✦ Member-only story

Load Testing WebSockets With k6

Using a built-in WebSocket client to test the performance of your server

Ng Wai Foong · Follow

Published in Better Programming

6 min read · Mar 3, 2021

Listen Share ••• More

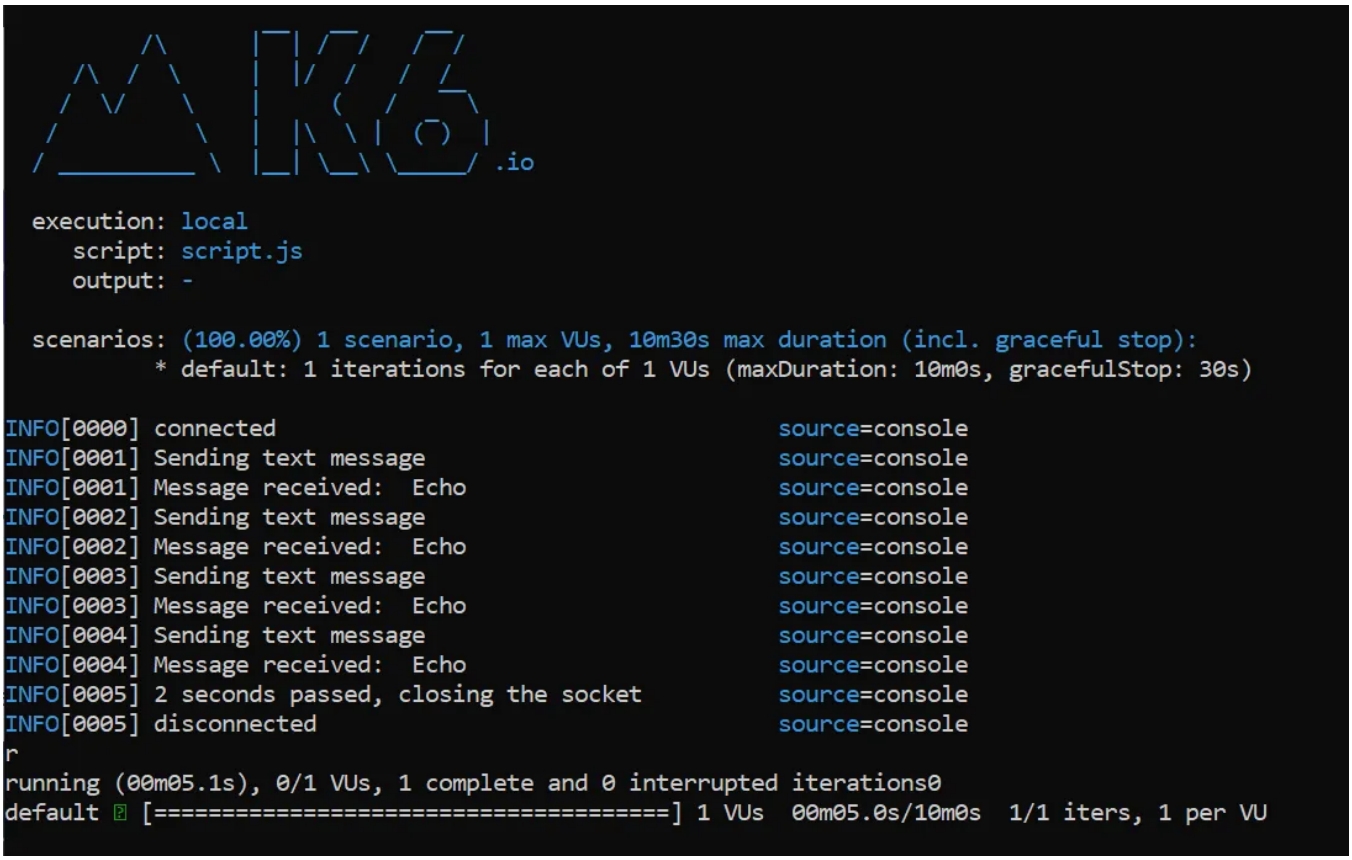


Image by the author

Building on top of my previous article on k6, the topic for this article is on load testing WebSocket. Unlike HTTP, WebSocket offers full-duplex communication

channels over a single TCP connection. This enables your server to send push notifications directly to users.

Fortunately, k6 provides its own `ws` client based on the WebSocket protocol. It's slightly different than the `http` client, as each VU runs on an asynchronous event loop.

Setup

k6 installation

Before we begin, make sure you've installed the necessary packages and modules for k6. Kindly refer to my previous article on how to install k6 in your local machine.

Using your own WebSocket server (optional)

This tutorial assumes you have an existing WebSocket server. If you do not have one, kindly use the following script, which is based on FastAPI. It will echo back the message sent by the user via WebSocket.

```python
from fastapi import FastAPI, WebSocket, WebSocketDisconnect

app = FastAPI()


@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    try:
        while True:
            data = await websocket.receive_text()
            await websocket.send_text(data)
    except WebSocketDisconnect:
        pass
```

k6-websocket-myapp.py hosted with ❤️ by GitHub view raw

You can find the installation steps at the following documentation.

