwards the requests.
   a. Listen for new requests on port `80`.
   b. Process requests matching `/` pattern only.
   c. Forward requests to upstream server group.
5. Check `compose.yaml` for the defined services.
   a. 3 Flask servers serve at port `5000`, assigned with a server name.
   b. 1 nginx server loads the `nginx.conf` file, serving at port `80` which is exposed to localhost.
6. Start the environment:
   a. `docker compose up -d`



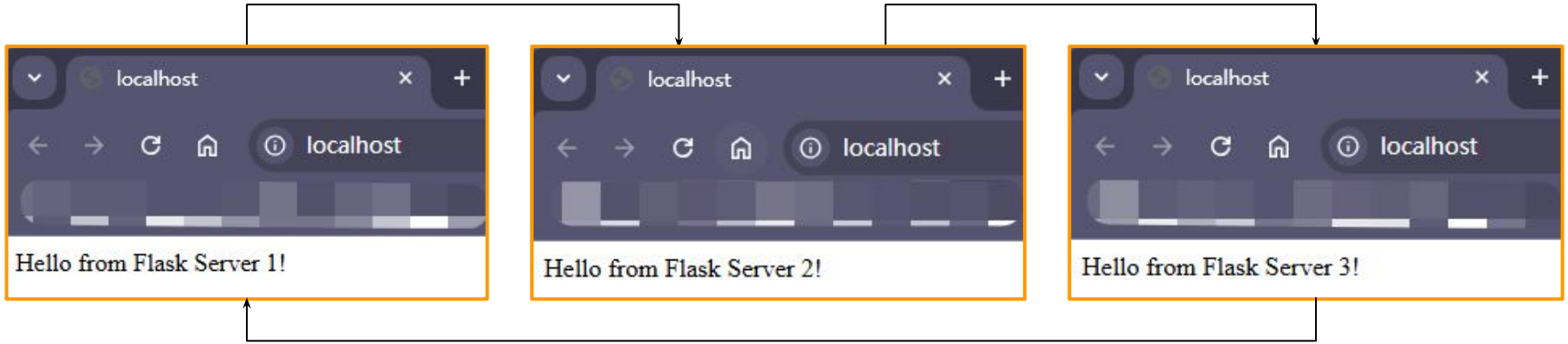
TASK: Reverse Proxy - Nginx

Load balance requests to 3 identical Flask servers.

> Reference codebase: `revproxy_nginx`

7. Start a web browser and access http://localhost:80/. Refresh the page multiple times. Note how the requests are handled by different Flask servers in a round-robin fashion (default load balancing approach of nginx).
8. Optionally, test with Postman or cURL.
   a. e.g., `curl localhost`

```
docker compose up -d
[+] Running 6/6
 ✔ Network 0_reverse_proxy_default                Created
 ✔ Container 0_reverse_proxy-flask_server3-1      Started
 ✔ Container 0_reverse_proxy-flask_server2-1      Started
 ✔ Container 0_reverse_proxy-flask_server1-1      Started
 ✔ Container 0_reverse_proxy-nginx-1              Started
```



TASK: Reverse Proxy - Nginx

Load balance requests to 3 identical Flask servers.

> Reference codebase: `revproxy_nginx`

9. Check `test/clients.py` where 1000 requests are sent concurrently to the nginx server. Then, a map is constructed to record how many requests are handled in each Flask server. Run:
   - `python test/clients.py`

```
python test/clients.py
{1: 333, 2: 333, 3: 334}
```

```
python test/clients.py
0: Hello from Flask Server 1!
1: Hello from Flask Server 3!
2: Hello from Flask Server 2!
3: Hello from Flask Server 2!
8: Hello from Flask Server 3!
5: Hello from Flask Server 1!
4: Hello from Flask Server 3!
7: Hello from Flask Server 1!
6: Hello from Flask Server 2!
9: Hello from Flask Server 1!
{1: 4, 2: 3, 3: 3}
```

10 concurrent requests

clients.py

rpc_rest > 2_others > 0_reverse_proxy > test > clients.py > ...

