



KodeKloud

© Copyright KodeKloud

Follow us on <https://kodekloud.com/> to learn more about us.

Introduction to WebAssembly (WASM)

Objectives

- 01 | Clear understanding of what WebAssembly is and why it's significant in modern web development
- 02 | Able to identify situations where WebAssembly is beneficial
- 03 | Understand its relationship with JavaScript

By the end of this section, learners will have a clear understanding of what WebAssembly is and why it's significant in modern web development. They will be able to identify situations where WebAssembly is beneficial and understand its relationship with JavaScript.

What Is WebAssembly?



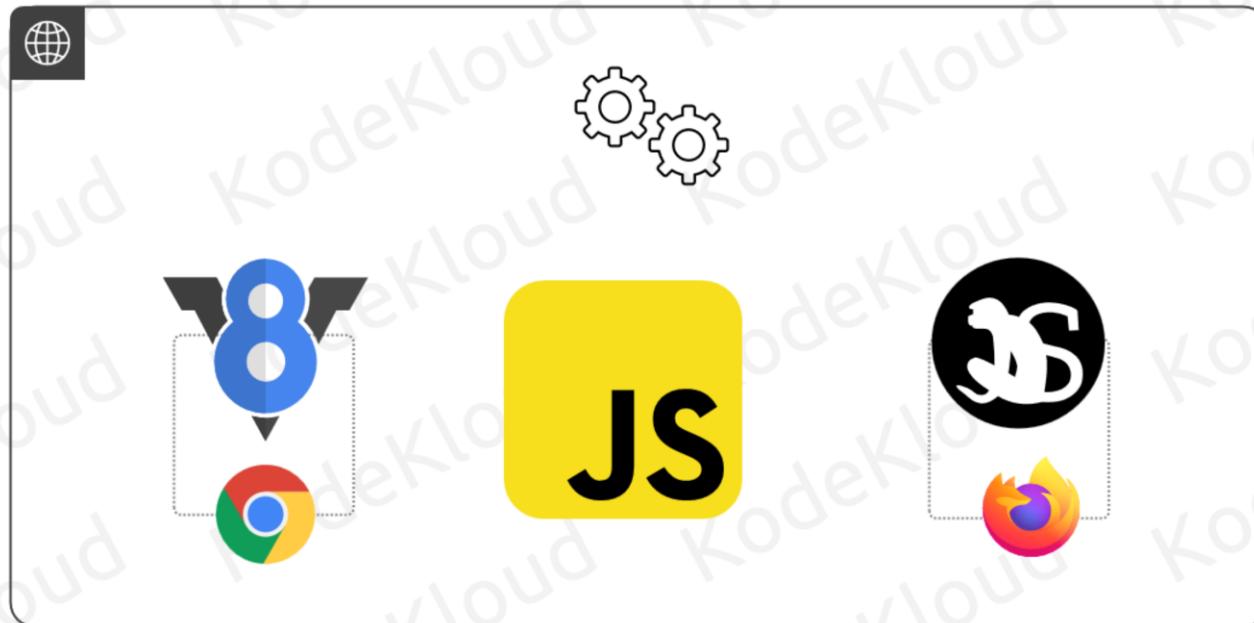
Performance Bottlenecks



© Copyright KodeKloud

JavaScript, the bedrock of web development, has powered countless online applications with its versatility and universal adoption. However, it hasn't been without its challenges:

Performance Bottlenecks



© Copyright KodeKloud

JavaScript is an interpreted language, which means it's translated on-the-fly by the browser's JavaScript engine (like V8 for Chrome or SpiderMonkey for Firefox) during execution.

Performance Bottlenecks



© Copyright KodeKloud

While modern engines employ Just-In-Time (JIT) compilation to speed up execution, it's still generally slower than pre-compiled languages. This on-the-fly translation can introduce delays, especially for tasks that require rapid, repetitive computations, such as real-time graphics rendering or complex mathematical operations.

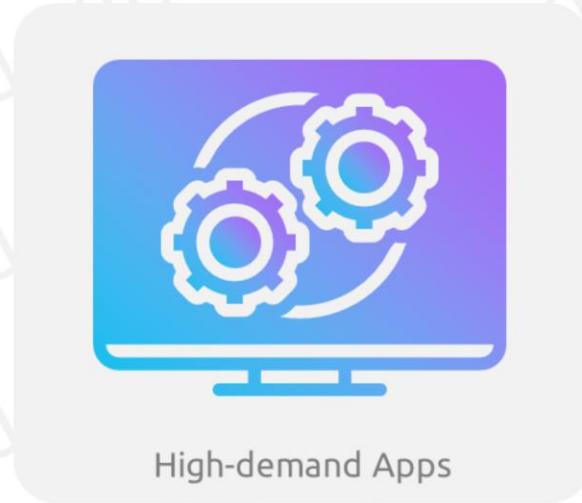
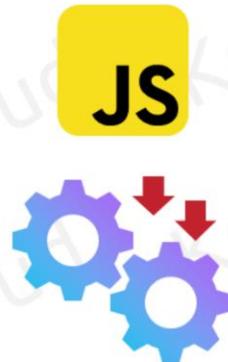
Performance Bottlenecks

The image shows a white kitten with orange patches on its ears and face, looking upwards. It is framed by a dotted border. In the top left corner is a globe icon, in the top right is a blue icon with the number '8', and in the bottom right is a yellow icon with the letters 'JS'. A green circle with a checkmark is visible in the bottom right corner of the image area.

© Copyright KodeKloud

This on-the-fly translation can introduce delays, especially for tasks that require rapid, repetitive computations, such as real-time graphics rendering or complex mathematical operations.

Performance Bottlenecks



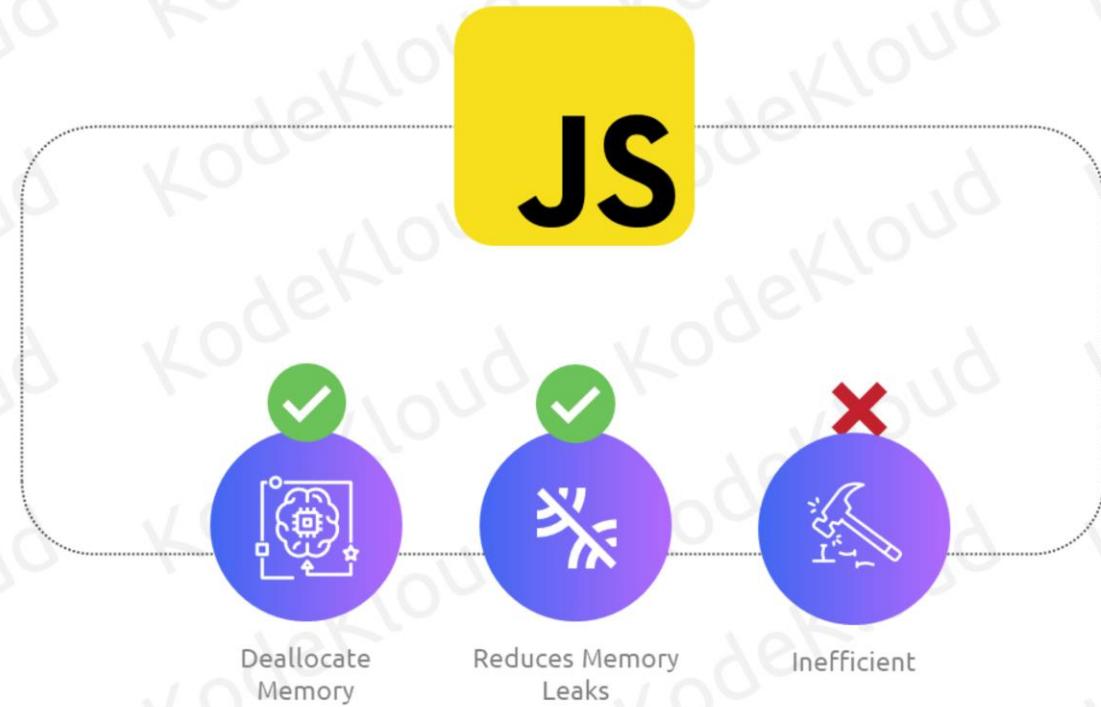
High-demand Apps



© Copyright KodeKloud

High-demand applications, especially those requiring real-time graphics or intensive computations, often encounter performance hiccups with JavaScript.

Memory Limitations

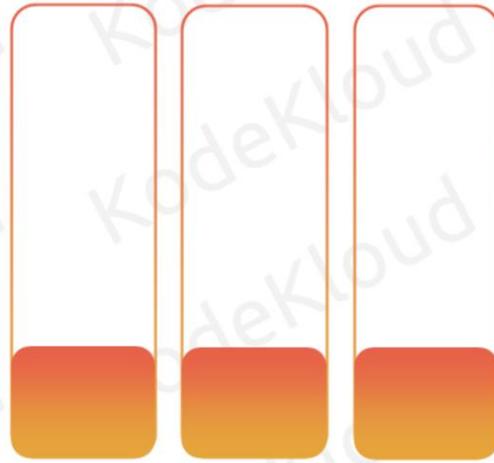


© Copyright KodeKloud

Memory Limitations:

JavaScript uses a garbage-collected memory management system. While this automates memory deallocation and reduces the risk of memory leaks, it's not always the most efficient.

