Porting JSONSki to JavaScript Environments

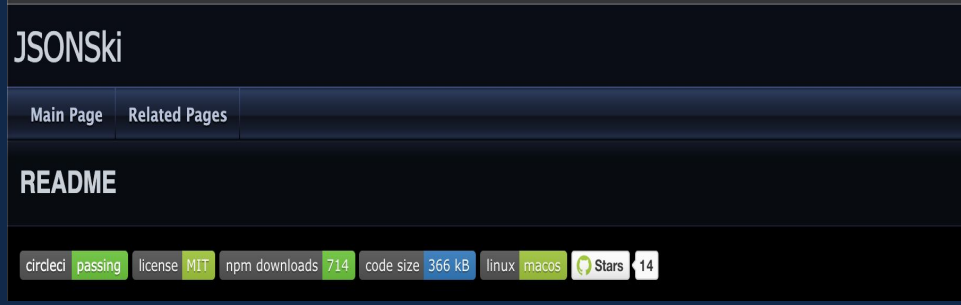
Gandharva Deshpande

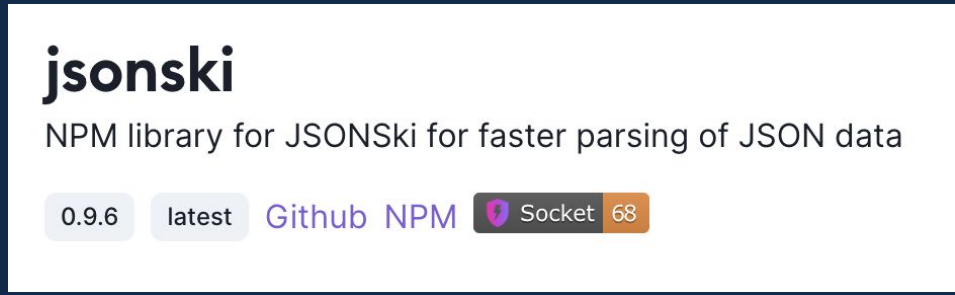
Department of Computer Science

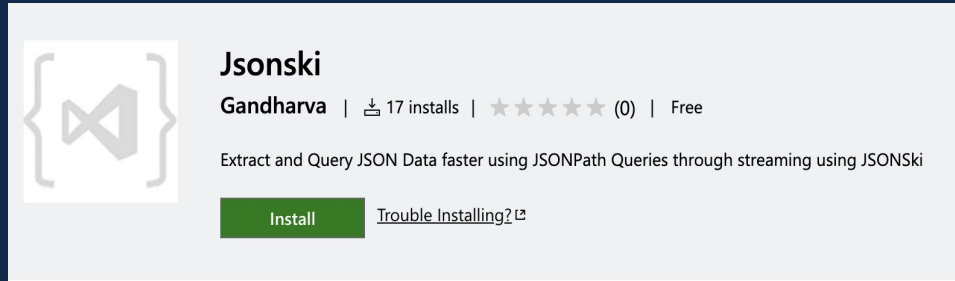
University of California, Riverside

Abstract

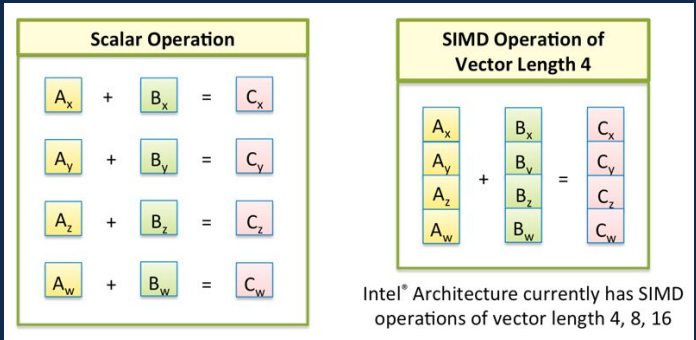
- JSONSki is a streaming JSONPath processor that was originally developed in C++ and uses SIMD and bitwise operations.
- In this project, JSONSki was ported to the Javascript running environment such that it can be accessed programmatically via a Node Project, an NPM library and used as an extension tool in VS Code.
- These three tools make JSONSki more accessible to developers and end users while keeping its performance intact.