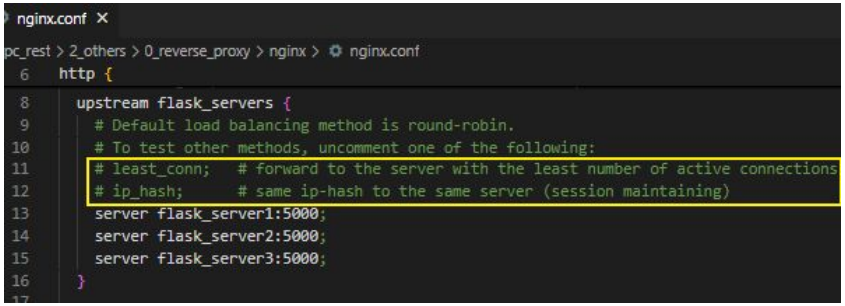
```python
import sys
from concurrent.futures import ThreadPoolExecutor, as_completed
from pprint import pprint

import requests

def send_request(id, url):
  try:
    response = requests.get(url)
    msg = response.text.strip()
    server_id = int(msg.rsplit(' ', maxsplit=1)[-1][:-1])
  except requests.RequestException as e:
    msg = str(e)
    server_id = -1
  return server_id, f'{id}: {msg}'

def test_load_balancing(url, n_requests):
  # server_id => cnt
  res_dict = {}
  with ThreadPoolExecutor(max_workers=n_requests) as executor:
    res_futures = [executor.submit(send_request, i, url) for i in range(n_requests)]
    for res_future in as_completed(res_futures):
      res_server_id, res_msg = res_future.result()
      # statistics
      if res_server_id not in res_dict:
        res_dict[res_server_id] = 0
      res_dict[res_server_id] += 1
      # sys.stdout.write(f'{res_msg}\n')
      # sys.stdout.flush()
  pprint(res_dict)

if __name__ == '__main__':
  test_load_balancing(url='http://localhost:80/', n_requests=1000)
```

TASK: Reverse Proxy - Nginx

Load balance requests to 3 identical Flask servers.

> Reference codebase: `revproxy_nginx`

9. Check `test/clients.py` where 1000 requests are sent concurrently to the nginx server. Then, a map is constructed to record how many requests are handled in each Flask server. Run:
   - `python test/clients.py`
10. Switch to other load balancing methods by modifying `nginx.conf`. Restart the nginx server container manually to refresh and run the test script again.
11. Clean up the resources via:
    - `docker compose down`




```
python test/clients.py
{1: 340, 2: 333, 3: 327}
```

least_conn

```
python test/clients.py
{1: 1000}
```

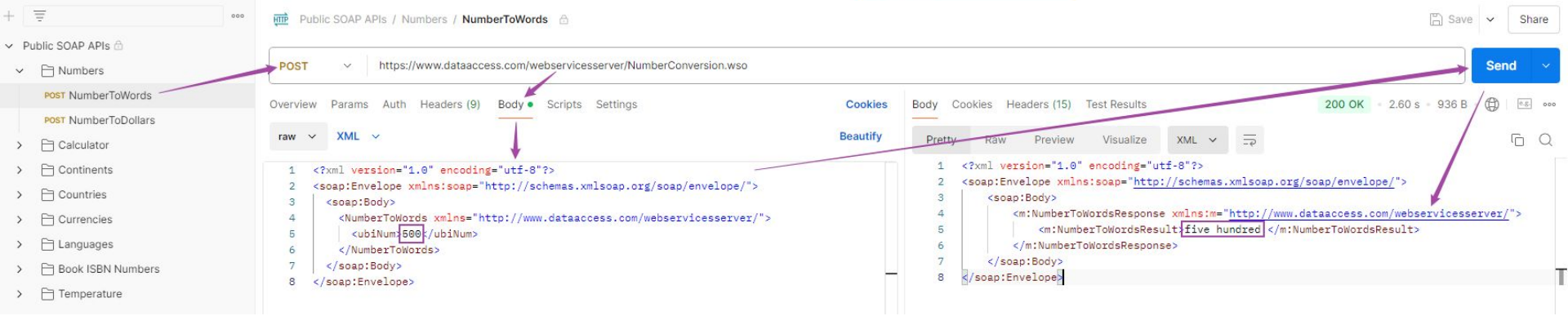
ip_hash

SOAP

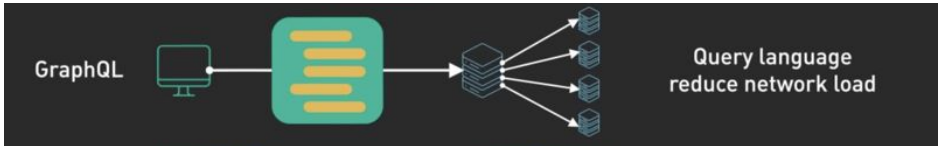
- Use XML messages + a predefined contract
- Strict, complex and verbose

- Play with a Postman workspace about SOAP.
- Check its WSDL definition by appending `?WSDL` to the service URL, or click this link.