Memory Limitations



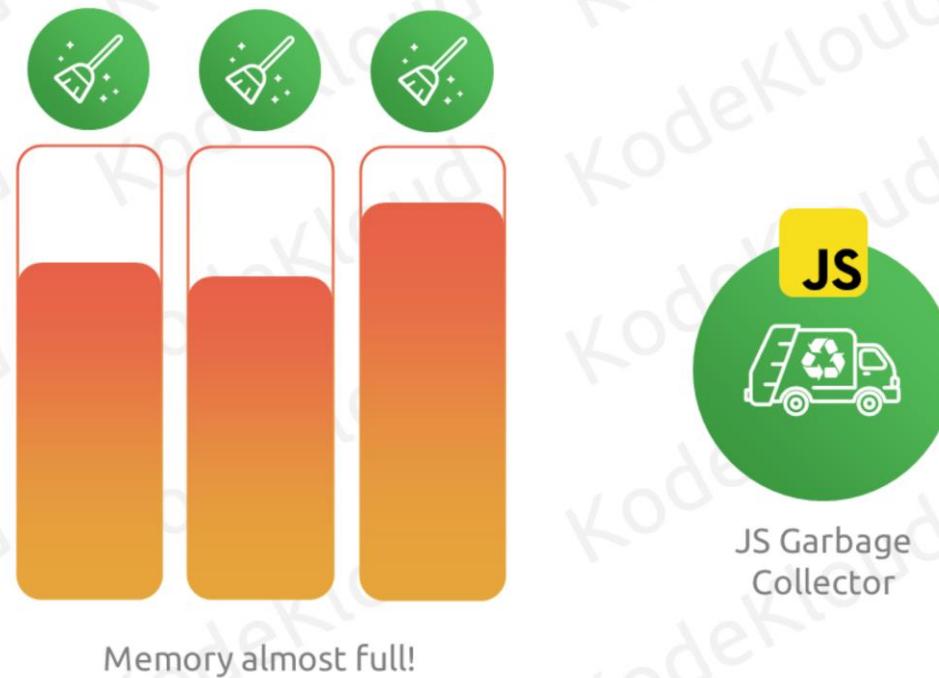
Memory Clean-up!



© Copyright KodeKloud

The garbage collector can introduce pauses in application execution, especially if there's a significant amount of memory to clean up.

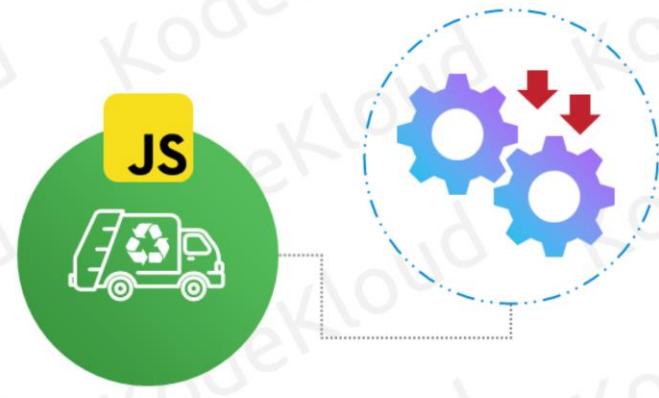
Memory Limitations



© Copyright KodeKloud

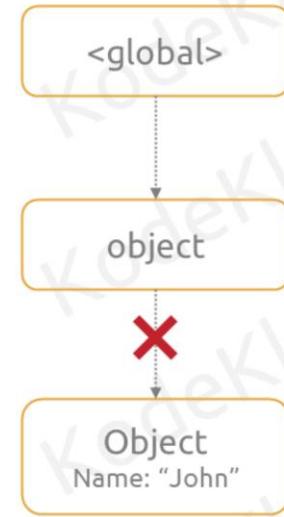
Additionally, JavaScript doesn't provide developers with direct access to memory management, which can be limiting when building large-scale applications or systems that require fine-tuned memory control.

Memory Limitations

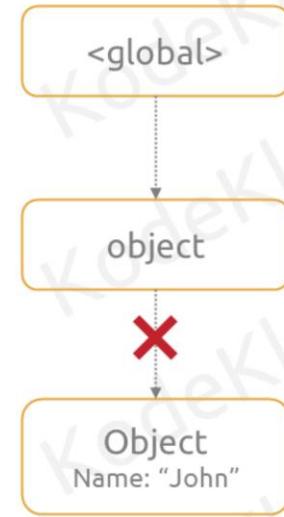
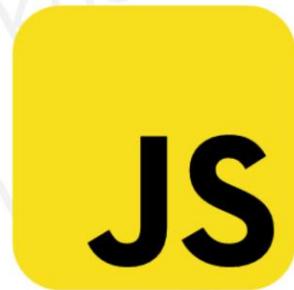


