



STAGE

플랫폼 소개

데이터 펌플러스 주식회사

배 경



❖ 소프트웨어 개발 방법론의 변화

- 개발과 운영이 단일화 되는 형태로 변화 (DevOps)

❖ 효과적 자원관리와 개발 요구

- FaaS(Function-as-a-Service)

❖ 변화하는 환경

- 인프라 환경의 변화: 클라우드 기반
- 소비 환경의 변화: 특화되는 콘텐츠에 집중화
- 개발 환경의 변화: 직접 개발 → 사용 & 활용
- 개발 인력의 변화: 풀 스택(Full-Stack) 개발자 증가



Google에서 개발된
Javascript 엔진(V8)을 사용

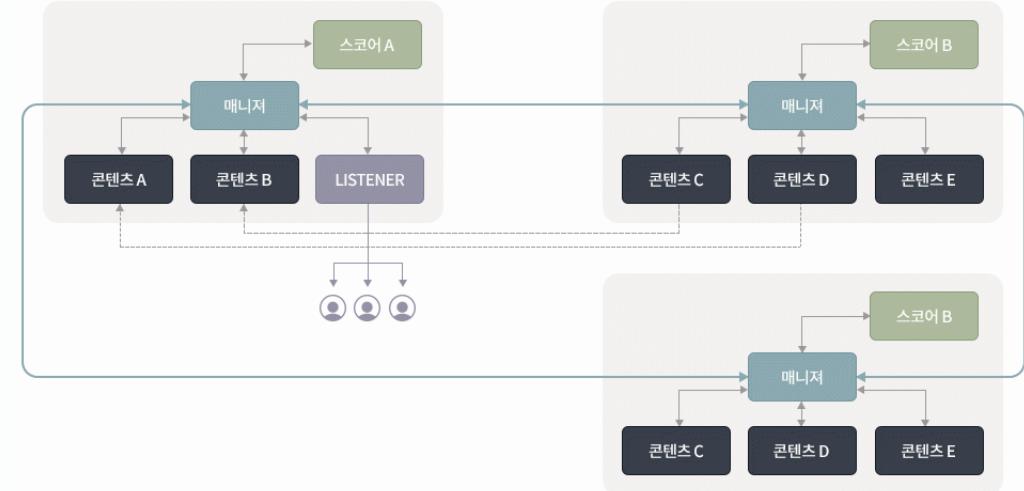
다수의 기업이 서비스에 사용 중

- PayPal, Netflix, 네이버 등

클라우드 결합해 FaaS에서 활용

- AWS Lambda
- Google Cloud Functions

STAGE : 플랫폼

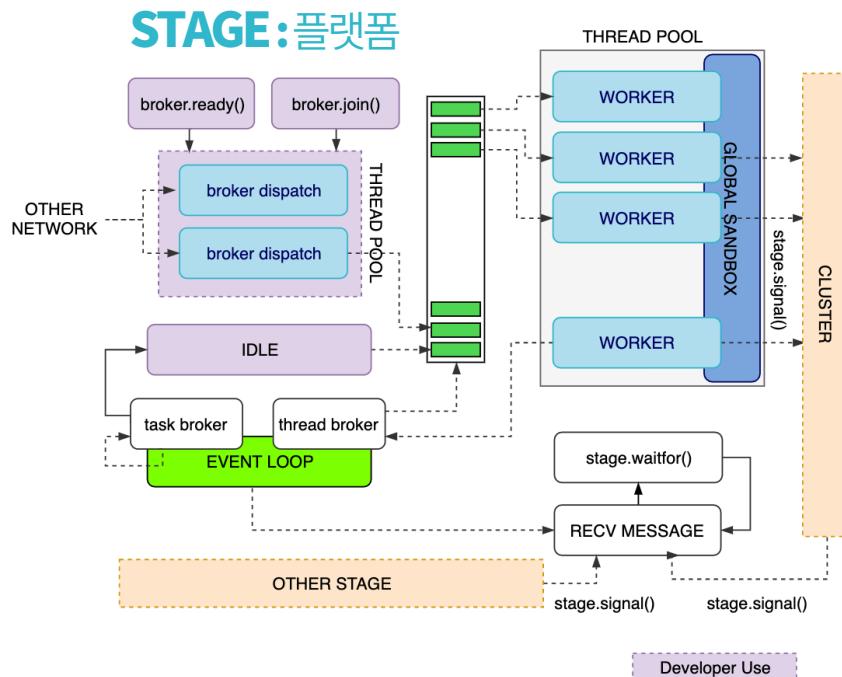


제품(개발 플랫폼) 형태

- 안정성을 유지하면서 최적화된 개발
예: Unity(게임엔진), TensorFlow(딥러닝)
- 기능(Function) 단위 개발 및 확장 구조
- 오류 및 응답 지연에 대한 강력한 안정화 기능