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:soap="http:
 name="NumberConversion" targetNamespace="http://www.dataaccess.com/webservicesserver/">
  <types>
  ...
  </types>                                                    data schema
  <message name="NumberToWordsSoapRequest">
    <part name="parameters" element="tns:NumberToWords"/>
  </message>
  <message name="NumberToWordsSoapResponse">
    <part name="parameters" element="tns:NumberToWordsResponse"/>
  </message>                                                  service operation input/output message
  <message name="NumberToDollarsSoapRequest">
  ...
  </message>
  <message name="NumberToDollarsSoapResponse">
  ...
  </message>
  <portType name="NumberConversionSoapType">
  ...
  </portType>                                                 service operation
  <binding name="NumberConversionSoapBinding" type="tns:NumberConversionSoapType">
  ...
  </binding>                                                  service operation
  <binding name="NumberConversionSoapBinding12" type="tns:NumberConversionSoapType">  bindings to different
  ...                                                         SOAP versions
  </binding>
  <service name="NumberConversion">
  ...
  </service>                                                  service endpoint
</definitions>
```



GraphQL



- queries exactly needed within a single request
- suitable for complex data requirements

- Explore Countries GraphQL API.
- Check schema definition here.

```
builder.objectType(CountryRef, {
  fields: (t) => ({
    code: t.exposeID("code"),
    name: t.string({
      args: {
        lang: t.arg.string({
          validate: (lang) => langs()
        }),
      },
      resolve: async (country, { lang
      },
    }),
    native: t.exposeString("native"),
    phone: t.exposeString("phone"),
```

```
    languages: t.field({
      type: [LanguageRef],
      resolve: (country) =>
        country.languages.map((code) => ({
          ...languages[code as keyof typeof languages],
          code,
        })),
    }),
```

```
query GetCountry {
  country(code: "BR") {
    name
    native
    capital
    emoji
    currency
    languages {
      code
      name
    }
  }
}
```

Variables Headers

countries.trevorblades.com