JS Garbage
Collector

Memory Limitations

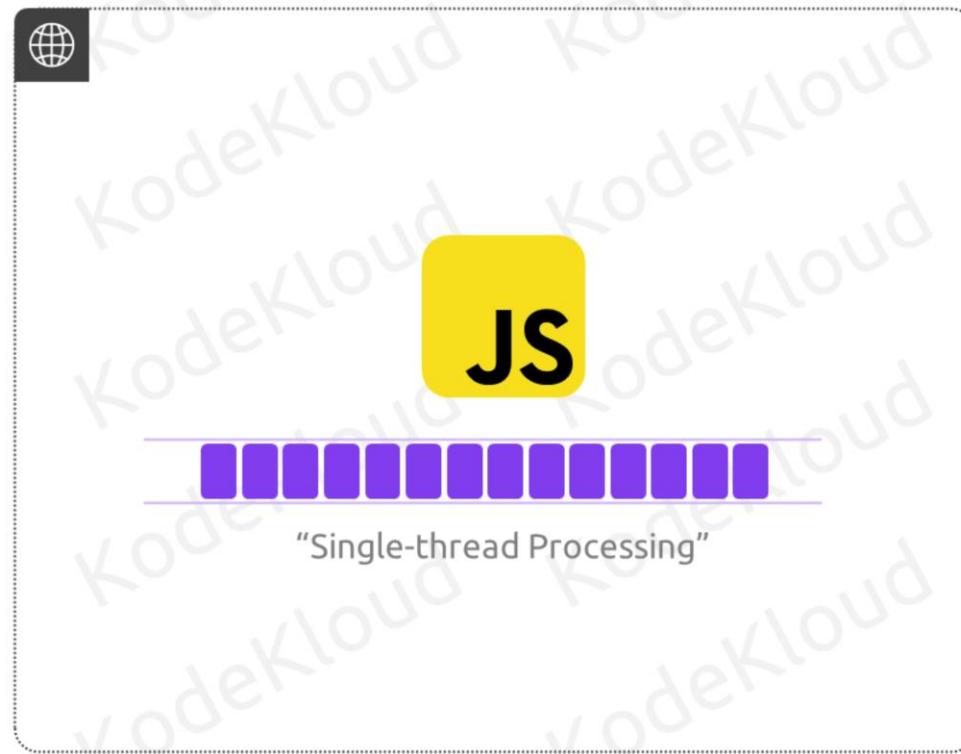


Memory Limitations





Concurrency Challenges

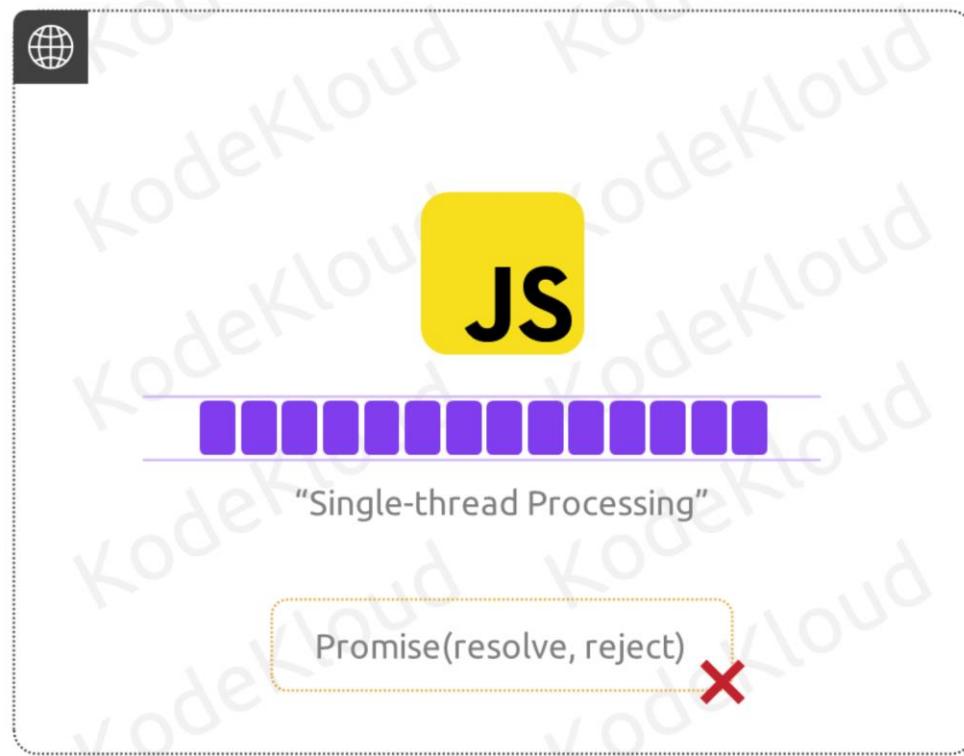


© Copyright KodeKloud

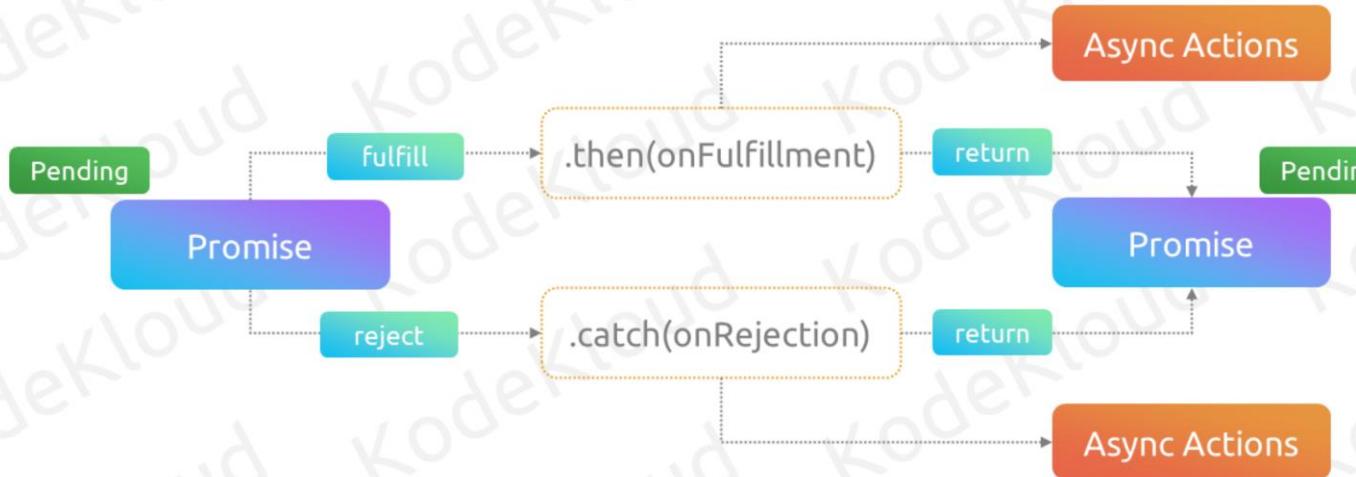
JavaScript operates in a single-threaded environment within the browser, meaning it processes one operation at a time in a single sequence.



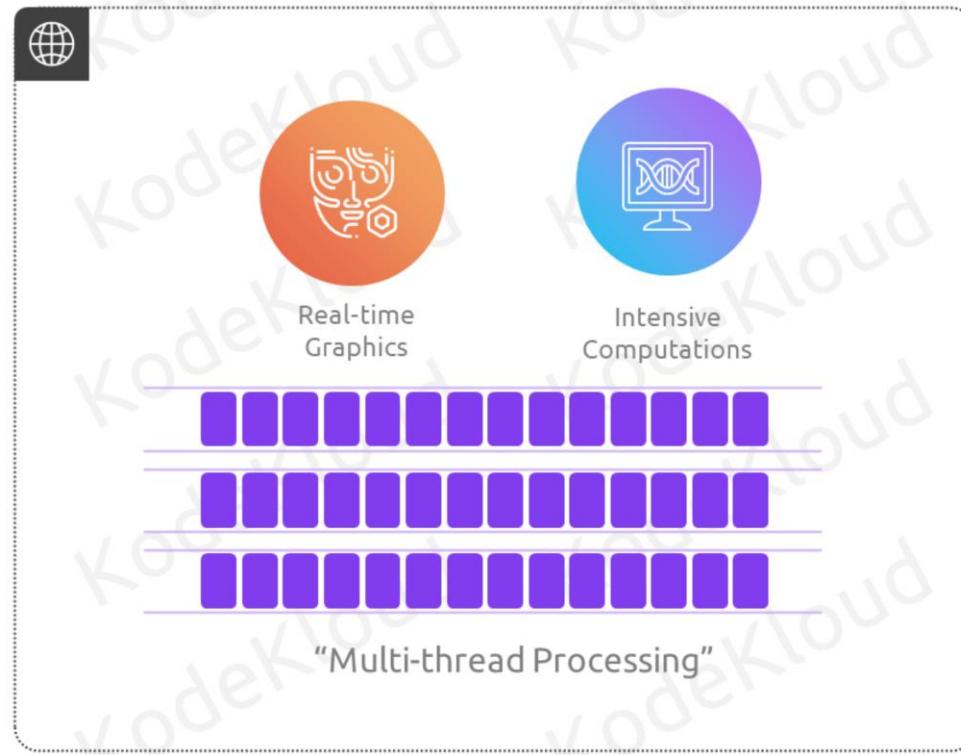
Concurrency Challenges



Concurrency Challenges



Concurrency Challenges



© Copyright KodeKloud

This means that while you can efficiently manage and sequence asynchronous operations, you can't truly run multiple operations simultaneously in parallel threads. This limitation becomes evident in applications that could benefit from multi-threaded processing, such as data-intensive tasks or real-time simulations.



© Copyright KodeKloud

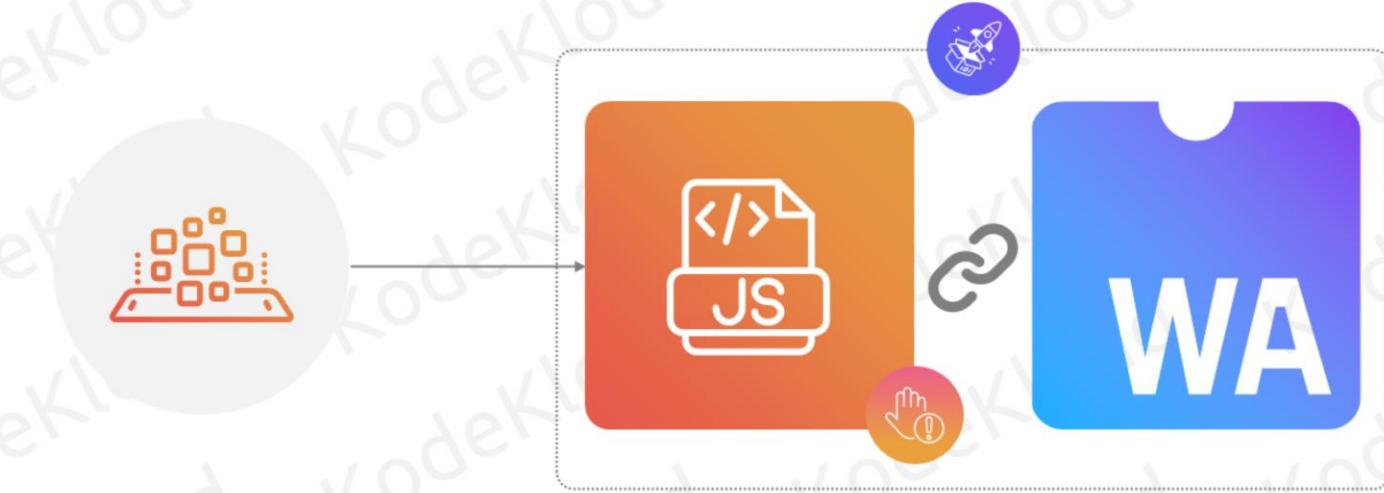
As the digital landscape evolved and the need for more performant web applications grew, it became clear that an alternative or complementary solution to JavaScript was necessary.

This is where WebAssembly, often referred to as "Wasm," seamlessly fits into the narrative.

WebAssembly (WASM) in Modern Development:

Benefits, Use Cases, and Real-World Adoption

Introduction



© Copyright KodeKloud

WebAssembly (WASM) has firmly established itself as a revolutionary force in the web development domain. As the digital landscape continues to evolve, the limitations of traditional tools like JavaScript have become more apparent. Enter WASM, a technology designed to complement and enhance the capabilities of existing web development tools.

WASM



01 | Benefits of WebAssembly



02 | Use Cases of WebAssembly



03 | Real-World Examples of WebAssembly Adoption

This lesson delves deep into the multifaceted advantages of WASM, its diverse applications, and its adoption by industry leaders.



Benefits of WebAssembly

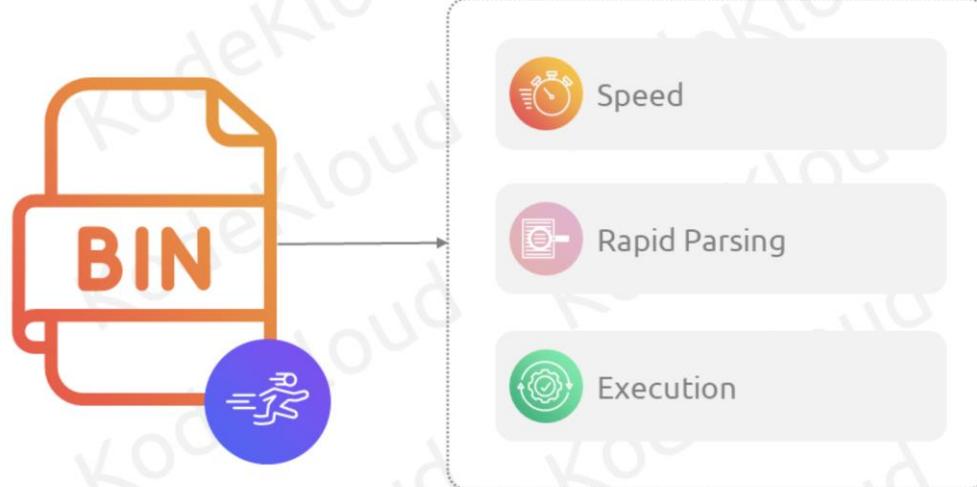
Performance Enhancement



© Copyright KodeKloud

Performance Enhancement: WASM works super fast, almost like native applications. This means web apps run smoothly, especially when they have heavy tasks, making everything feel snappy for the user.