높은 생산성과 운영 안정성

DevOps(Development + Operation) 역할에 최적화



샌드박스 실행

보호된 영역에서 실행되는 스크립트

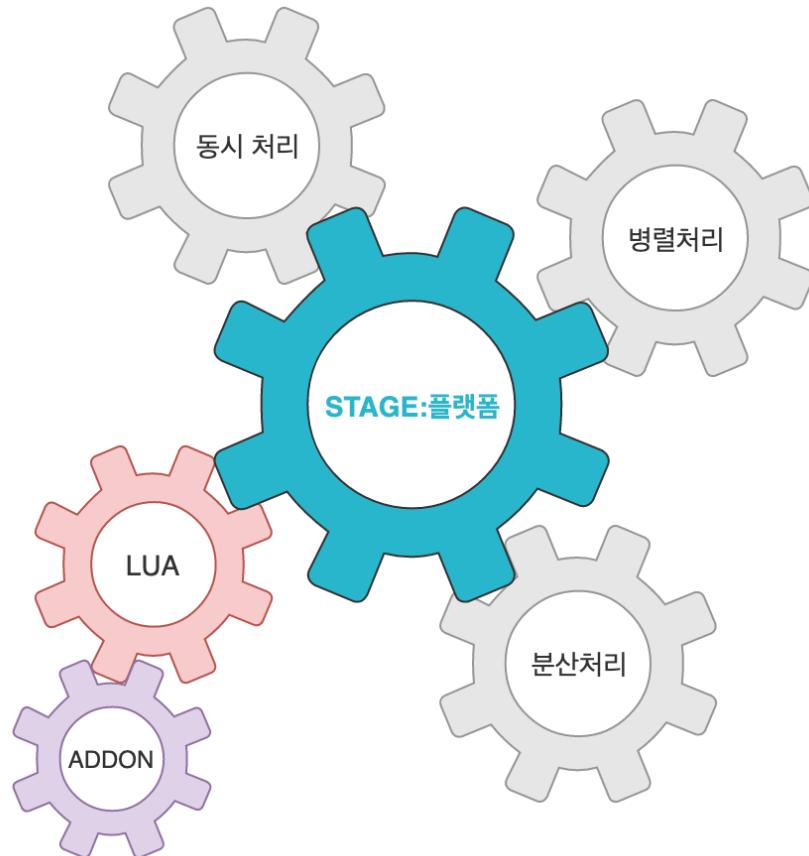
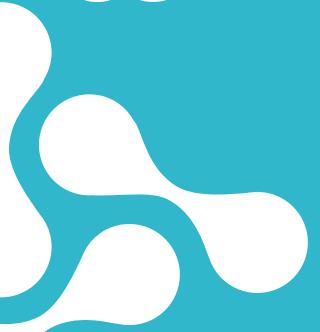
스냅샷 접근

동시 접근에 대한 안정적인 접근 제공

처리 지연 발생에 대한 적극적 대응

1. 주기적인 기능(Function)의 실행 제한
2. RPC(Remote Procedure Call)기능 해제와 집중 처리 구조 전환
3. 과도한 부하에 대한 강제 종료

프로세서 단위 모니터링 및 자동화된 복구



분산 및 병렬&동시(스레드) 처리

STAGE:플랫폼 간 밸런싱

RPC(Remote Procedure Call) 지원

RPC over LUA: 함수(Fuction) 전송 후 원격 실행

STAGE:SDK를 이용한 Native/C++ 지원

성능 향상을 위한 확장성

1. Native(C/C++)로 개발 가능한 애드온
2. luarocks 애드온 사용 (<https://luarocks.org>)
>> LUV(libuv binding for lua): NODE.JS와 동일한 비동기 엔진
3. 3rd Party 서비스와 연동

The screenshot shows the Luarocks homepage. At the top, there's a brief introduction to what Luarocks is: "Luarocks is the package manager for Lua modules. It allows you to create and install Lua modules as self-contained packages called rocks. You can download and install Luarocks on Unix and Windows." Below this is a note about a security vulnerability from March 2019, with a link to "Please Read More".