```
{
  "data": {
    "country": {
      "name": "Brazil",
      "native": "Brasil",
      "capital": "Brasília",
      "emoji": "BR",
      "currency": "BRL",
      "languages": [
        {
          "code": "pt",
          "name": "Portuguese"
        }
      ]
    }
  }
}
```


HIT 607ms RequestID: 2382755471883397900

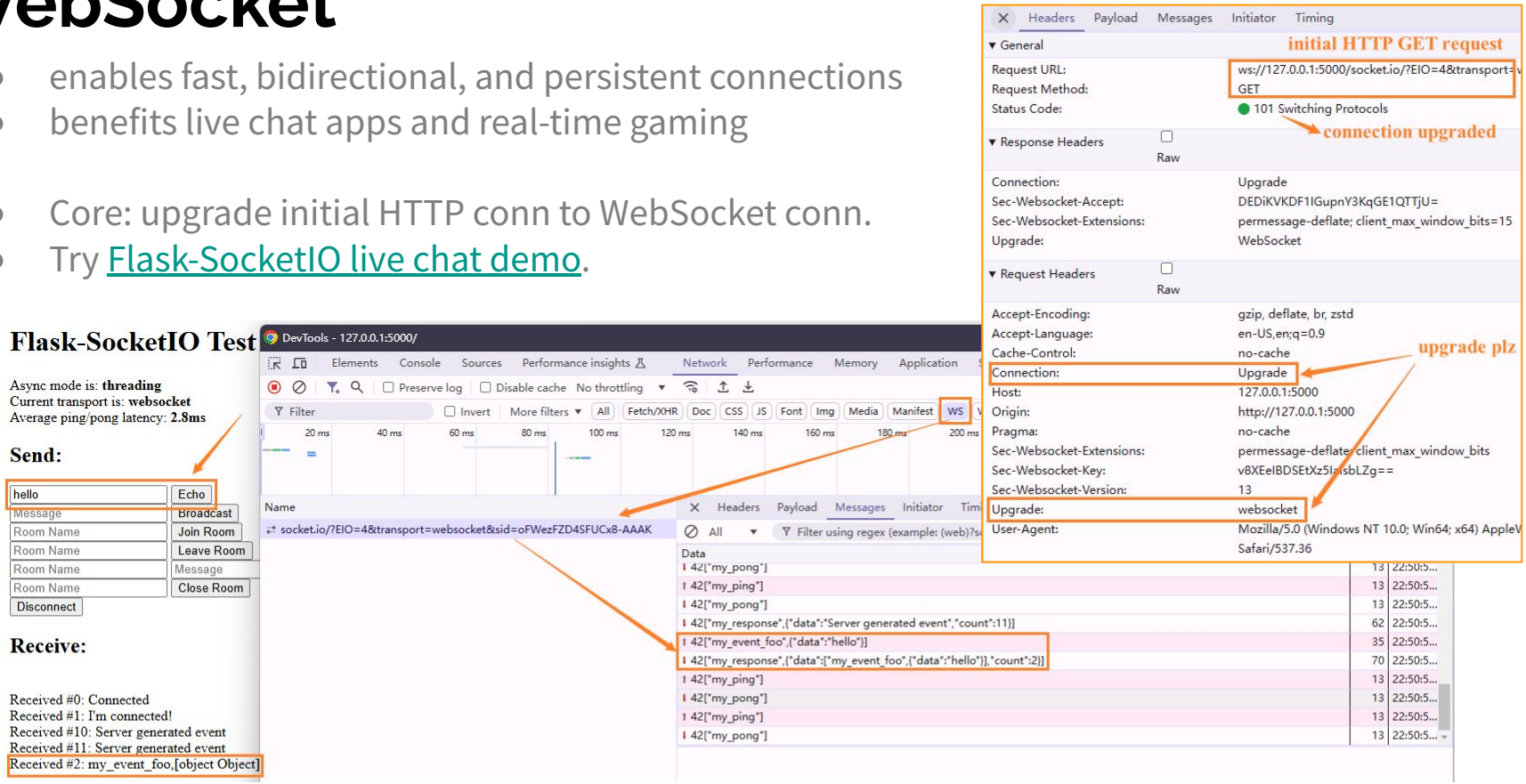
IP limit 47 (out of 50) requests remaining (refills in 14s)

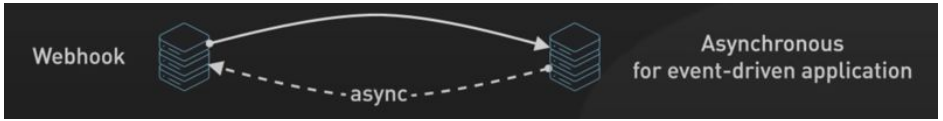
https://mannhowie.com/rest-api

WebSocket

- enables fast, bidirectional, and persistent connections
- benefits live chat apps and real-time gaming

- Core: upgrade initial HTTP conn to WebSocket conn.
- Try [Flask-SocketIO live chat demo](#).

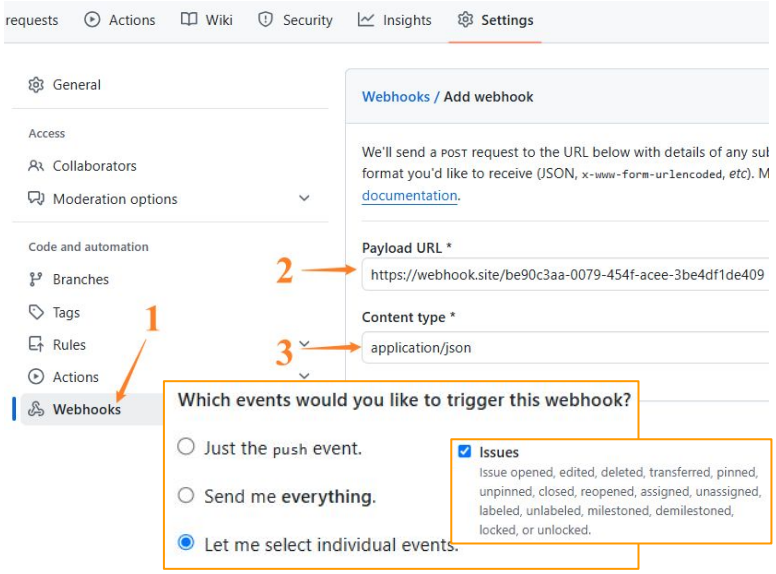


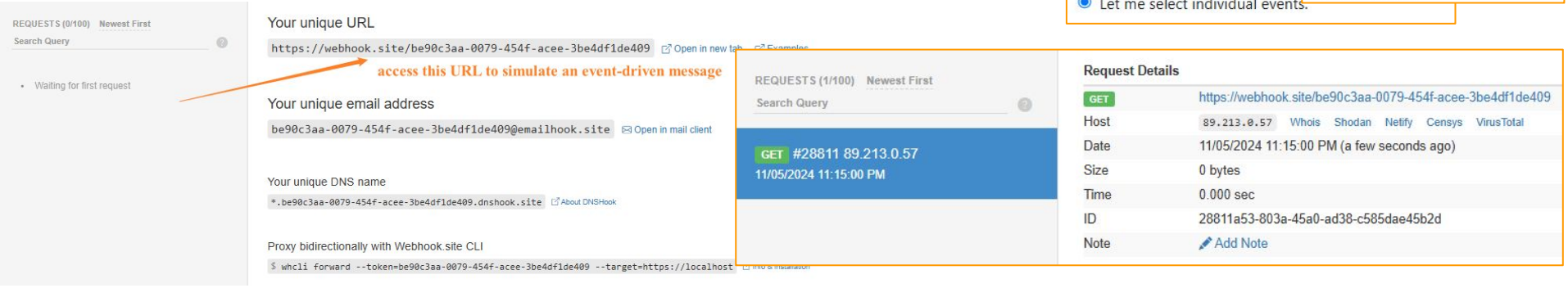


Webhook

- supports asynchronous & event-driven notifications