Summary of JSONSki



JSONSki presents a bit-parallel (SIMD) solution for implementing fast-forward optimizations with high-efficiency to stream over JSON Data originally written in C++.

Key Differentiators:

1. Data Streaming over Querying + Parsing (Tree Parsing)
2. SIMD over SISD
3. Fast Forward Optimizations over Complete Parse tree

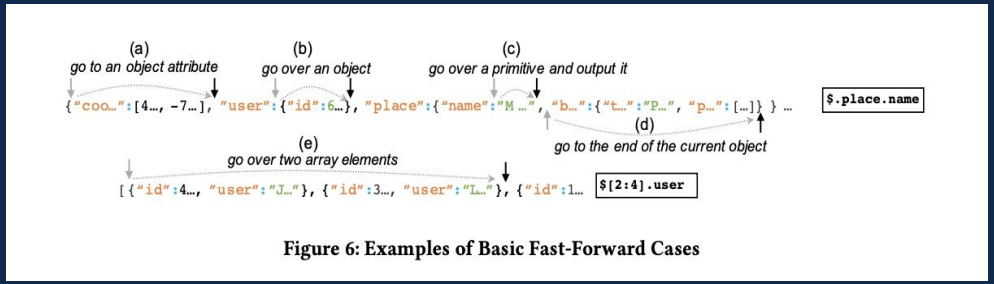


Figure 6: Examples of Basic Fast-Forward Cases

Porting C++ codebase to Node.js

While there are a lot of ways of converting C++ code into a Node.js application, creating a Node.js Add-on offers a ton of flexibility, code reusability and certain performance benefits.

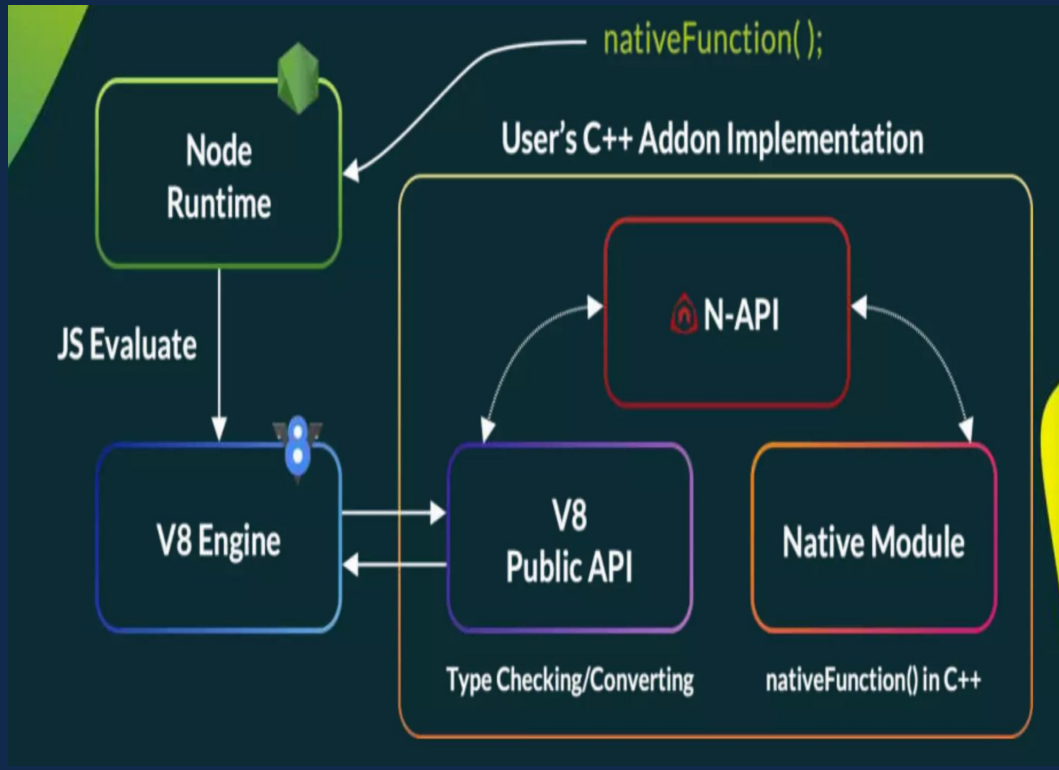
A node.js Add on - Provides interface between C++ and Javascript