Performance Enhancement



© Copyright KodeKloud

How It's Achieved: WASM is a low-level binary format, which means it's closer to machine code. This allows it to execute faster than traditional high-level languages. Moreover, its binary format is optimized for speed, ensuring rapid parsing and execution.



Language Versatility



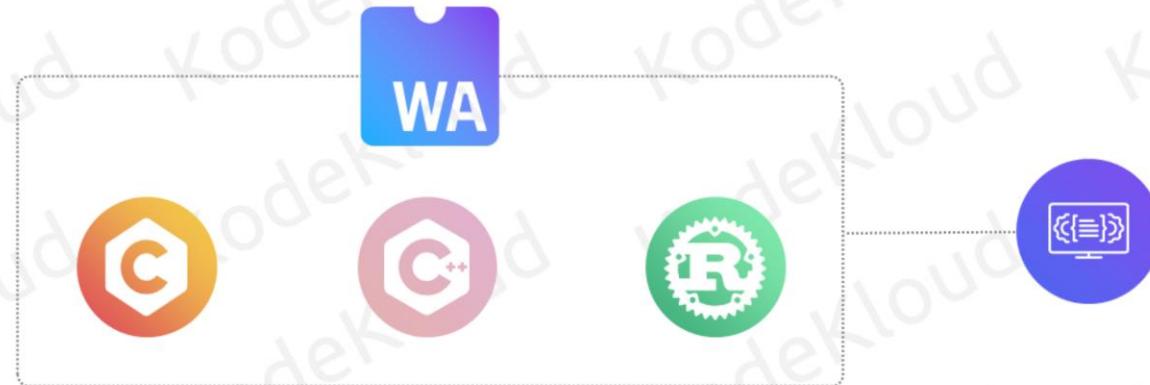
© Copyright KodeKloud

Language Versatility: With WASM, developers aren't stuck with just JavaScript. They can use other languages they're comfortable with, like C or Rust, and then run that code in browsers. It's like having more tools in your toolbox!

Language Versatility



Language Versatility



© Copyright KodeKloud

How It's Achieved: WASM acts as a compilation target for multiple languages. This means code from languages like C, C++, or Rust can be compiled into WASM and then run in the browser, bridging the gap between web and other programming languages.

Optimized Security



© Copyright KodeKloud

Optimized Security: WASM runs code in a "sandboxed environment." Think of this as a safe playground where code can operate without harming anything outside of it. This keeps web applications safer from potential threats.

Optimized Security





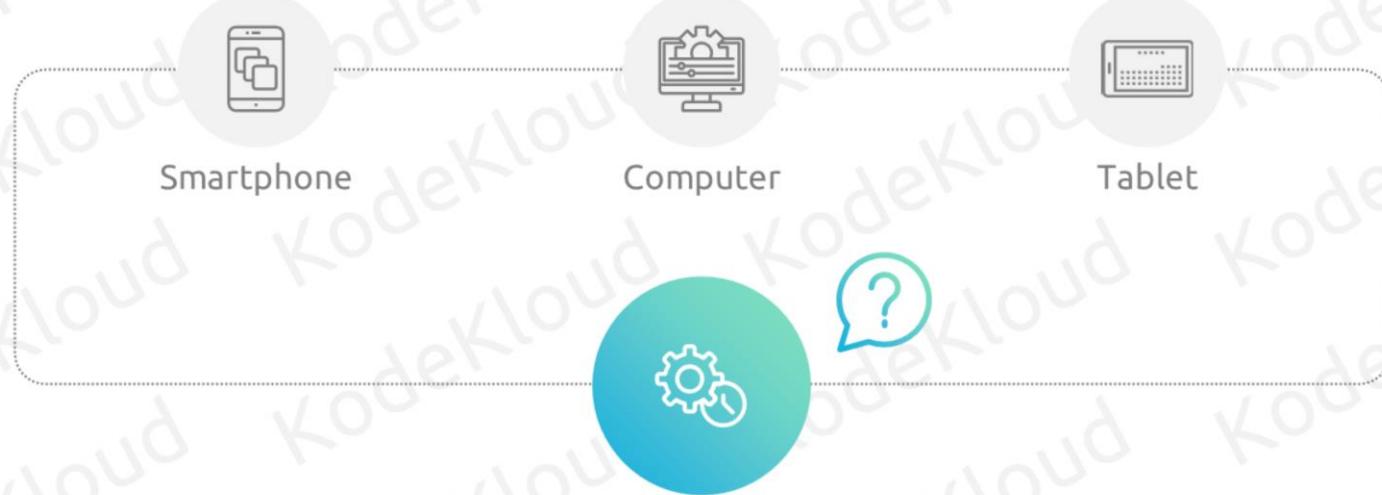
Optimized Security



© Copyright KodeKloud

How It's Achieved: The WASM architecture is designed to validate and execute code in a confined environment within the browser. This isolation ensures that even if malicious code is executed, it can't harm the broader system.

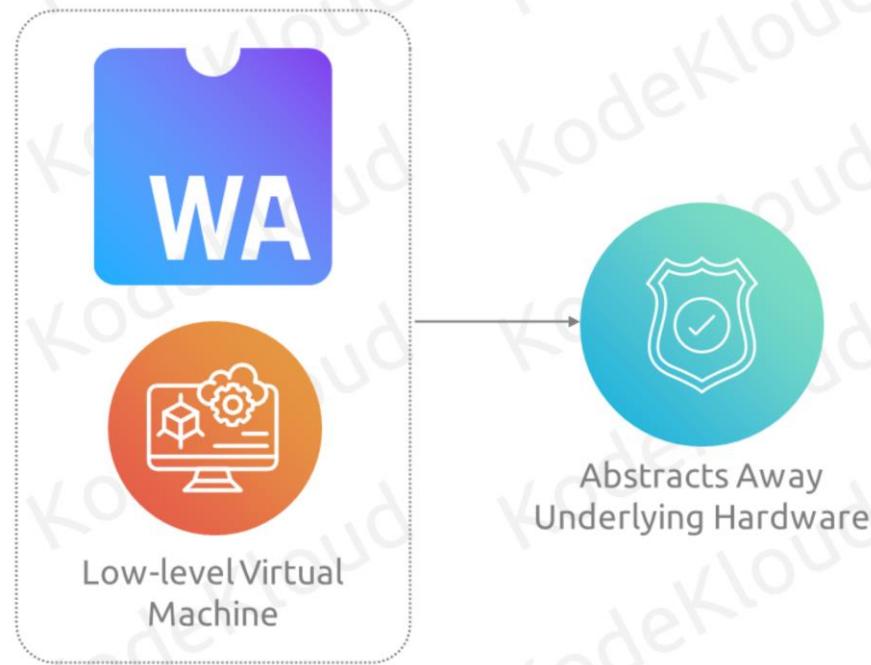
Consistent Portability



© Copyright KodeKloud

Consistent Portability: No matter where you run a WASM application—be it a smartphone, computer, or tablet—it's designed to work consistently and efficiently.

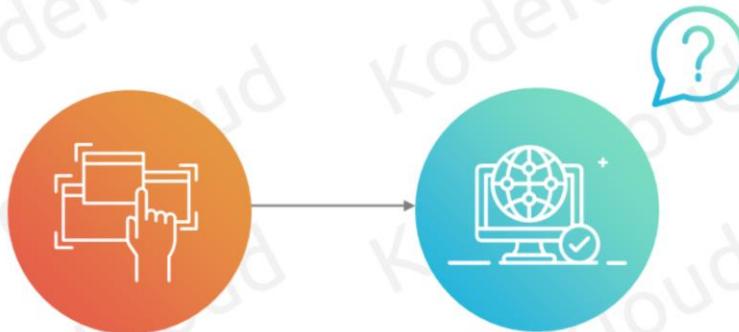
Consistent Portability



© Copyright KodeKloud

How It's Achieved: WASM's design as a low-level virtual machine ensures that it abstracts away the underlying hardware, allowing for consistent performance across various platforms.