Run it via the following command:

```
uvicorn myapp:app
```

Using a public WebSocket server

If you have trouble setting up FastAPI and would like to test out k6, simply replace the WebSocket URL with any of the public websocket URL.

For example, you can use the following link from javascript.info:

```
wss://javascript.info/article/websocket/demo/hello
```

The Basics

In order to use the `ws` client, you have to import it as follows:

```
import ws from 'k6/ws';
```

Then, you can call the built-in `connect` function to establish a connection to your server.

```
const url = 'ws://localhost:8000/ws';
const res = ws.connect(url, null, function (socket) {
    ...
}
```

The `connect` function accepts the following input parameters:

- `url` — the requested string URL
- `params` — object containing additional parameters
- `callback` — the callback function that will be called when the WebSocket connection is initiated. It represents the `Socket` object.

The `Socket` object comes with the following methods:

- `close` — close the connection
- `on (event, callback)` — event listener for the following: open, message, ping, pong, close, error
- `ping` — send a ping request to WebSocket server
- `send (data)` — send data to WebSocket server
- `setInterval(callback, interval)` — call a function repeatedly at a certain interval
- `setTimeout(callback, timeout)` — call a function after a delay

Let's go through them one by one with example code snippets.

close

You need to close the connection to the WebSocket server once you're done with it. You can simply call `socket.close()` to do the job.

```
const res = ws.connect(url, null, function (socket) {
  ...
  socket.close();
}
```

on(event, callback)

The `on` function enables the creation of an event listener. You can use it to log messages when a connection is established or even do a check on the data sent by the server.

```
const res = ws.connect(url, null, function (socket) {
  ...
  socket.on('open', () => console.log('connected'));
  socket.on('message', (data) => console.log('Received: ', data));
  socket.on('close', () => console.log('disconnected'));
}
```

ping

Sending a ping request is a common method used to check the health status of a server. You can do so via the following code:

```
const res = ws.connect(url, null, function (socket) {
  ...
  socket.ping();
}
```

send(data)

Sending data is as simple as calling the `send` function. It accepts an input string. For sending a JavaScript object, simply convert it to a JSON string via `JSON.stringify` before sending it.

```
const res = ws.connect(url, null, function (socket) {
  ...
  socket.send('text');
  socket.send(JSON.stringify({ data: 'hello' }));
}
```

setInterval(callback, interval)

You can execute a function repeatedly at a certain interval via `setInterval` function. For example, you can send data to the WebSocket server repeatedly once every one second as long as the connection is still opened.

```
const res = ws.connect(url, null, function (socket) {
  ...
  socket.setInterval(function interval() {
    socket.send('Hello');
  }, 1000);
}
```

`interval` is based in milliseconds. For a one-second interval, you have to use `1000` as the input parameter.

setTimeout(callback, timeout)

`setTimeout` is another useful function. It triggers a function after a delay. For example, you can use it to close the connection to a WebSocket server after a few seconds of delay.

```
const res = ws.connect(url, null, function (socket) {
  ...
  socket.setTimeout(function () {
```

```
    socket.close();
  }, 5000);
}
```

Implementation

Let's combine all the knowledge you've learned and build a simple load-testing script. Create a new JavaScript file called `k6-websocket-script.js`.

Import

Add the following import statement at the top of the file.

```
import ws from 'k6/ws';
import { check } from 'k6';
```

Options

Next, set the options to be used for testing. The test consists of the following stages:

- Start by generating new VUs to 30 in 10 seconds
- Maintain the VUs for 30 seconds
- Continue ramping up VUs to 60 in 20 seconds
- Maintain the VUs for 30 seconds
- Scale down VUs to 30 in 20 seconds
- Maintain the VUs for 30 seconds
- Slowly reduce the VUs to 0 in 10 seconds

```
export let options = {
  stages: [
    { duration: '10s', target: 30 },
    { duration: '30s', target: 30 },
    { duration: '20s', target: 60 },
    { duration: '30s', target: 60 },
    { duration: '20s', target: 30 },
    { duration: '30s', target: 30 },
    { duration: '10s', target: 0 },
```

```
  ],
};
```

Feel free to modify the number of VUs based on your own preferences.

Main function

Create a new default function with the following variables:

```
export default function () {
  const text = 'Echo';
  const url = 'ws://localhost:8000/ws';
}
```

Continue by adding the following code underneath it. It'll first establish a connection to the WebSocket server and set up an event listener. Inside the listener, it contains the `setInterval` function, which will send data to the server repeatedly once every second.

```
export default function () {
  ...
  const res = ws.connect(url, null, function (socket) {
    socket.on('open', function open() {
      console.log('connected');
      socket.setInterval(function interval() {
        socket.send(text);
        console.log('Message sent: ', text);
      }, 1000);
    });
  });
}
```

After that, implement two more event listeners:

- Check incoming data on message receipt from the server and log it to the console
- Log to console on socket close

```
export default function () {
  ...
  const res = ws.connect(url, null, function (socket) {
```

```
    ...
    socket.on('message', function message(data) {
      console.log('Message received: ', data);
      check(data, { 'data is correct': (r) => r && r === text });
    });

    socket.on('close', () => console.log('disconnected'));
  });
}
```