Two main sections are displayed:

- Recent Modules** (View all) (Recent versions): A grid of 10 recent module releases, each with a title, author, download count, and a brief description.
- Most Downloaded** (This week): A grid of 10 most downloaded modules this week, also with their details.

Below these sections is a "View Modules by Labels" section showing a large list of labels used for modules, such as audio, authentication, awesome, aws, base-n, batteries, bitwise, commandline, compression, crypto, database, datastructure, dbus, debug, dns, doc, email, event, feed, ffi, filesystem, game, git, gui, haml, html, http, i18n, image, ini, irc, jit, json, lapis, lint, linux, logs, love, jpeg, markdown, math, messagepack, metatable, mqlua, moonscript, mpi, network, object, openresty, posix, redis, regex, rocks, search, serialization, spreadsheet, statemachine, strict, template, test, threads, time, torch, unicode, web, wiki, windows, wsapi, xml, yaml, zeromq.

At the bottom is a "Daily Module Downloads" chart showing the number of daily downloads over a month, with a shaded area representing the range of data.

DevOps(Development + Operations)

- 도커(Docker)를 통해 손쉽게 개발 환경 구축
- 백엔드 개발에 최적화된 스크립트
- 다양한 확장 패키지: <https://luarocks.org/>

분업(협업)에 최적화: 기능(Function) 단위 개발

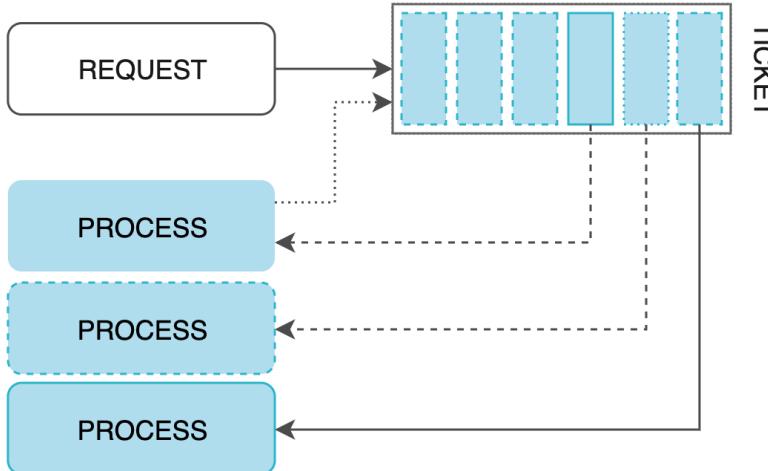
비교적 낮은 개발 접근성

- 단순화되어 실무자(비 전문가: 기획자 등)의 개발 가능
- 결과물에 대한 일정한 품질 유지 가능

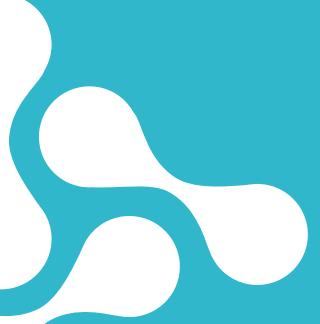
분산 및 확장(Scale-up & Down)에 대한 대응 개발 불필요

동시 실행 제한을 효과적으로 관리

예: 티켓 예약, 동시 로그인 수 제한, 승부 판단 처리, 채팅 룸 등

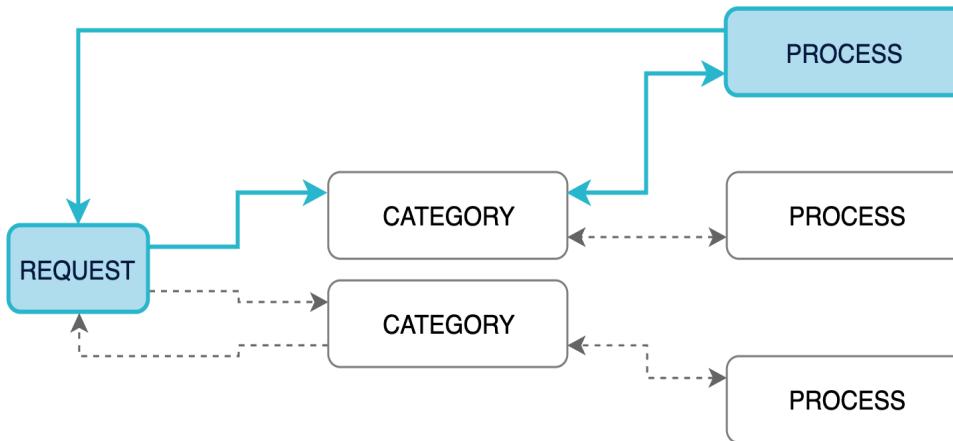


- 제한된 시간 사용 후 자동 회수: 자연 가능
- 시간 및 대기(예약) 조건 설정 다양화
 - 1) 일정 시간 이후 발행 중지
 - 2) 대기(예약) 가능
- TICKET에 메시지 전달 가능



효과적인 메시지 로드밸런싱

예: 1:1 채팅, 데이터베이스 연동 서버



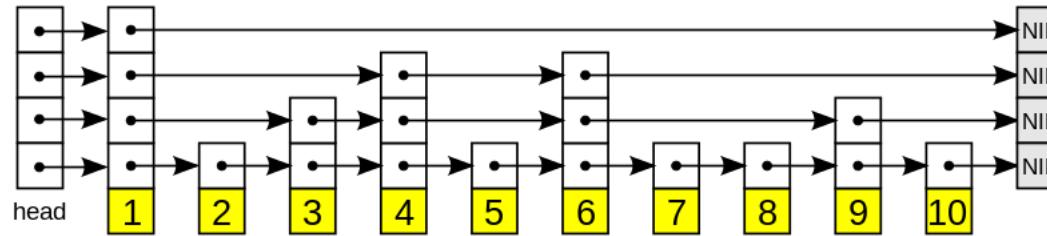
특징

- 선택적 로드밸런싱을 통해 메시지 처리
- 함수(Function)와 명령에 대한 재정의 : Mapping RPC
- 메시지 전달 여부 확인 가능



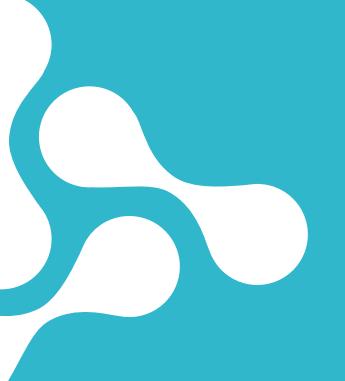
Key-Value기반 In-Memory 색인 엔진

예: 게임의 랭킹, 로깅 분석



특징

- Key에 연결되는 다수의 정렬 값
- 다양한 형태의 검색
 - 1) Key를 기준으로 한 범위 검색
 - 2) 순서에 대한 범위 검색
 - 3) 정렬 값 근접 범위 검색
- 색인 파일로 저장



TICKET

- 데이터베이스 캐싱:+LUA
- 로그인 대기열
- 채팅 공간(룸)