The possible options for add-ons

1. NAN
2. N - API (For C)
3. Node Addon-Api (For C++)

Why Node-addon-API?

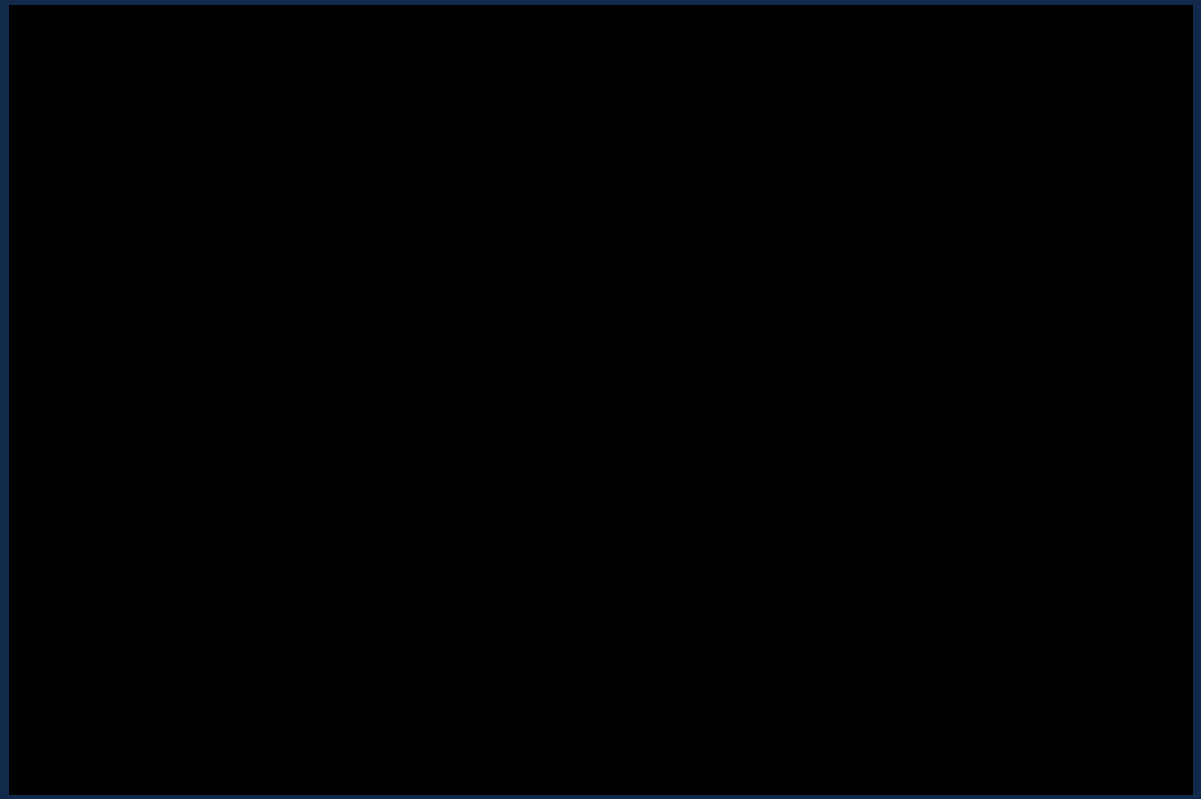
- Since Node addon API is a part of Node.js, it is documented in-depth on their official documentation.
- NAN requires rebuilding the module for each NODE_MODULE_VERSION (major version of Node.js)
- Modules using N-API/Node-Addon-API are forward-compatible
- Competitors like SIMDJSON implemented their bindings using Node-addon-API