The easiest way to close the connection is via the `setTimeout` function. Let's configure it to close after five seconds (`5000` milliseconds).

```
export default function () {
  ...
  const res = ws.connect(url, null, function (socket) {
    ...
    socket.setTimeout(function () {
      console.log('5 seconds passed, closing the socket');
      socket.close();
    }, 5000);
}
```

Lastly, add a final check function to determine if the returned response object is of status `101` . It represents the switching protocols used by WebSockets.

```
check(res, { 'status is 101': (r) => r && r.status === 101 });
```

You can find the complete code for `script.js` at the following link.

```
import ws from 'k6/ws';
import { check } from 'k6';

export let options = {
  stages: [
    { duration: '10s', target: 30 },
    { duration: '30s', target: 30 },
    { duration: '20s', target: 60 },
    { duration: '30s', target: 60 },
    { duration: '20s', target: 30 },
    { duration: '30s', target: 30 },
    { duration: '10s', target: 0 },
  ],
};

export default function () {
  const text = 'Hello from server, Guest!';
  // public websocket server for quick test
  //const url = 'wss://javascript.info/article/websocket/demo/hello';
  const url = 'ws://localhost:8000/ws';    // local websocket server

  const res = ws.connect(url, null, function (socket) {
    socket.on('open', function open() {
      console.log('connected');
      socket.setInterval(function interval() {
        socket.send(text);
        console.log('Message sent: ', text);
      }, 1000);
    });

    socket.on('message', function message(data) {
      console.log('Message received: ', data);
      check(data, { 'data is correct': (r) => r && r === text });
    });

    socket.on('close', () => console.log('disconnected'));

    socket.setTimeout(function () {
      console.log('5 seconds passed, closing the socket');
      socket.close();
    }, 5000);
  });

  check(res, { 'status is 101': (r) => r && r.status === 101 });
}
```

k6-websocket-script.js hosted with ❤️ by GitHub view raw

Test

Let's test the script by running the following command in a new terminal.

```
k6 run k6-websocket-script.js
```

You should see the following user interface in the console which indicates that the test is running without error.

```
INFO[0001] connected                                     source=console
INFO[0001] connected                                     source=console
INFO[0001] connected                                     source=console
INFO[0002] connected                                     source=console
INFO[0002] Message sent:  Echo                           source=console
INFO[0002] Message recevied:  Echo                       source=console
INFO[0002] connected                                     source=console
INFO[0002] Message sent:  Echo                           source=console
INFO[0002] Message recevied:  Echo                       source=console
INFO[0002] connected                                     source=console
INFO[0002] Message sent:  Echo                           source=console
INFO[0003] Message recevied:  Echo                       source=console
INFO[0003] connected                                     source=console
```

Image by the author

At the end of the test, you should get the following output indicating the load testing result.

```
running (2m33.9s), 00/60 VUs, 387 complete and 0 interrupted iterations
default ✓ [======================================] 00/60 VUs  2m30s

    ✓ data is correct
    ✓ status is 101

    checks.................: 100.00% ✓ 1110 ✗ 0
    data_received..........: 84 kB   546 B/s
    data_sent..............: 85 kB   550 B/s
    iteration_duration....: avg=15.83s min=5.03s   med=14.72s max=30.66s  p(90)=27.39s p(95)=28.14s
    iterations.............: 387     2.514962/s
    vus....................: 7       min=3  max=60
    vus_max................: 60      min=60 max=60
    ws_connecting..........: avg=2.1ms  min=939.2µs med=1.99ms max=16.95ms p(90)=2.94ms p(95)=2.99ms
    ws_msgs_received......: 723     4.698495/s
    ws_msgs_sent..........: 922     5.991718/s
    ws_session_duration...: avg=15.83s min=5.03s   med=14.72s max=30.66s  p(90)=27.39s p(95)=28.14s
    ws_sessions...........: 387     2.514962/s
```

Image by the author

Conclusion

Let's recap what you've learned today.

This tutorial started off with a brief explanation of the difference between WebSocket and HTTP.

The next step was to set up and install the necessary packages.

Later on, this article guided you to explore in-depth the basic concepts and usage behind the WebSocket module provided by k6.

After that, we moved onto the implementation part, where you built your own load-testing script, which verified if the `Response` status was `101` and checked the messages sent by the WebSocket server. It'd have disconnected from the server after five seconds.

Lastly, the test section covered how to run the script and analyze the result of the test.

Thanks for reading. Hope to see you again in the next article!