ROUTE

- 채널(방송) 메시지
- 메시지 정형화

INDEX

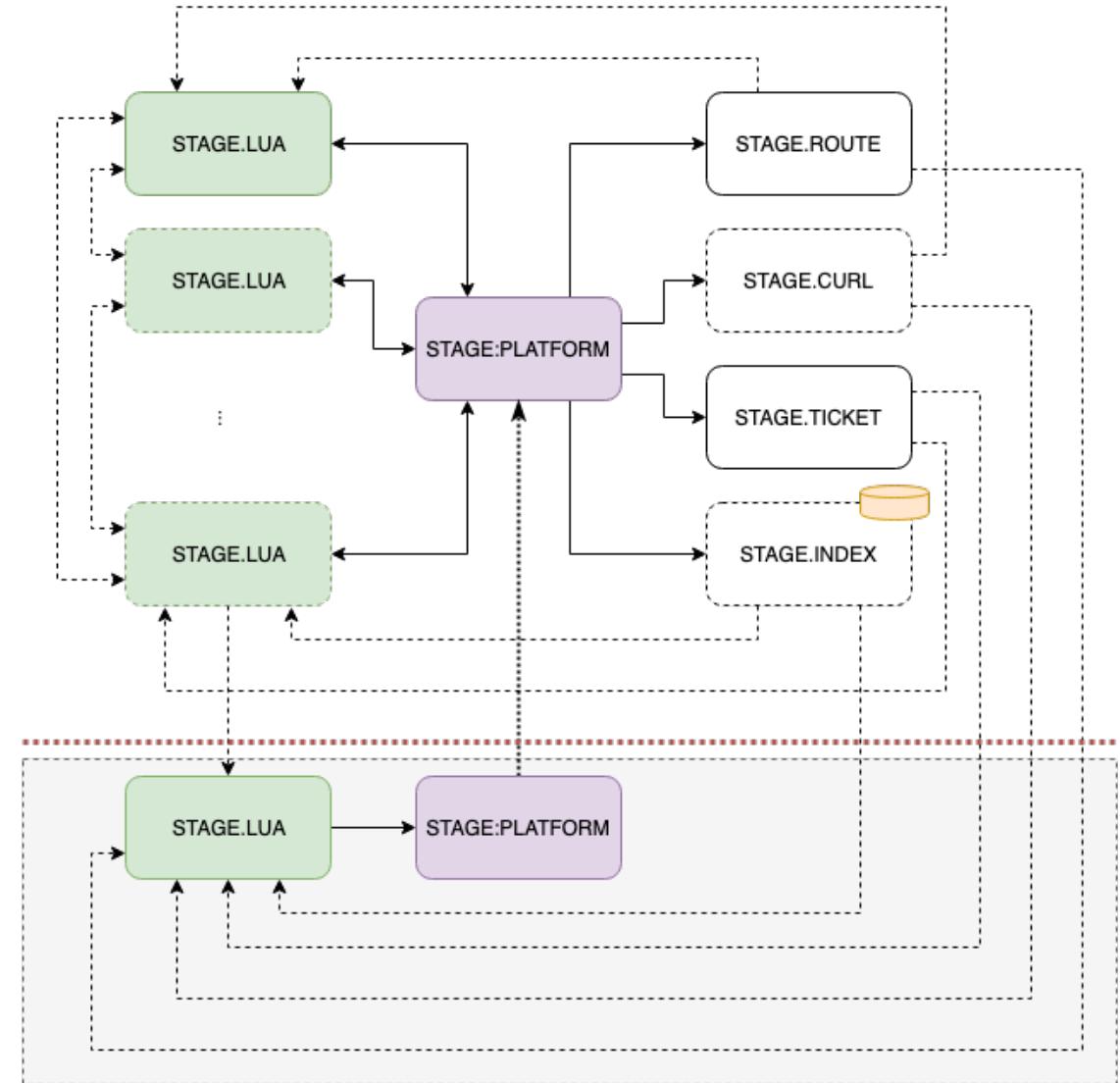
- 랭킹
- 키워드 색인

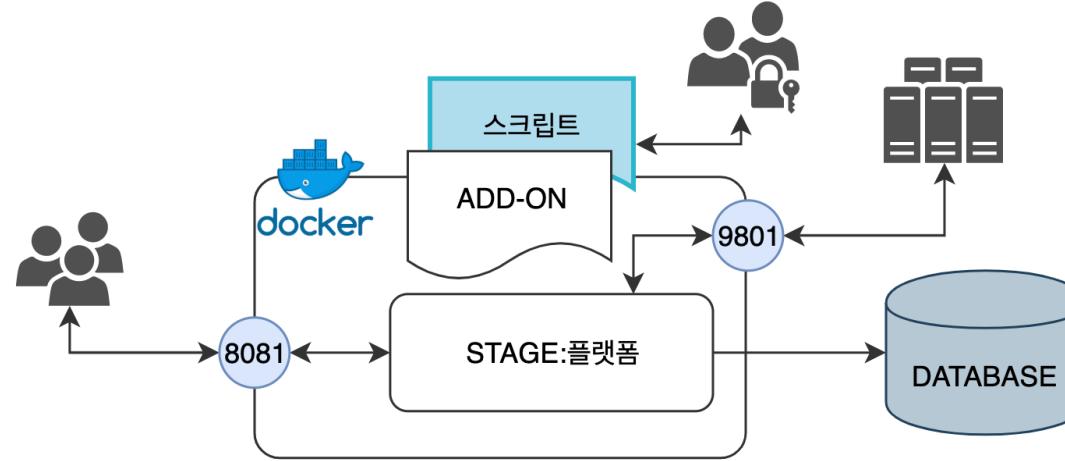
CURL

- REST API 연동
- HTTP(S) 기반 다운로드

LUA/LUAJIT

- 콘텐츠 개발





```
$ docker run -it --rm -v $PWD/stage:/opt/sgage -p 8081:8081 -p 9801:9801 datumflux/stage
```