Steps for Deployment

1. Installations
2. Writing Wrappers for C++ code base for interface functions
3. Setting up C++ flags for the build
4. Setting up index.js for the Javascript interface
5. Deploying the NPM library

Demo (Npm library)

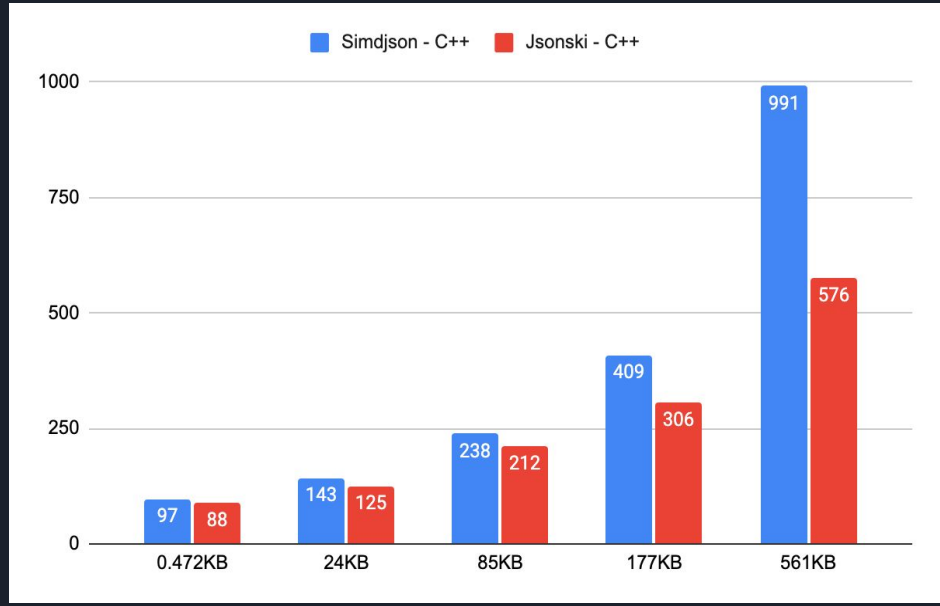


Performance Results

Native Performance Comparisons

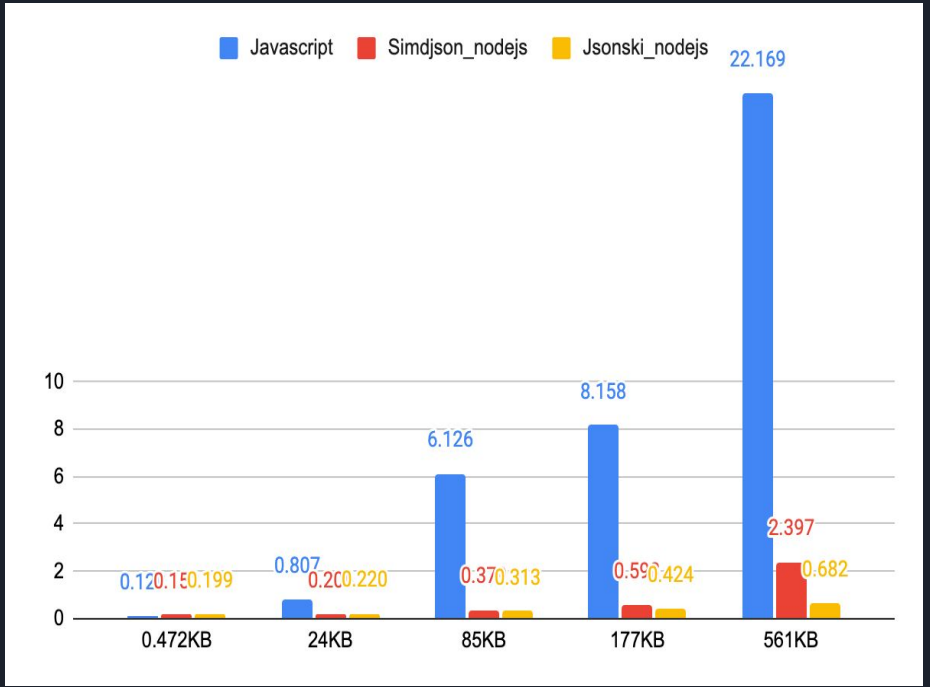
	JSONSki - C++ (load + stream)	simdjson-dom (load + parse + query)
472 Bytes	88 µs	97 µs
24 KB	125 µs	143 µs
85 KB	212 µs	238 µs
177 KB	306 µs	409 µs
561 KB	576 µs	991 µs
189.8 MB	213184 µs	541545 µs