References

1. k6 — WebSockets tutorial
2. k6 — WebSockets API
3. FastAPI — WebSockets

Programming JavaScript DevOps Web Development Kubernetes

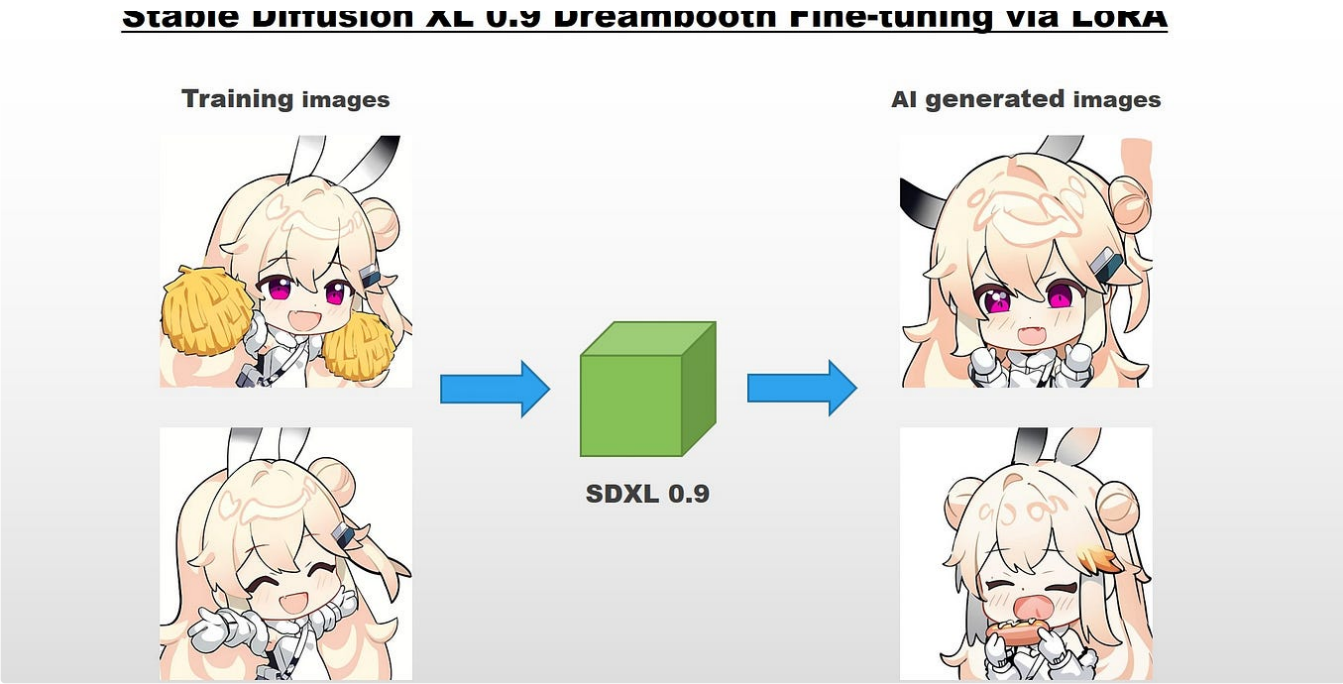
Follow

Written by Ng Wai Foong

4.2K Followers · Writer for Better Programming

Senior AI Engineer@Yoozoo | Content Writer #NLP #datascience #programming #machinelearning | Linkedin: https://www.linkedin.com/in/wai-foong-ng-694619185/