- Try Webhook.site.
- Optionally, add a new GitHub Webhook in a repository to trigger send messages upon issue-related events.
  - Create a new issue and do sth there. New messages will appear at webhook.site.
  - Remember to clean the GitHub Webhook later.

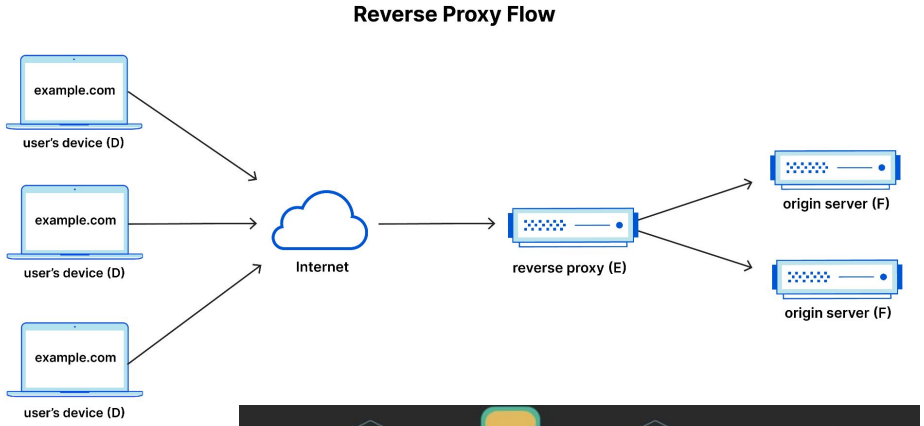


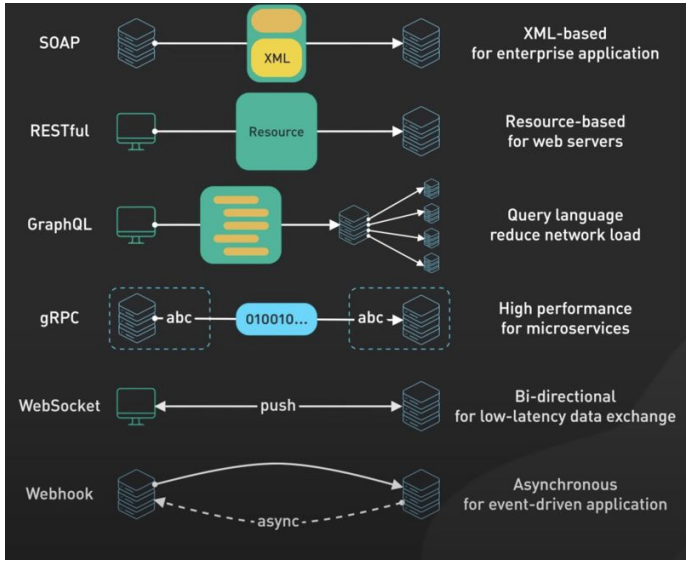


Summary

- **Reverse Proxy**
  - Reverse Proxy *vs.* Forward Proxy
  - Benefits:
    i. Load Balancing
    ii. Access Control & Security
    iii. Content Caching & Filtering
    iv. Monitoring & Logging
    v. SSL Encryption Offloading
  - Nginx as a reverse proxy
- **Other API Architectures**
  - SOAP
  - GraphQL
  - WebSocket
  - Webhook

That's all for this module!





https://mannhowie.com/rest-api
https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/