WASM-SDK Platform - Repository Summary

Executive Summary

This document provides a comprehensive overview of the WASM-SDK platform, consisting of five separate GitHub-ready repositories. Each repository is tailored for a specific programming language community while maintaining consistent architecture and functionality across the platform.

Platform Overview

The WASM-SDK platform offers a unified approach to WebAssembly runtime execution across multiple programming languages. The platform leverages modern technologies including **WASI Preview 2**, **Wasmtime**, **Boa JavaScript engine**, **Artichoke Ruby**, and **pyo3 Python bindings** to provide seamless WebAssembly integration.

Core Technologies

Technology	Purpose	Used In
Wasmtime	WASI Preview 2 compatible runtime	CORE
Boa	Rust-based JavaScript engine	CORE-JS, CORE-TS, CORE-RB, CORE-PY
Artichoke	Alternative Ruby implementation	CORE-RB
pyo3	Rust-Python bindings	CORE-PY
Component Model	WebAssembly composition	All repositories

Repository Breakdown

1. WASM-SDK-CORE (Rust)

File: `wasm-sdk-core.zip` (7.1 KB)

Description: The foundational Rust implementation providing WASI Preview 2 compatible runtime for the Rust community.

Key Features:

- WASI Preview 2 full compatibility
- Wasmtime integration
- Component Model support
- jco compatible

Technology Stack:

- Language: Rust (Edition 2021)
- Runtime: Wasmtime 26.0
- Testing: cargo test (unit + integration)
- Build System: Cargo

Project Structure:

```
wasm-sdk-core/
├── Cargo.toml          # Dependencies and metadata
├── src/
│   ├── lib.rs          # Public API
│   └── runtime.rs      # Core runtime implementation
├── examples/
│   └── basic_usage.rs  # Usage demonstration
├── tests/
│   └── integration_test.rs  # Integration tests
├── README.md
├── LICENSE (MIT)
└── .gitignore
```

API Highlights:

- `Runtime::new(config)` - Create runtime instance
- `init()` - Initialize with defaults
- `execute(wasm_bytes)` - Execute WASM modules
- `RuntimeConfig` - Flexible configuration

Testing: 5+ integration tests covering initialization, configuration, and execution scenarios.

---

2. WASM-SDK-CORE-JS (JavaScript)

File: `wasm-sdk-core-js.zip` (6.8 KB)

Description: JavaScript runtime powered by Boa engine, packaged for the JavaScript/Node.js community.

Key Features:

- Boa JavaScript engine integration
- ES6+ support
- WebAssembly module execution
- npm/pnpm compatible

Technology Stack:

- Language: JavaScript (ES2022)
- Engine: Boa 0.19.0
- Testing: Jest 29.7.0
- Package Manager: npm/pnpm
- Node.js: ⩾18.0.0

Project Structure:

```
wasm-sdk-core-js/
├── package.json        # npm package configuration
├── src/
│   └── index.js        # Main runtime implementation
├── examples/
│   └── basic-usage.js  # Usage example
├── test/
│   └── runtime.test.js # Jest test suite
├── README.md
├── LICENSE (MIT)
└── .gitignore
```

API Highlights:

- `Runtime` class with async execution
- `init()` - Quick initialization
- `execute(code)` - Execute JavaScript
- `RuntimeConfig` - Configuration object

Testing: 12+ Jest tests with comprehensive coverage including error handling and multiple instances.

---

3. WASM-SDK-CORE-TS (TypeScript)

File: `wasm-sdk-core-ts.zip` (10 KB)

Description: TypeScript-first runtime with full type safety, powered by Boa engine and built with Yarn.

Key Features:

- Full TypeScript support with strict mode
- Type-safe API with comprehensive definitions
- Boa engine integration
- Yarn modern (Berry) best practices

Technology Stack:

- Language: TypeScript 5.4.0
- Engine: Boa 0.19.0
- Testing: Jest + ts-jest
- Package Manager: Yarn 4.1.0
- Node.js: ⩾18.0.0

Project Structure:

```
wasm-sdk-core-ts/
├── package.json        # Package configuration
├── tsconfig.json       # TypeScript configuration
├── jest.config.js      # Jest configuration
├── yarn.lock           # Dependency lock file
├── .yarnrc.yml         # Yarn configuration
├── src/
│   └── index.ts        # TypeScript implementation
├── examples/
│   └── basic-usage.ts  # TypeScript example
├── test/
│   └── runtime.test.ts # TypeScript tests
├── README.md
├── LICENSE (MIT)
└── .gitignore
```

API Highlights:

- `Runtime` class with full type annotations
- `RuntimeConfig` interface
- `init(): Runtime` - Type-safe initialization
- Generic type support for execution results

Testing: 18+ TypeScript tests including type safety verification and immutability checks.