More from Ng Wai Foong and Better Programming



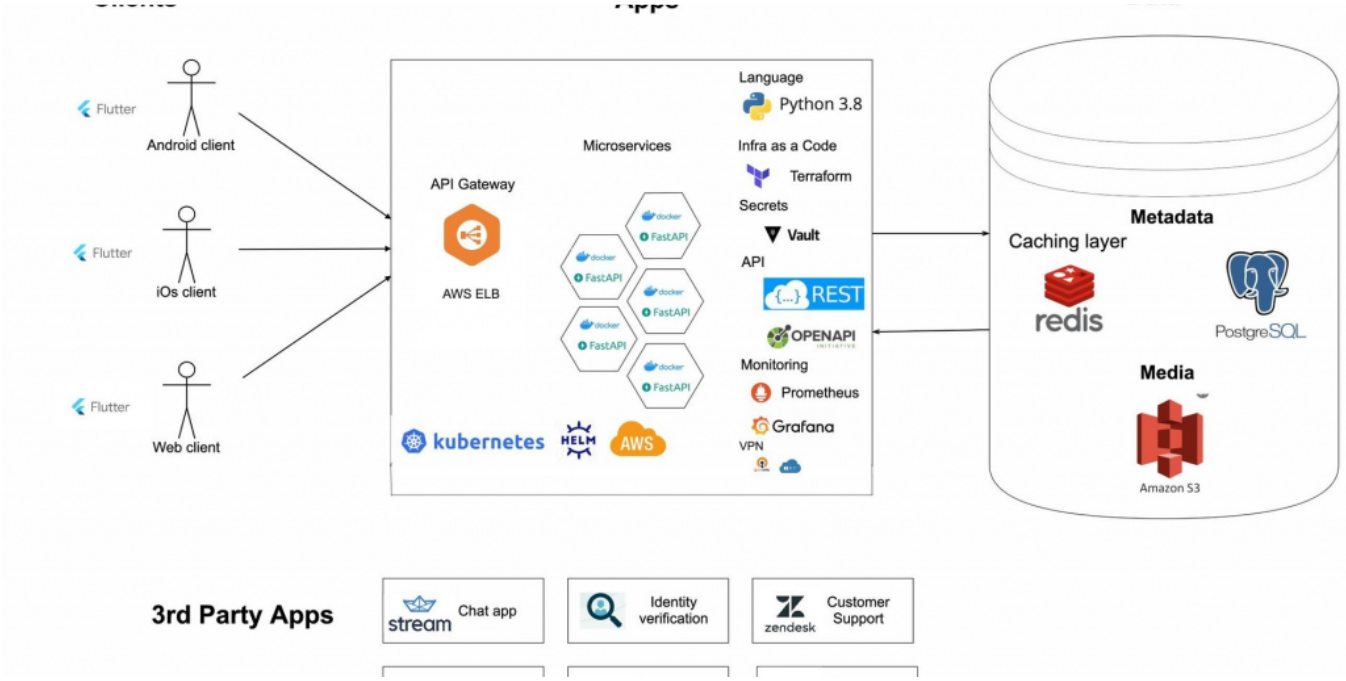
Ng Wai Foong

How to Fine-tune SDXL 0.9 using Dreambooth LoRA

Personalized generated images with custom datasets

· 6 min read · Jul 13

12 2



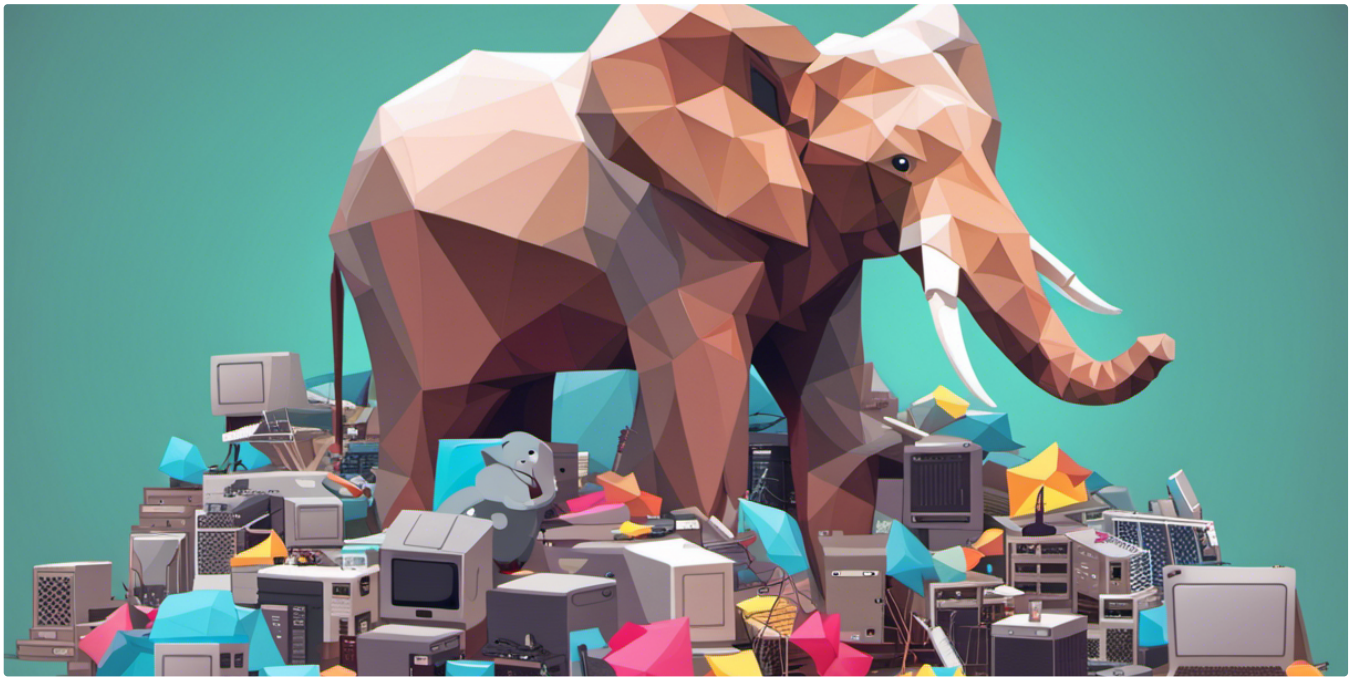
Dmitry Kruglov in Better Programming

The Architecture of a Modern Startup

Hype wave, pragmatic evidence vs the need to move fast

16 min read · Nov 8, 2022

4.2K 42



Sergei Savvov in Better Programming

7 Frameworks for Serving LLMs

Finally, a comprehensive guide into LLMs inference and serving with detailed comparison.