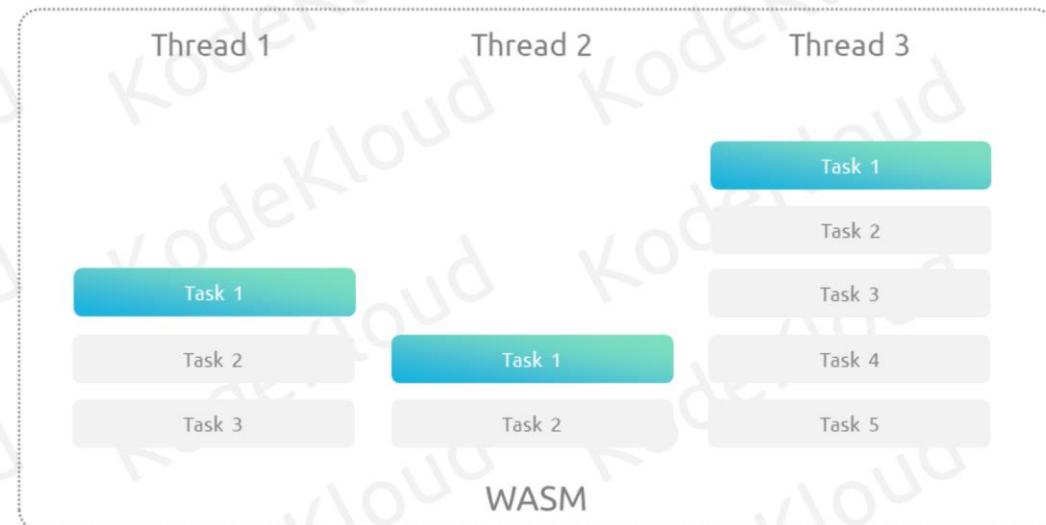
Concurrency and Parallelism



© Copyright KodeKloud

Concurrency and Parallelism: WASM is gearing up to handle multiple tasks at once, something JavaScript found challenging. This means smoother multitasking in web applications.

Concurrency and Parallelism



© Copyright KodeKloud

How It's Achieved: Future iterations of WASM aim to support threads, enabling it to execute multiple operations concurrently. This is in contrast to JavaScript, which is single-threaded and relies on event loops and callbacks for asynchronous operations.

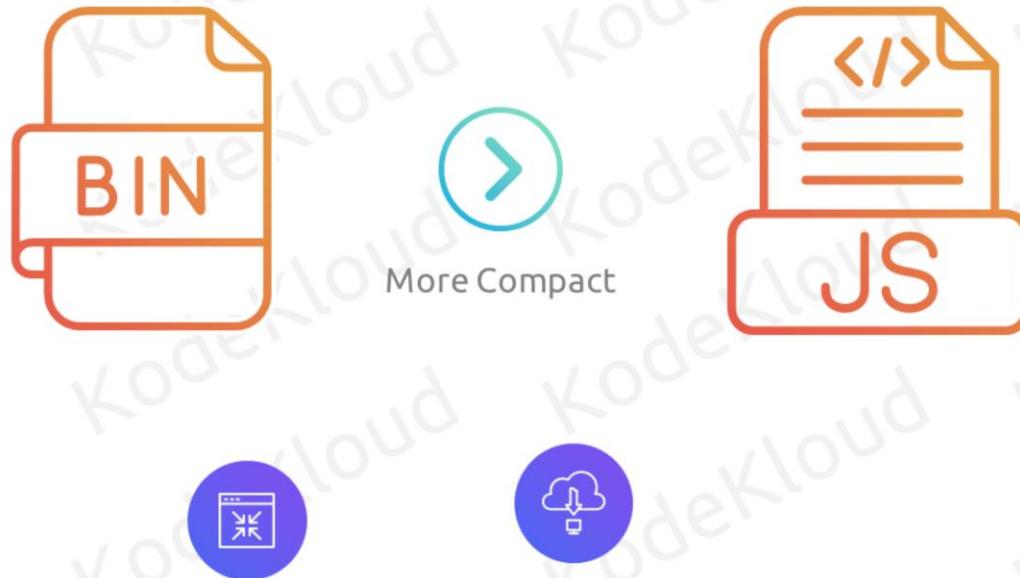
Compactness and Speed



© Copyright KodeKloud

Compactness and Speed: WASM's format is designed to be small and quick to decode. In simple terms, web applications using WASM will load and start faster, keeping users happy.

Compactness and Speed

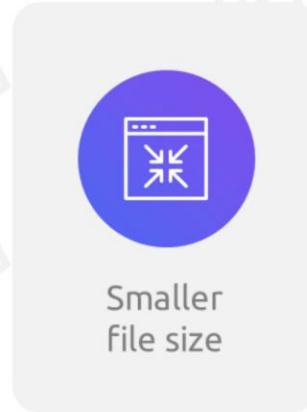


© Copyright KodeKloud

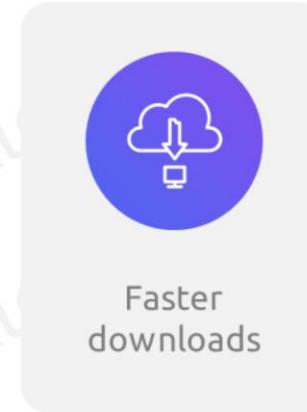
How It's Achieved: WASM's binary format is more compact than textual JavaScript. This means smaller file sizes and faster download times. Additionally, being a binary format, it's quicker for browsers to decode and execute compared to parsing textual JavaScript.



Compactness and Speed

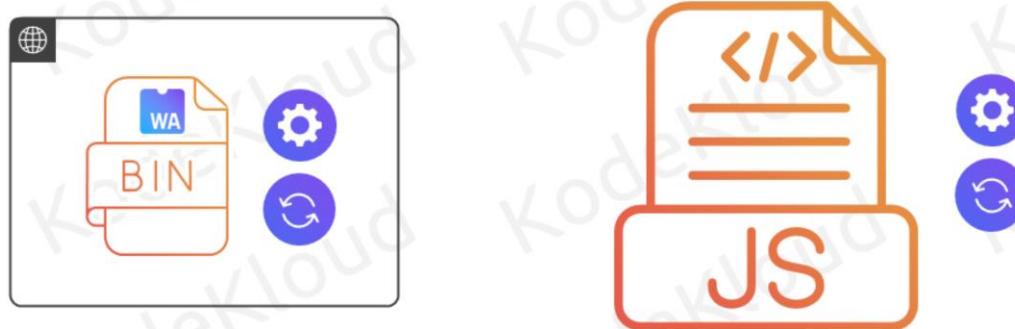


Smaller
file size



Faster
downloads

Compactness and Speed





Use Cases of WebAssembly

Machine Learning and AI



© Copyright KodeKloud

With the surge in AI-driven applications, WASM facilitates the efficient execution of machine learning models directly in browsers.



Machine Learning and AI



Real-time Predictions



Analyses



Decisions

© Copyright KodeKloud

This means applications can make real-time predictions, analyses, and decisions without the latency of server-side processing.

Cryptographic Operations



Cryptographic Computations



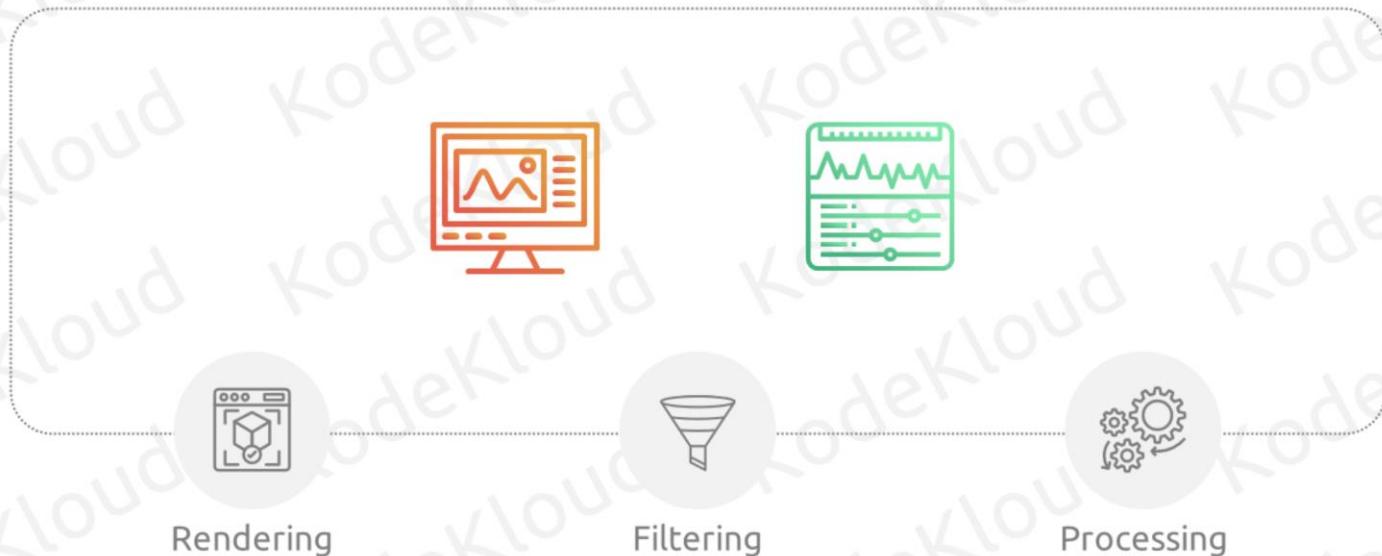
Ensuring Rapid and Secure Data Transactions

© Copyright KodeKloud

Security is paramount in the digital age. WASM accelerates cryptographic computations, ensuring rapid and secure data transactions, which is crucial for financial, healthcare, and other sensitive applications.