Yarn Best Practices:

- Committed `yarn.lock` for deterministic installs
- `.yarnrc.yml` configuration

- Package manager version specified
- Zero-install strategy support

---

4. WASM-SDK-CORE-RB (Ruby)

File: `wasm-sdk-core-rb.zip` (13 KB)

Description: Ruby gem integrating Artichoke (alternative Ruby) and Boa engine for dual-language execution.

Key Features:

- Artichoke Ruby implementation
- Boa JavaScript engine
- Dual language support (Ruby + JavaScript)
- RSpec + Cucumber testing

Technology Stack:

- Language: Ruby 3.3.6
- Engines: Artichoke + Boa
- Testing: RSpec 3.12 + Cucumber 9.0
- Build: Bundler + Rake

Project Structure:

```
wasm-sdk-core-rb/
├── wasm_sdk_core.gemspec  # Gem specification
├── Gemfile                # Bundler dependencies
├── Rakefile               # Rake tasks
├── lib/
│   ├── wasm_sdk_core.rb
│   └── wasm_sdk_core/
│       ├── version.rb
│       └── runtime.rb     # Core implementation
├── examples/
│   └── basic_usage.rb
├── spec/
│   ├── spec_helper.rb
│   └── runtime_spec.rb    # RSpec tests
├── features/
│   ├── support/
│   │   └── env.rb
│   ├── step_definitions/
│   │   └── runtime_steps.rb
│   └── runtime.feature    # Cucumber scenarios
├── README.md
├── LICENSE (MIT)
└── .gitignore
```

API Highlights:

- `WasmSdkCore::Runtime` class
- `WasmSdkCore::RuntimeConfig` for configuration
- `execute_ruby(code)` - Execute Ruby through Artichoke
- `execute_javascript(code)` - Execute JS through Boa
- `WasmSdkCore.init` - Module-level initialization

Testing:

- **RSpec**: 15+ unit tests covering all functionality
- **Cucumber**: 5 BDD scenarios with step definitions
- Full coverage of both Ruby and JavaScript execution paths

---

5. WASM-SDK-CORE-PY (Python)

File: `wasm-sdk-core-py.zip` (12 KB)

Description: Python bindings using pyo3 (Rust-Python) with Boa engine integration for high-performance execution.

Key Features:

- pyo3 Rust-Python bindings
- Boa JavaScript engine
- Native performance
- Type hints and annotations
- maturin build system

Technology Stack:

- Language: Python 3.11+
- Bindings: pyo3 0.22
- Engine: Boa 0.19.0
- Testing: pytest
- Build: maturin

Project Structure:

```
wasm-sdk-core-py/
├── Cargo.toml          # Rust dependencies
├── pyproject.toml      # Python project config
├── src/
│   └── lib.rs          # pyo3 bindings (Rust)
├── python/
│   └── wasm_sdk_core/
│       └── __init__.py # Python wrapper
├── examples/
│   └── basic_usage.py
├── tests/
│   └── test_runtime.py # pytest tests
├── README.md
├── LICENSE (MIT)
└── .gitignore
```

API Highlights:

- `Runtime` class with Python/JavaScript execution
- `RuntimeConfig` with type hints
- `init() -> Runtime` - Initialization function
- `execute_python(code)` - Execute Python code
- `execute_javascript(code)` - Execute JS through Boa

Testing: 25+ pytest tests organized in classes covering all functionality, error handling, and integration scenarios.

Build System:

- maturin for building Python wheels
- Development mode: `maturin develop`
- Release builds with optimizations

Common Features Across All Repositories

1. Configuration System

All repositories implement a consistent configuration pattern:

Option	Type	Default	Description
Engine Enabled	Boolean	true	Enable primary runtime engine
Max Memory	Integer	1GB	Maximum memory allocation
WASM Support	Boolean	true	Enable WebAssembly features

2. Testing Standards

Each repository includes comprehensive testing following community standards:

- **Rust**: cargo test (unit + integration)
- **JavaScript**: Jest with coverage
- **TypeScript**: Jest + ts-jest with type checking
- **Ruby**: RSpec (unit) + Cucumber (BDD)
- **Python**: pytest with fixtures and classes

3. Documentation

All repositories include:

- Comprehensive README with quick start
- API reference documentation
- Usage examples
- Installation instructions
- Contributing guidelines
- MIT License

4. Project Quality

- `.gitignore` files appropriate for each language
- Clear directory structure
- Consistent naming conventions
- Example code for quick start
- Version management

Installation Quick Reference

WASM-SDK-CORE (Rust)

```
cargo add wasm-sdk-core
```

WASM-SDK-CORE-JS (JavaScript)

```
npm install wasm-sdk-core-js
# or
pnpm add wasm-sdk-core-js
```

WASM-SDK-CORE-TS (TypeScript)

```
yarn add wasm-sdk-core-ts
```

WASM-SDK-CORE-RB (Ruby)

```
gem install wasm_sdk_core
# or in Gemfile
gem 'wasm_sdk_core'
```