execution Time Comparison (in microseconds)



Execution Time Comparison (in milliseconds)

Size	JSON.parse()	simdjson_nodejs	JSONSki_nodejs
472 Bytes	0.128ms	0.152ms	0.199ms
24 KB	0.807ms	0.206ms	0.220ms
85 KB	6.126ms	0.379ms	0.313ms
177 KB	8.158ms	0.598ms	0.424ms
561 KB	22.169ms	2.397ms	0.682ms
26 Mb	359.974ms	122.651ms	14.579ms
189.8 Mb	3133.188ms	927.448ms	180.056ms

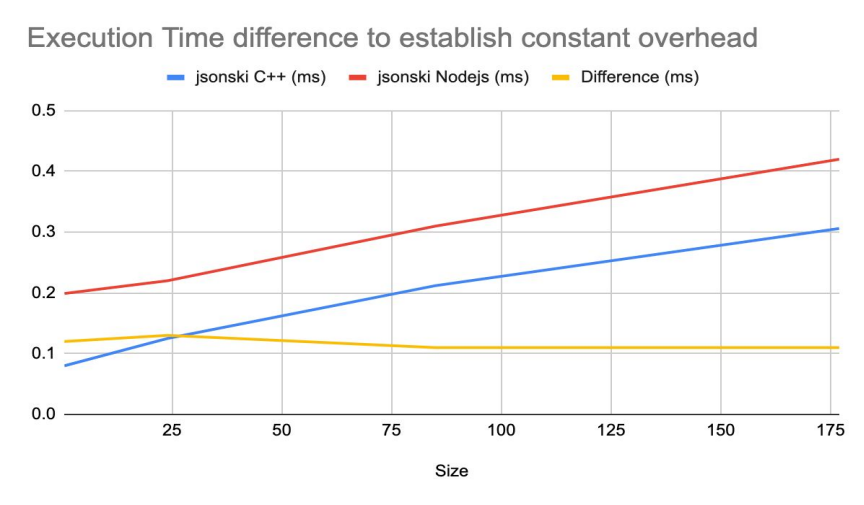


Note: Queries are 1-2 levels deep, randomly positioned Eg: $.tiger.koko (2 level deep in the JSON on line #777)

Performance Analysis (Javascript)

- Jsonski Node.Js sometimes perform slower for small to mid sized data, suspecting the reason to be that fixed wrapping overhead and an explicit inevitable type conversion, which eventually becomes negligible with increasing size.
- As the size of the data grows, JSONSki begins performing exponentially better compared to Javascript