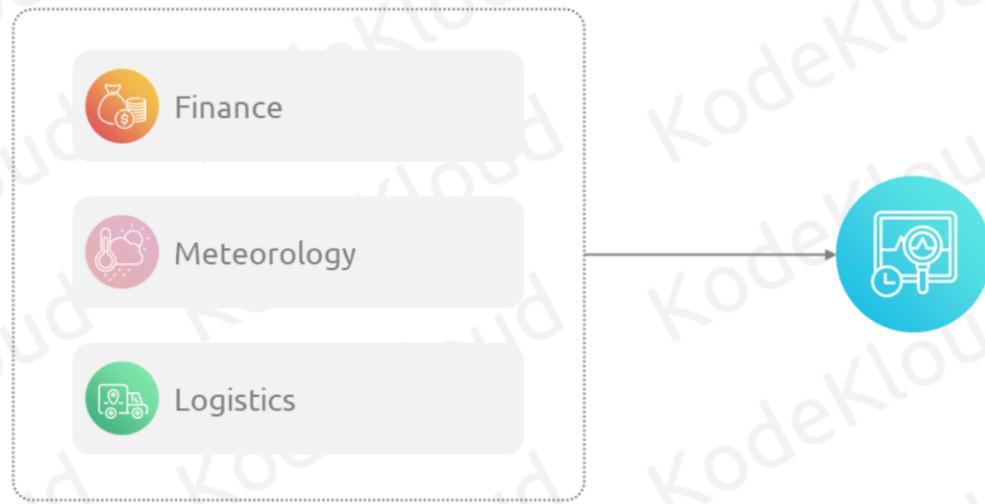
Interactive Multimedia Applications



© Copyright KodeKloud

From advanced photo editors to music creation tools, WASM's computational prowess ensures multimedia applications run smoothly, handling tasks like rendering, filtering, and processing with ease.

Real-time Data Visualization



© Copyright KodeKloud

In sectors like finance, meteorology, and logistics, real-time data visualization is crucial. WASM ensures datasets are processed and visualized seamlessly, aiding in swift decision-making.

3D Gaming and Simulations



Intensive Graphics Operations



Browser-based 3D Games



Simulation Run
Without Hitches

© Copyright KodeKloud

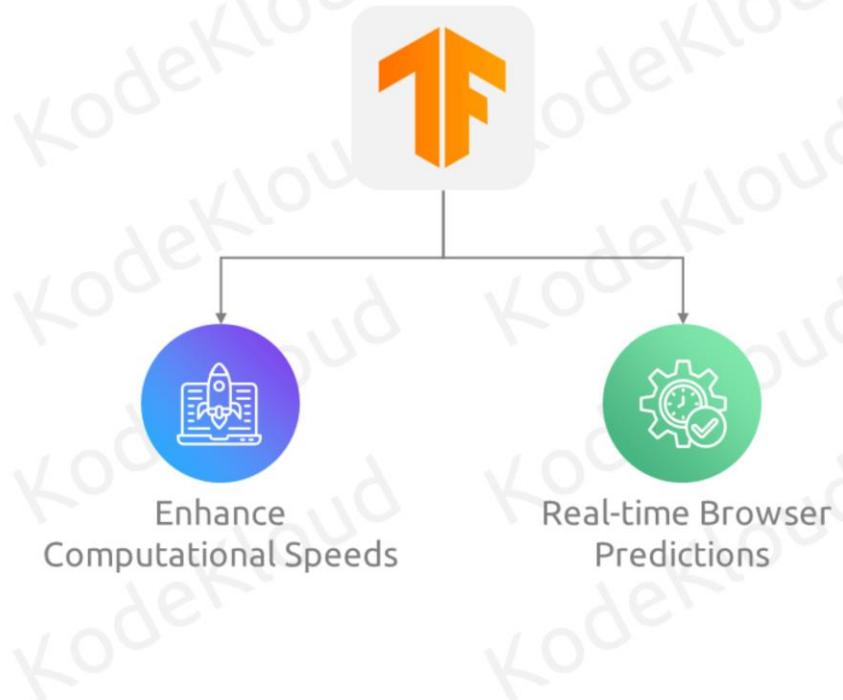
The gaming industry continually pushes technological boundaries. WASM caters to this by managing intensive graphics operations, ensuring browser-based 3D games and simulations run without hitches.



Real-World Examples of WebAssembly Adoption



Google's TensorFlow.js



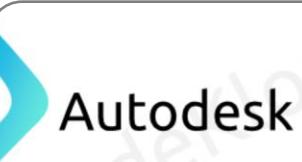
© Copyright KodeKloud

Google's machine learning library for JavaScript has integrated WASM to enhance computational speeds, making real-time browser predictions a reality.

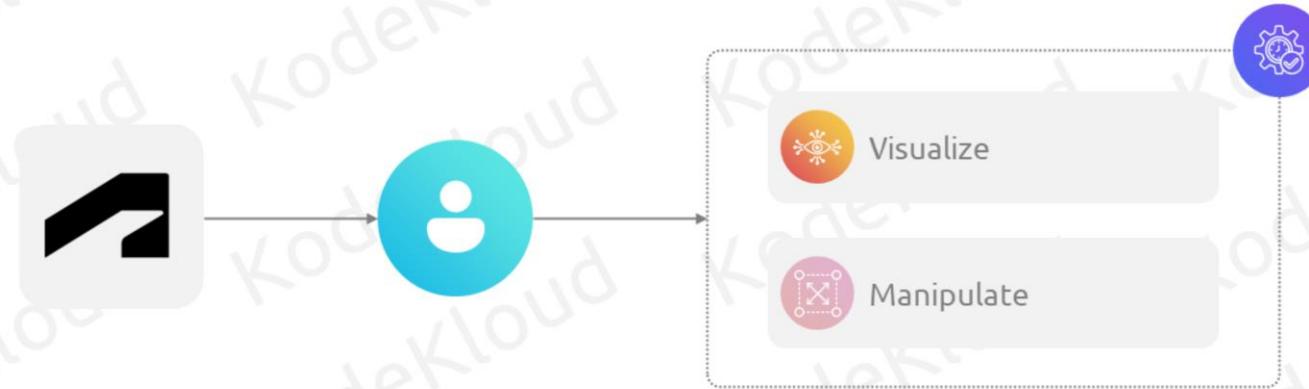


© Copyright KodeKloud

This design tool has harnessed WASM to ensure intricate design operations and rendering processes are fluid and responsive, enhancing the user experience for designers globally.



Autodesk



© Copyright KodeKloud

Recognizing the potential of WASM, Autodesk brought its CAD software, like AutoCAD, to the web, allowing professionals to visualize and manipulate complex designs in real-time.

Unity



© Copyright KodeKloud

A frontrunner in game development, Unity supports WASM, enabling developers to create high-performance 3D browser games, offering gamers a seamless experience.

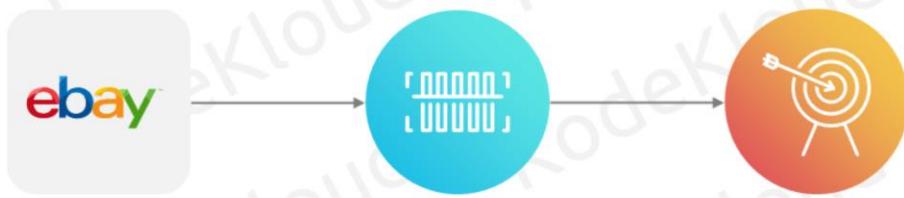


Cloudflare Workers



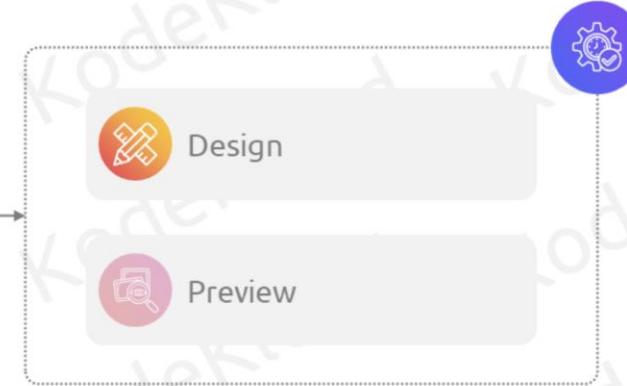
© Copyright KodeKloud

Cloudflare uses WASM to efficiently execute user-defined code at the edge, optimizing the speed and reliability of operations on its vast network.



© Copyright KodeKloud

The e-commerce giant employs WASM for barcode scanning in its web app, ensuring swift and accurate product searches.



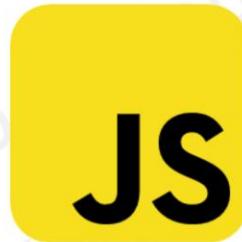
© Copyright KodeKloud

Leveraging WASM, Shopify has optimized its theme editor, allowing merchants to design and preview their online stores in real-time.

WebAssembly vs JavaScript



JavaScript and WASM

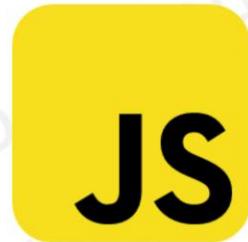


© Copyright KodeKloud

In the world of web development, there are two important technologies: Javascript and now WebAssembly (WASM). JavaScript is like an experienced sailor, while WebAssembly is a new explorer discovering uncharted territories. They both play crucial role on the web, but they have different strengths. Let's take a closer look at these two technologies with simple examples to understand them better.



JavaScript and WASM



© Copyright KodeKloud

JavaScript is a high-level, interpreted scripting language that has been the heart and soul of web interactivity for over two decades.