14 min read · 5 days ago

1.3K 10

Ng Wai Foong in Better Programming

The Beginner's Guide to Pydantic

A Python package to parse and validate data

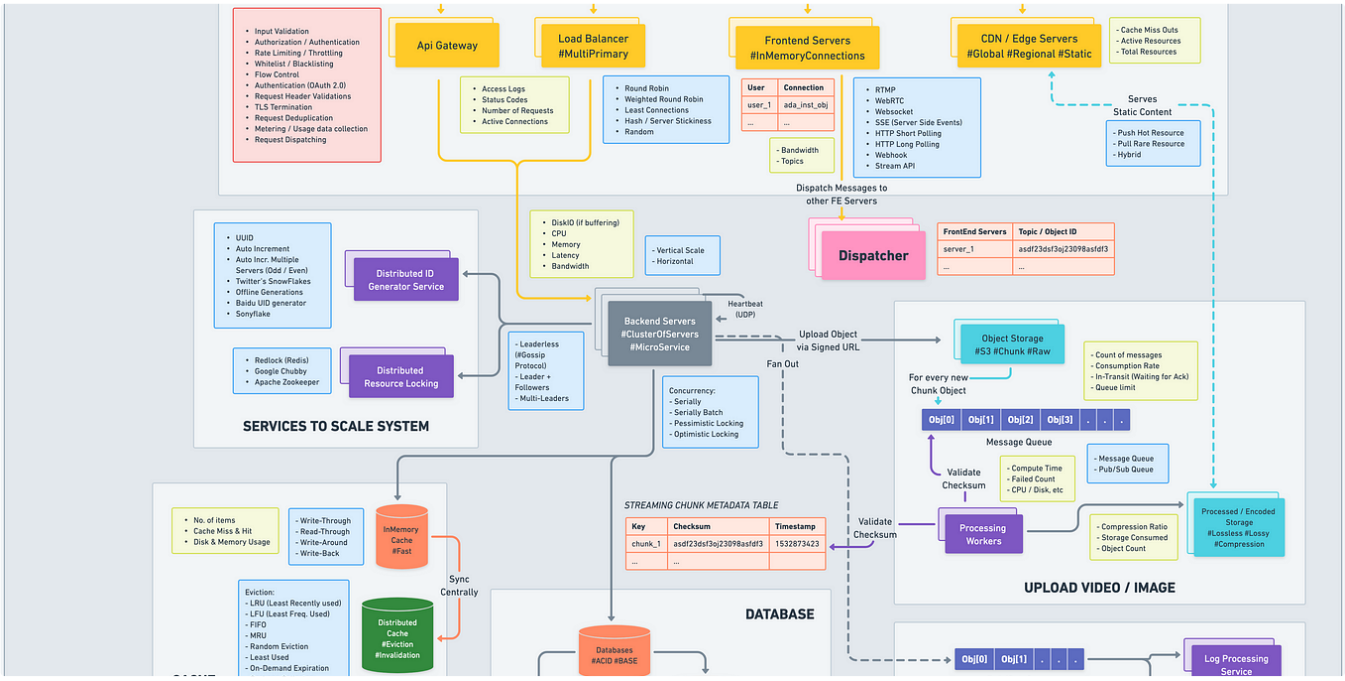
· 7 min read · Aug 11, 2020

734 2

See all from Ng Wai Foong

See all from Better Programming

Recommended from Medium



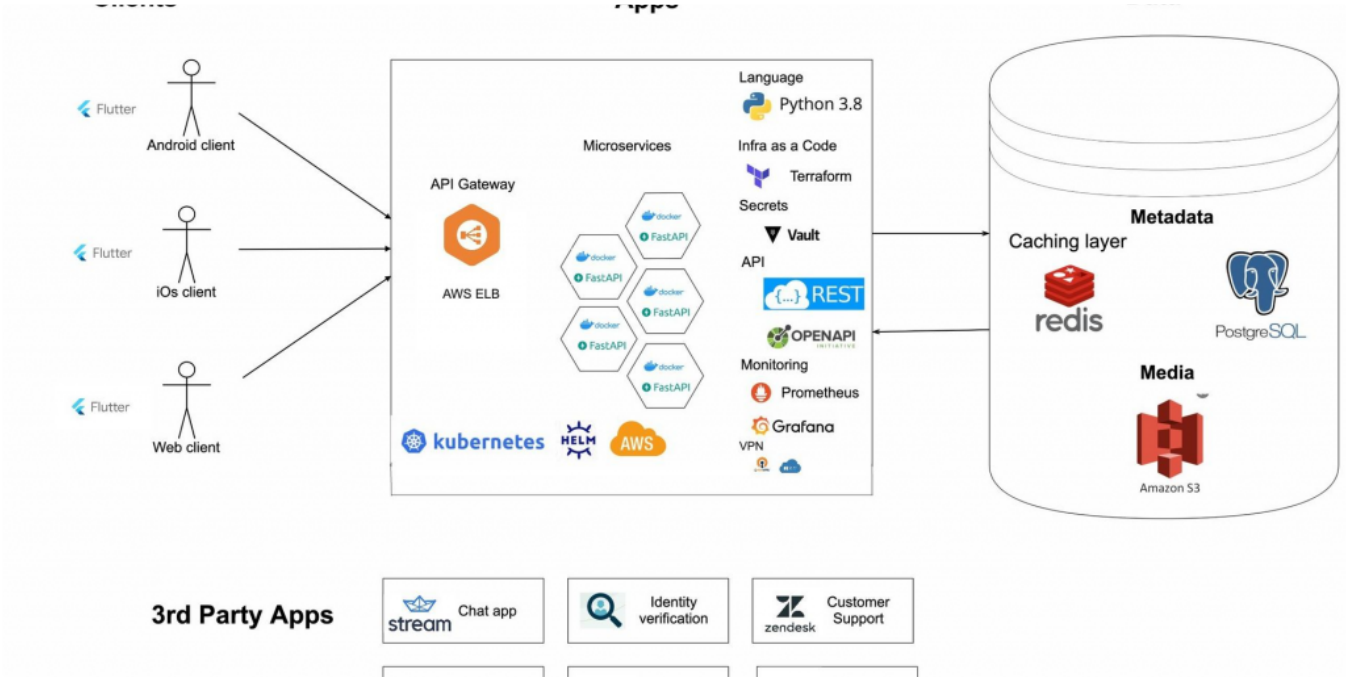
Love Sharma in ByteByteGo System Design Alliance