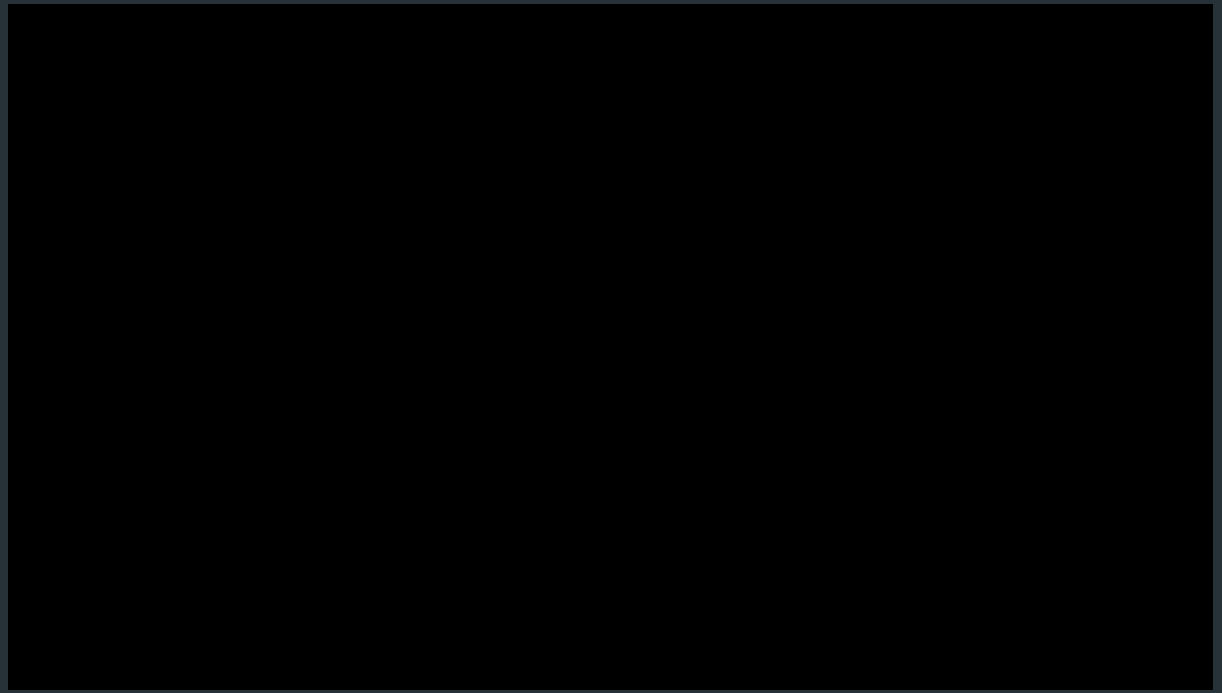
Size	jsonski C++	jsonski Nodejs	Difference
0.472	0.08	0.199	0.12
24	0.125	0.22	0.13
85	0.212	0.31	0.11
177	0.306	0.42	0.11



Query	Execution time (ms)	Number of rows
$.features[0:3]	175.94	3
$.features[0:300]	171.23	300
$.features[0:3000]	177.2	3000

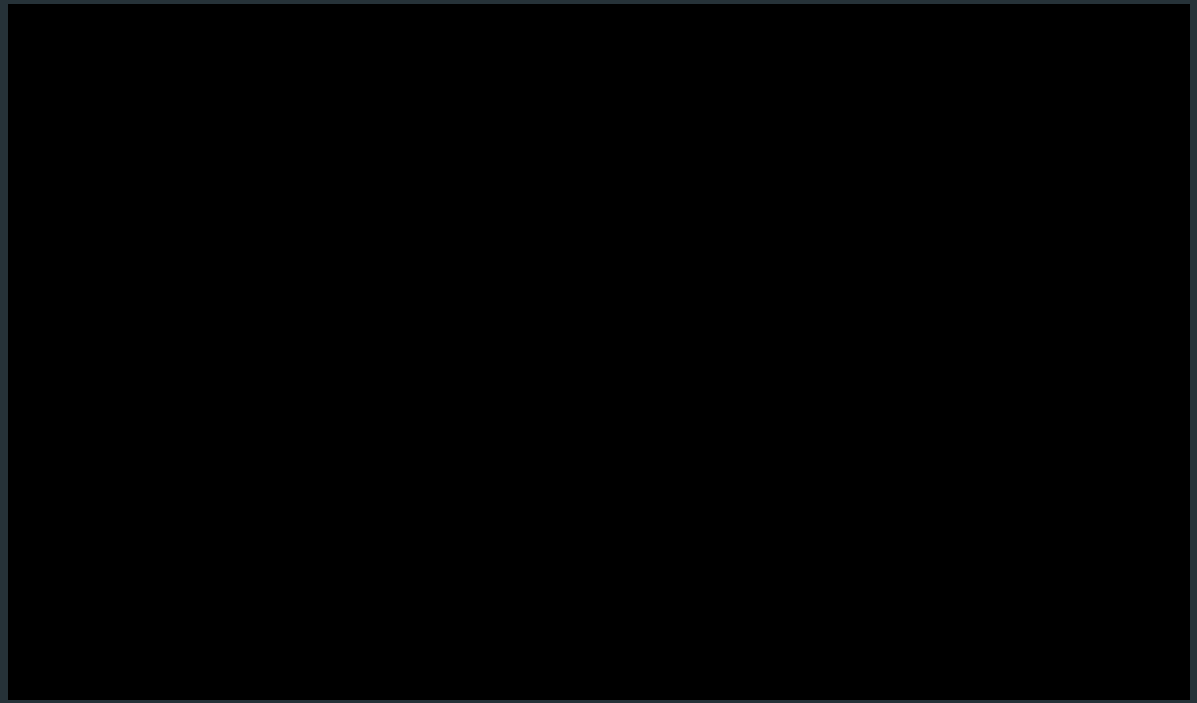
Benchmarking Repositories (Demo)