JavaScript and WASM

01

Ubiquity



02

Ecosystem



© Copyright KodeKloud

Major Strengths are:

Ubiquity: Every modern web browser understands and speaks JavaScript. It's like the universal language of the web.

Ecosystem: With its vast library and frameworks like react, angular, and vue, building anything from simple websites to applications becomes feasible.

Example

Hello! Sign up to get started!

Username

Email

Password

Confirm Password

Sign Up

```
...  
  
document.getElementById("myForm").addEventListener("submit",  
function(event) {  
  
let name = document.getElementById("name").value;  
if(name === "") {  
alert("Name cannot be empty");  
event.preventDefault();  
}  
});
```

Imagine creating a simple interactive form on a website.

Example

Hello! Sign up to get started!

User101@email.com

Sign Up

```
...  
  
document.getElementById("myForm").addEventListener("submit",  
function(event) {  
  
let name = document.getElementById("name").value;  
if(name === "") {  
    alert("Name cannot be empty");  
    event.preventDefault();  
}  
});
```

© Copyright KodeKloud

With just a few lines of JavaScript, you can validate user input in real time.

Example

Hello! Sign up to get started!

*Name cannot be empty

User101@email.com

Sign Up

```
...  
  
document.getElementById("myForm").addEventListener("submit",  
function(event) {  
  
let name = document.getElementById("name").value;  
if(name == "") {  
alert("Name cannot be empty");  
event.preventDefault();  
}  
});
```



WebAssembly (WASM)

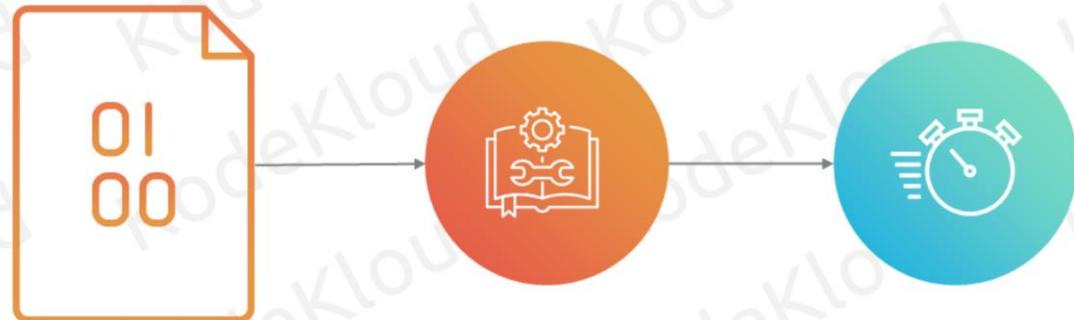


© Copyright KodeKloud

WebAssembly or WASM is a low-level binary format.



WebAssembly (WASM)



© Copyright KodeKloud

Think of it as a set of instructions that modern browsers can understand and execute at near-native speed.

WebAssembly – Major Strengths



Performance

Faster than JavaScript
for heavy computations



Flexibility

Write code in languages like C,
C++, or Rust, compile it to WASM,
and run it in the browser.

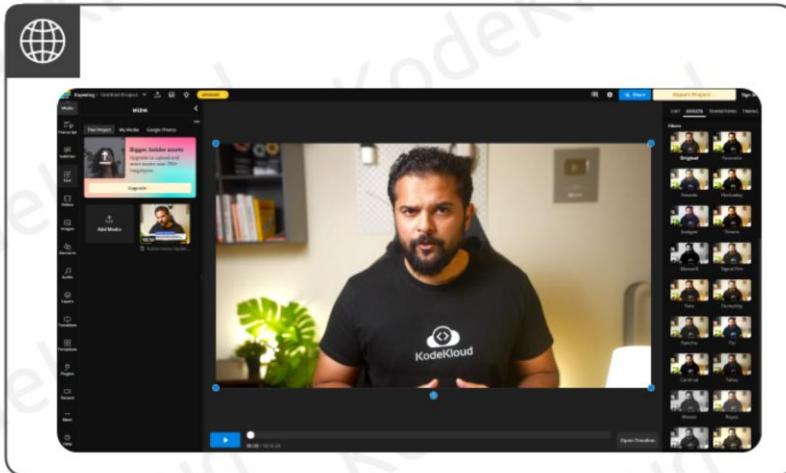
WebAssembly – Major Strengths

```
...  
// C code to apply a filter  
Void applyFilter(int* pixels, int length) {  
    for(int i = 0; i < length; i++) {  
        pixels[i] = /* complex filter operation */;  
    }  
}
```

© Copyright KodeKloud

Consider a web-based video editing tool that applies complex filters. Such heavy computations can be slow in JavaScript. But with WASM, the process can be significantly faster. Here's a simplified representation:
After writing this in C, you'd compile it to WASM, and then it can be executed in the browser, delivering a smoother video editing experience.

WebAssembly – Major Strengths



```
...  
  
// C code to apply a filter  
Void applyFilter(int* pixels, int length) {  
    for(int i = 0; i < length; i++) {  
        pixels[i] = /* complex filter operation */;  
    }  
}
```



© Copyright KodeKloud

Consider a web-based video editing tool that applies complex filters. Such heavy computations can be slow in JavaScript. But with WASM, the process can be significantly faster. Here's a simplified representation:

WebAssembly – Major Strengths

```
...  
// C code to apply a filter  
Void applyFilter(int* pixels, int length) {  
    for(int i = 0; i < length; i++) {  
        pixels[i] = /* complex filter operation */;  
    }  
}
```



© Copyright KodeKloud

After writing this in C, you'd compile it to WASM, and then it can be executed in the browser, delivering a smoother video editing experience.



How They Work Together



© Copyright KodeKloud

How They Work Together

WebAssembly and JavaScript, when combined, can create a powerful synergy in web development.

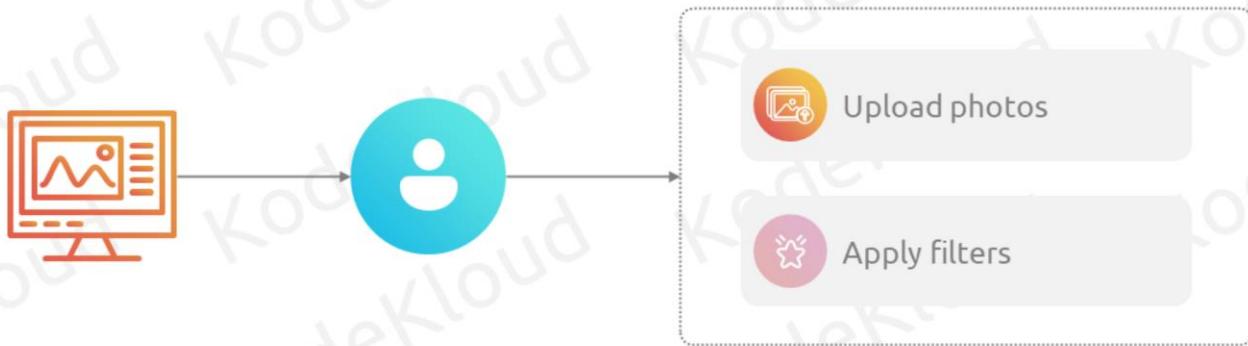
How They Work Together



© Copyright KodeKloud

Each brings its unique strengths to the table, and their collaboration can lead to efficient and dynamic web applications.

Example: Image Processing Web App



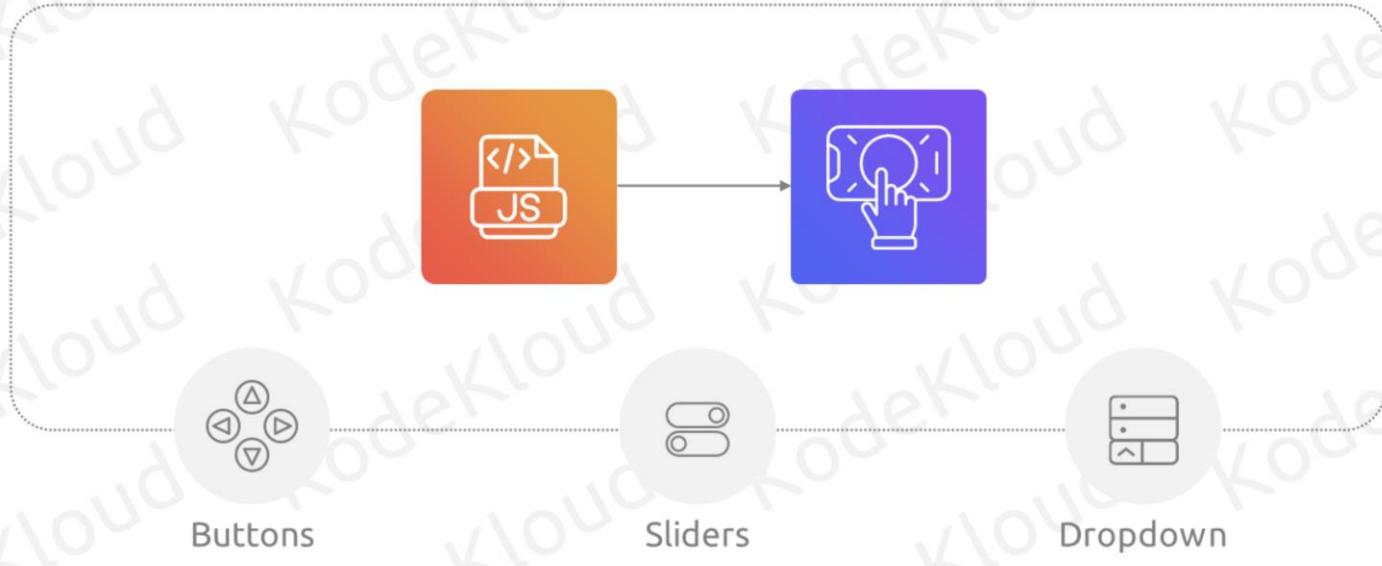
© Copyright KodeKloud

Imagine a web application designed for image processing, allowing users to upload photos and apply various filters.