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

✦ · 9 min read · Apr 21

7.5K 52



Dmitry Kruglov in Better Programming

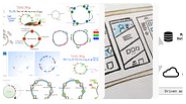
The Architecture of a Modern Startup

Hype wave, pragmatic evidence vs the need to move fast

16 min read · Nov 8, 2022

4.2K 42

Lists



General Coding Knowledge

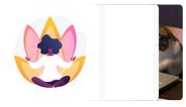
20 stories · 214 saves

It's never too late or early to start something

13 stories · 72 saves

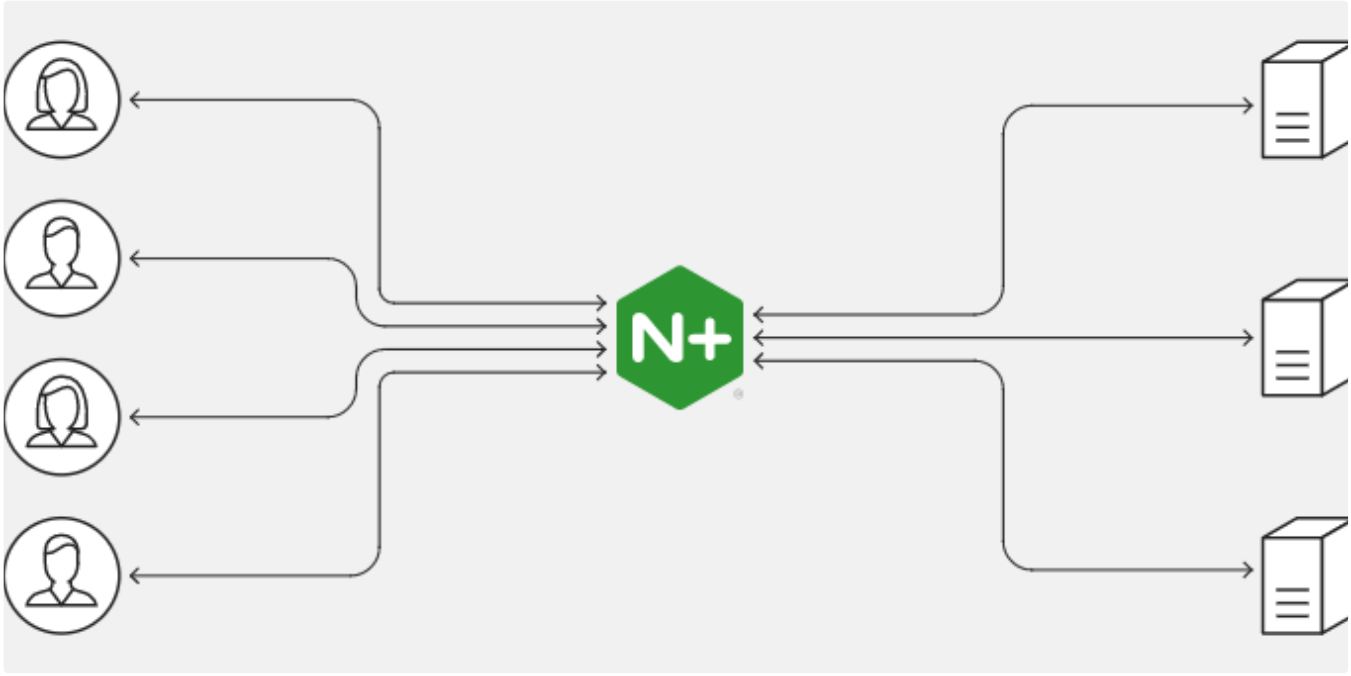
Coding & Development

11 stories · 104 saves



Stories to Help You Grow as a Software Developer

19 stories · 278 saves

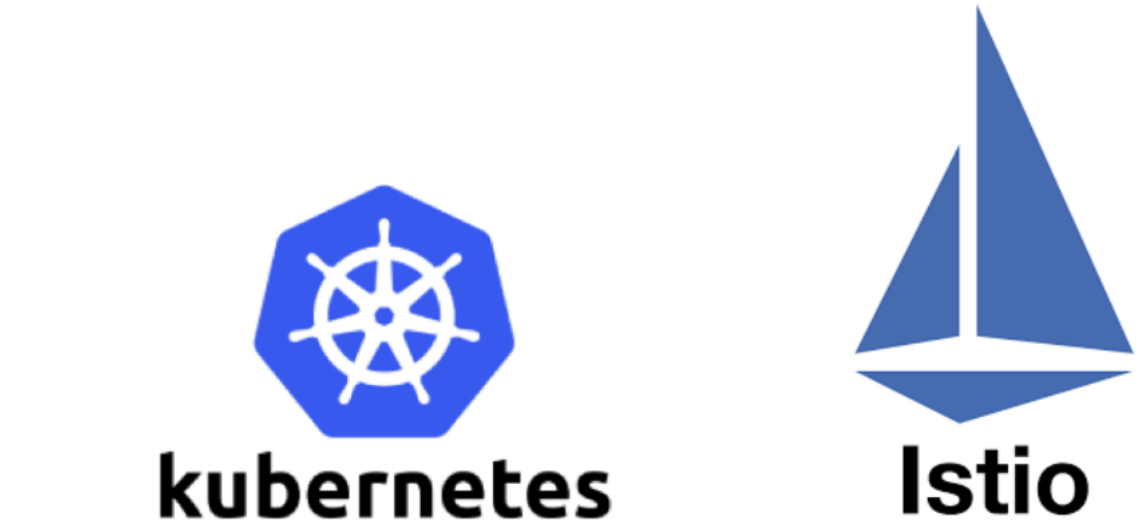


AM

NGINX as a Reverse Proxy: Benefits and Best Practices