- Repositories for benchmarking:
  - JsonSki vs simdjson Vs RapidJson in C++:https://github.com/AutomataLab/JSON-Parser-Benchmarking
  - JsonSki vs simdjson in Node.js : https://github.com/AutomataLab/NPM-JSON-Parser-Benchmarking



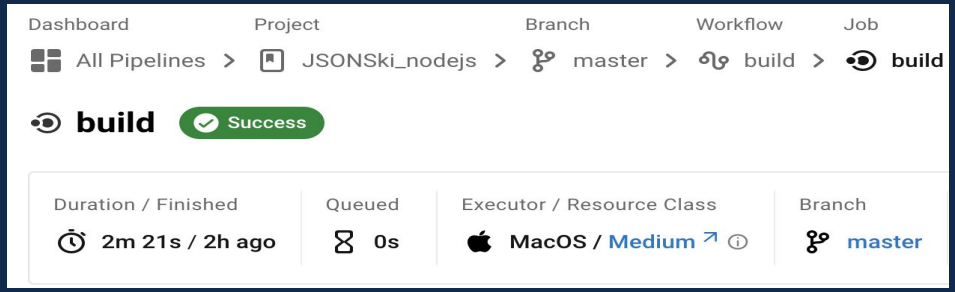
Porting to a VSCode Extension

- Local Use
- Added Accessibility and Interactivity



CI and Scalability Integrations

- Set up CircleCI pipeline for MacOS and linux for JSONSki C++ (Current Support)
- Set up MacOS pipeline for JSONSki Node.js (Current Support)
- Added Doxygen Documentation to JSONSki Repositories
- Added shield IO badges to improve visibility.



Potential Applications

1. Data Streaming performs better only when Parsing isn't already performed and we don't have the parse tree in the memory
2. Eg- streaming and not where we need to query multiple times over a stored parsed object.
3. Replacing JSON.Parse() where the application needs limited number of queries over an object
4. Tools that could potentially use JSONSki: Eg: Postman

References

JSONSki: streaming semi-structured data with bit-parallel fast-forwarding Lin Jiang , Zhijia Zhao, https://dl.acm.org/doi/pdf/10.1145/3503222.3507719

https://nodeaddons.com/getting-your-c-to-the-web-with-node-js/

https://stackoverflow.com/questions/54740947/node-js-addons-nan-vs-n-api

https://medium.com/jspoint/a-simple-guide-to-load-c-c-code-into-node-js-javascript-applications-3fcccf54fd32

https://stackoverflow.com/questions/18382957/tree-parser-vs-stream-parser

https://www.slideshare.net/nasottola/next-generation-napi

https://medium.com/jspoint/a-simple-guide-to-load-c-c-code-into-node-js-javascript-applications-3fcccf54fd32

https://blog.atulr.com/node-addon-guide/

https://ducmanhphan.github.io/2018-09-19-Configure-Binding.gyp-file-in-C++-Addon-Node.js/