JavaScript – Role

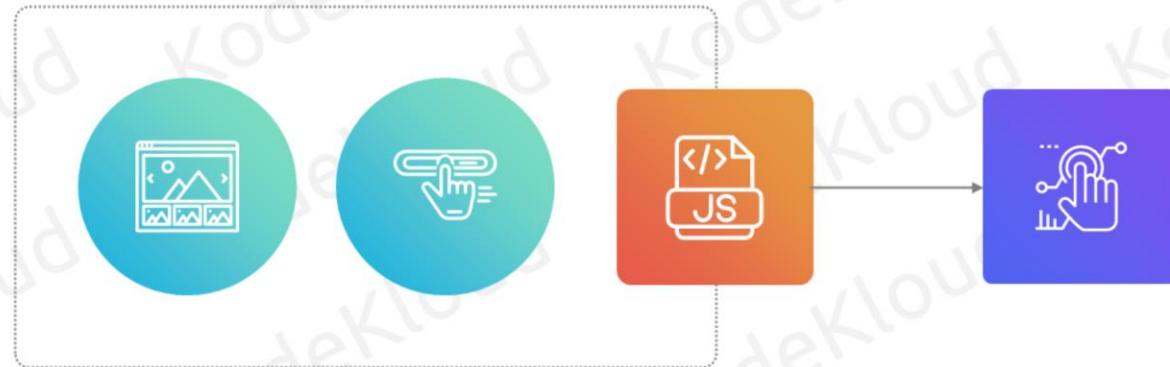
User Interface



© Copyright KodeKloud

JavaScript can be used to create the interactive UI of an app. This includes elements like buttons, sliders, and dropdown menus that users interact with.

Event Handling

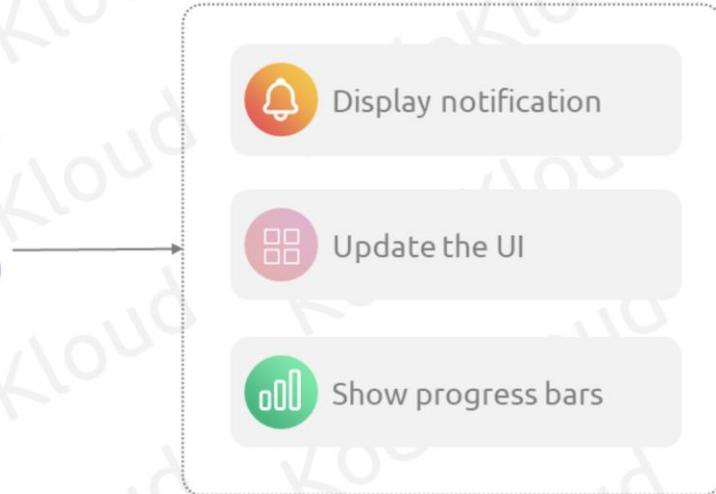


© Copyright KodeKloud

When a user selects a filter or adjusts a slider, JavaScript detects these actions and determines the appropriate response.



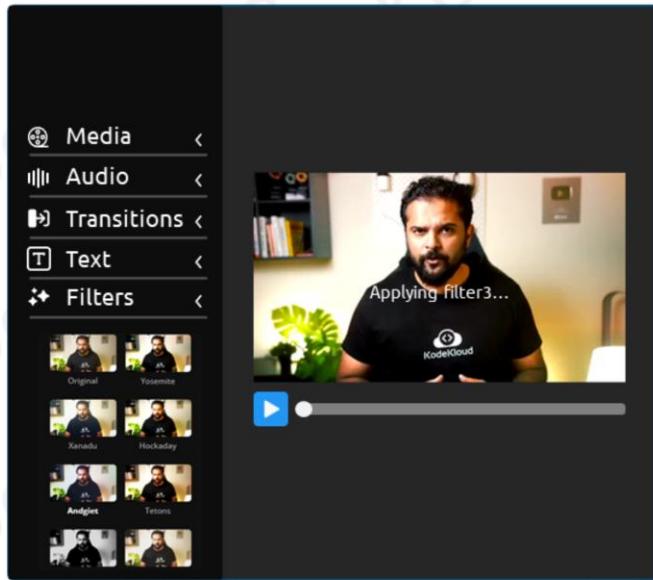
Feedback



© Copyright KodeKloud

After processing, JavaScript can display notifications, update the UI, or show progress bars.

Feedback



```
...  
  
document.getElementById("applyFilterBtn").addEventListener("click", function(){  
    let selectedFilter = document.getElementById("filterDropdown").value;  
    // Call the WASM function to process the image  
    if processImage = applyWASMFilter(selectedFilter);  
    displayImage(processedImage);  
});
```



WebAssembly – Role

Heavy Lifting



© Copyright KodeKloud

The actual image processing, which is computationally intensive, can be handled by WebAssembly. This ensures that filters are applied quickly and efficiently, without causing the web app to lag.

Optimized Algorithms



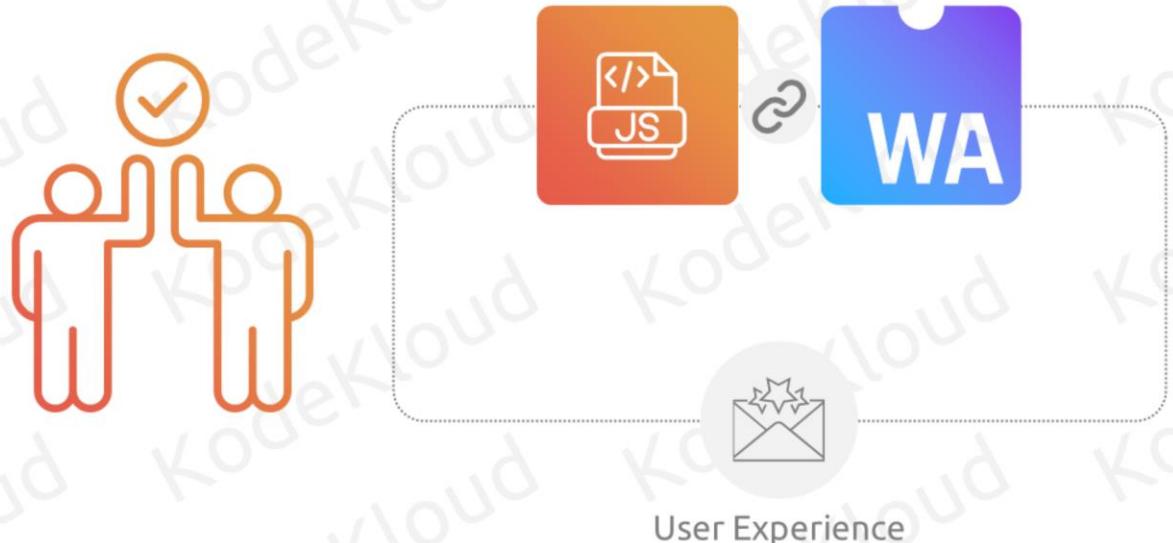
© Copyright KodeKloud

Complex algorithms, perhaps ported from existing C or C++ libraries, can be compiled to WASM and used to process the image data.

Optimized Algorithms

```
...  
  
// C code compiled to WASM for image processing  
Void applyWASMFilter(int* pixels, int length, Char* filterType) {  
    if (strcmp(filterType, "grayscale") == 0) {  
        for(int i + 0; i <length; i++) {  
            int avg = (pixels[i].r + pixels[i].g + pixels[i].b) / 3;  
            pixels[i].r = avg;  
            pixels[i].g = avg;  
            pixels[i].b = avg;  
        }  
        // Other filter algorithms. . .  
    }  
}
```

Dividing Responsibilities



© Copyright KodeKloud

By dividing responsibilities based on their strengths, JavaScript and WebAssembly can collaboratively deliver a seamless and responsive user experience, even in applications that require significant computational power.

Summary

- 01 Definition of WebAssembly and its role in web development
- 02 Explores the key benefits and practical use cases of WebAssembly
- 03 A comparative analysis with JavaScript is also provided to highlight the unique strengths and appropriate use cases of each
- 04 A quiz to reinforce these fundamental concepts

This section provides a foundational understanding of WebAssembly. It begins by defining what WebAssembly is and its role in web development. The section then explores the key benefits and practical use cases of WebAssembly, illustrating why it's an important skill for developers. A comparative analysis with JavaScript is also provided to highlight the unique strengths and appropriate use cases of each. The section concludes with a quiz to reinforce these fundamental concepts.



KodeKloud

© Copyright KodeKloud

Follow us on <https://kodekloud.com/> to learn more about us.