NGINX is a versatile and powerful web server that offers a multitude of functionalities. It can be used for various purposes, such as

5 min read · Apr 28



NIRAV SHAH

Sticky Session with Istio

Envoy+Istio is widely adopted as the standard protocol in the Kubernetes ecosystem. However, configuring even simple Nginx settings within...

3 min read · Mar 19

5

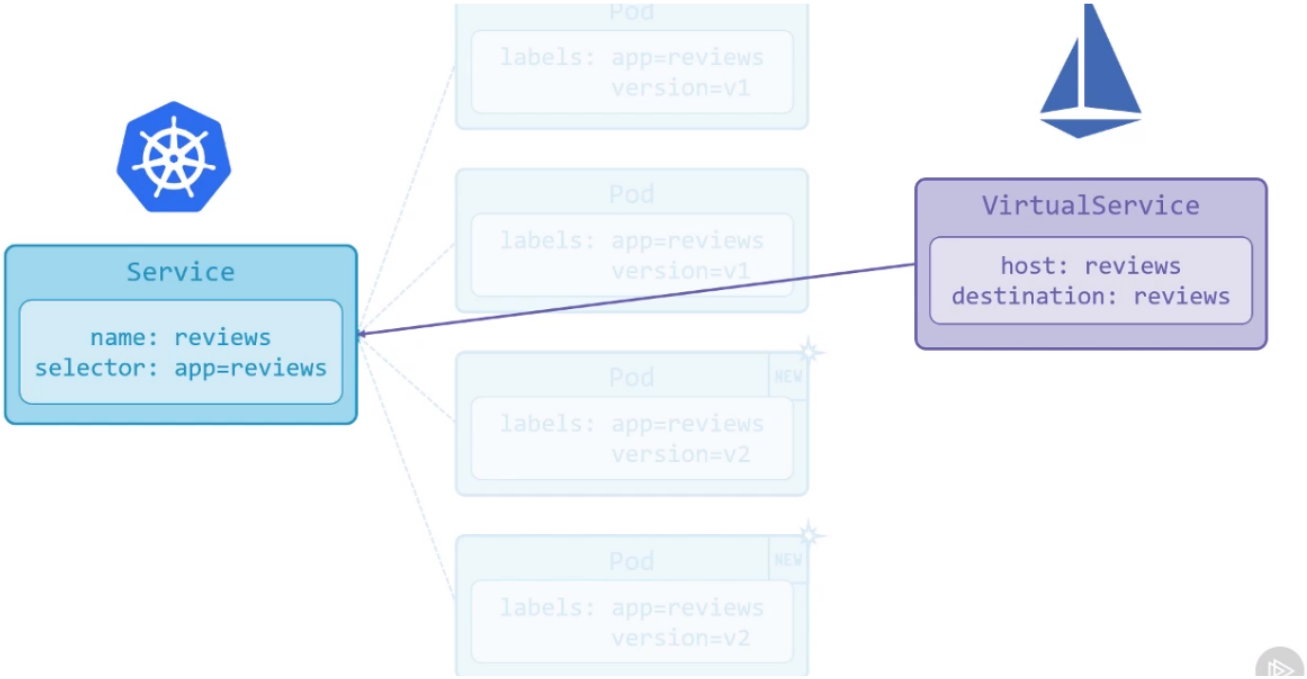
Tuhin Banerjee in Bits and Pieces

A Comprehensive Guide to Redis Cluster

How Redis Cluster works—communication, management, and more and learn how to set up a Redis Cluster locally.

15 min read · Aug 2

5



Doshin

Mastering Virtual Services with Istio

Introduction:

5 min read · Aug 3

2

See more recommendations