생산성 향상

- 개발 접근성(스크립트 & 일정한 결과) 향상
- 개발 인력 구성이 용이
- 개발 코드의 재활용 향상
- 개발 및 테스트 환경 효율화: 도커(Docker) 사용 – Any OS

운영 효율성 증대

- 자원(Resource)의 효율성: 기능(Function) 단위의 확장 가능

직관적 구현

```

local uv = require('luv')

local function create_server(host, port, on_connection)
    local server = uv.new_tcp()
    server:bind(host, port)
    server:listen(128, function(err)
        -- Make sure there was no problem setting up listen
        assert(not err, err)

        -- Accept the client
        local client = uv.new_tcp()

        server:accept(client)
        on_connection(client)
    end)
    return server
end

local server = create_server("0.0.0.0", 0, function (client)
    client:read_start(function (err, chunk)
        -- Crash on errors
        assert(not err, err)
        if chunk then
            -- Echo anything heard
            client:write(chunk)
        else
            -- When the stream ends, close the socket
            client:close()
        end
    end)
end)

print("TCP Echo server listening on port " .. server:getsockname().port)
return function ()
    require('luv').run('nowait')
    return 0
end

```

STAGE:플랫폼

- tcp server 구현 예제

차이

```

var net = require('net');
var server = net.createServer(function(client) {
    client.setTimeout(500);
    client.setEncoding('utf8');

    client.on('data', function(data) {
        writeData(client, data.toString());
    });

    client.on('end', function() {
        console.log('Client disconnected');
        server.getConnections(function(err, count){
            console.log('Remaining Connections: ' + count);
        });
    });

    client.on('error', function(err) {
        console.log('Socket Error: ', JSON.stringify(err));
    });

    client.on('timeout', function() {
        console.log('Socket Timed out');
    });
});

server.listen(8107, function() {
    console.log('Server listening: ' + JSON.stringify(server.address()));
    server.on('close', function(){
        console.log('Server Terminated');
    });
    server.on('error', function(err){
        console.log('Server Error: ', JSON.stringify(err));
    });
});

function writeData(socket, data){
    var success = !socket.write(data);
    if (!success){
        (function(socket, data){
            socket.once('drain', function(){
                writeData(socket, data);
            });
        })(socket, data);
    }
}

```

node.js



효과적인 스케일링

- 소프트웨어 컨테이너와 함수(Function)의 분리: 공통 컨테이너
- RPC(Remote Procedure Call): 함수(Function)의 효과적 분배
 >> 고성능의 Group과 일반 Group으로 운영
- 컴팩트한 소프트웨어 컨테이너: v1.7 기준 56MB

함수(Function)체인

- RaaS(Reduce-as-a-Service): 결과를 정리해 최적의 결과 생성

다양한 형태의 데이터 스트림 지원



성능과 안정성이 검증된 경량화된 엔진

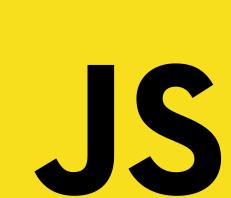
- 상호 의존성이 낮은 엔진: 상호 의존이 높을수록 메모리 문제 발생
- 잘못된 개발(코딩)에서 안정적인 메모리 관리: 메모리 강제 회수
- 데이터 주도적 설계가 목적: 데이터 처리에 최적화
- Native (C/C++) 연동이 쉬운 구조

단순한 스크립트 문법 구조

- 표현방법이 다양 할수록 코딩의 복잡도 및 단순 오류 증가
- 코딩 스타일의 단순화 필요: 복잡한 표현이 가능하면 이해 어려움

LUA

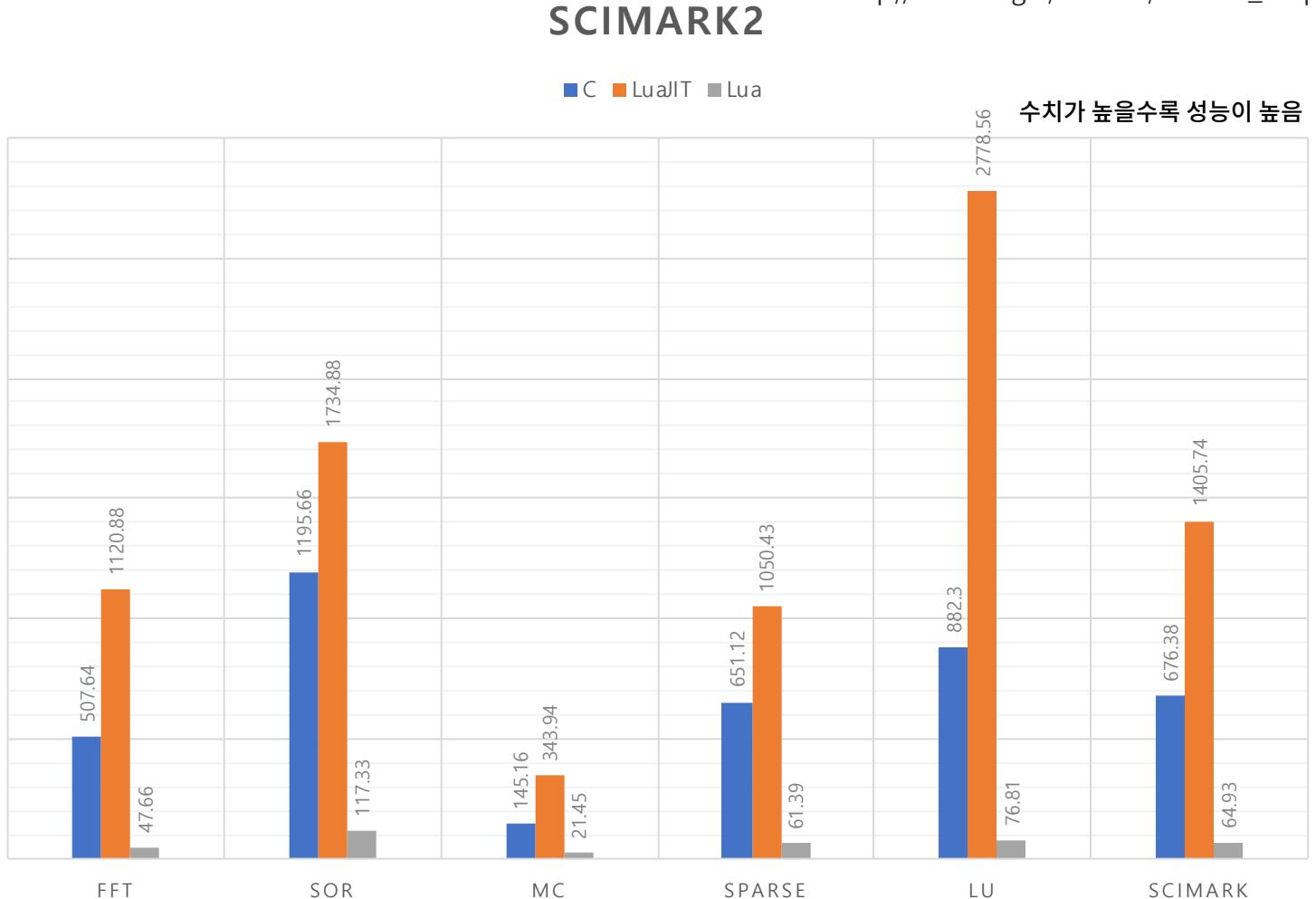
vs JavaScript



- C/C++에 포함되어 사용되는 형태로 개발 (Embed)
 - 객체 지향적이며 함수형 프로그래밍이 가능한 구조
 - 단순한 문법과 손쉬운 C/C++ 연동으로 다양한 분야에서 활용
 - JIT(Just-In-Time Compiler)를 지원하는 LuaJIT는 타 JIT보다 우수한 성능
-
- 웹에서 동적인 처리 목적으로 개발
 - node.js를 통해, 독립 실행이 가능한 형태로 서버 개발에도 사용
 - 동적 타입 언어의 특성과, 함수형이며 객체 지향형 언어의 다양한 특성
 - 예상과 다른 동작이 많고 코드 리딩과 메모리 관리에 어려움 존재

CPU 성능 테스트

- Fast Fourier Transform
- Jocobi Successive Over-Relaxation
- Monte Carlo Integration
- Spare Matrix Multiplication
- Dense Matrix Factorization





DATUM FLUX

콘텐츠에 필요한
기술과 솔루션을 개발

support@datumflux.co.kr