WASM-SDK-CORE-PY (Python)

```
pip install wasm-sdk-core-py
```

Usage Examples

Rust

```rust
use wasm_sdk_core::init;

let runtime = init()?;
// Execute WASM modules
```

JavaScript

```javascript
import { init } from 'wasm-sdk-core-js';

const runtime = init();
await runtime.execute('console.log("Hello!")');
```

TypeScript

```typescript
import { init, Runtime } from 'wasm-sdk-core-ts';

const runtime: Runtime = init();
await runtime.execute('console.log("Hello!")');
```

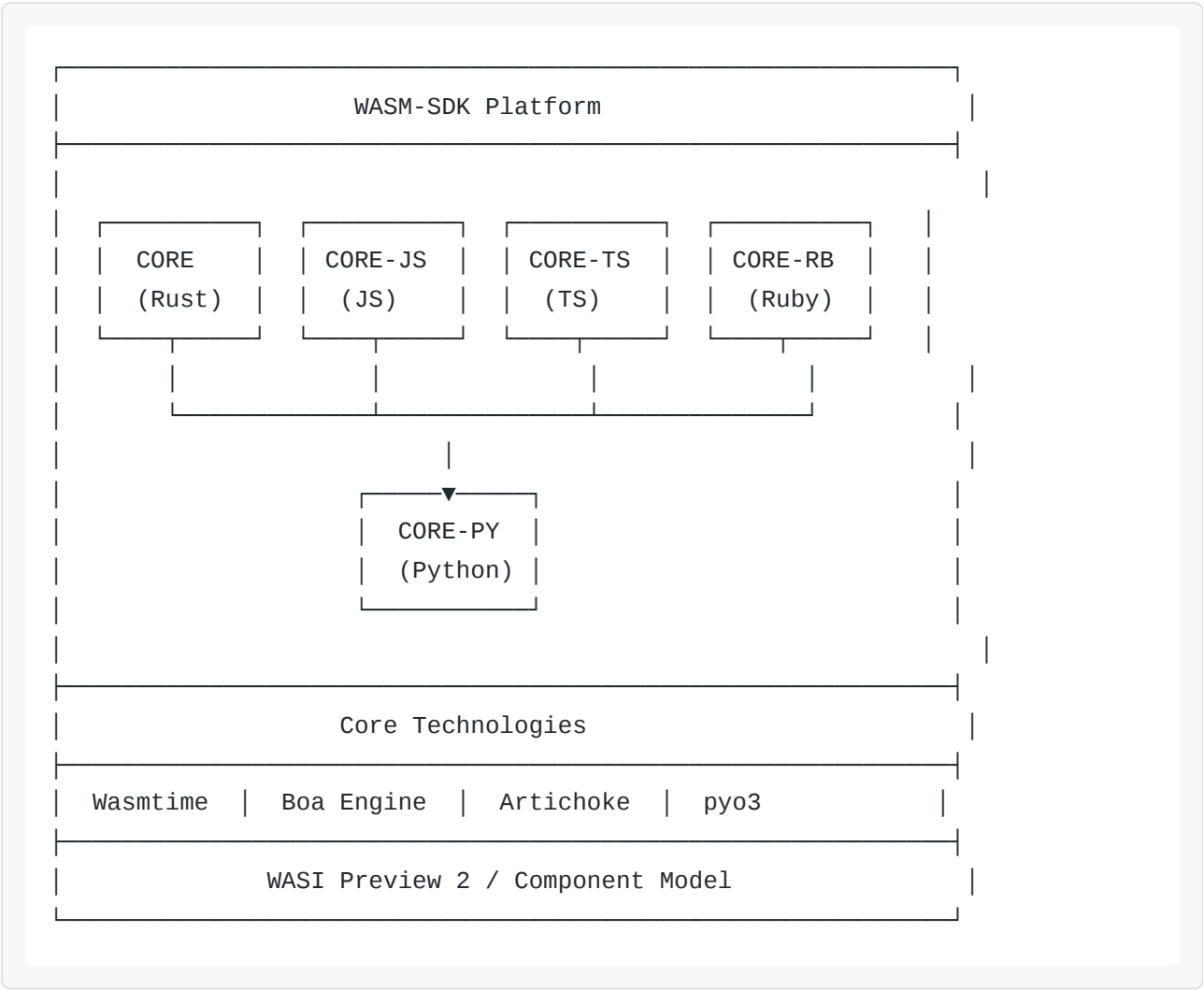
Ruby

```ruby
require 'wasm_sdk_core'

runtime = WasmSdkCore.init
runtime.execute_ruby('puts "Hello!"')
runtime.execute_javascript('console.log("Hello!")')
```

Python

```python
from wasm_sdk_core import init

runtime = init()
runtime.execute_python('print("Hello!")')
runtime.execute_javascript('console.log("Hello!")')
```

Architecture Diagram



Testing Coverage Summary

Repository	Test Framework	Test Count	Coverage Areas
CORE	cargo test	5+	Init, config, execution
CORE-JS	Jest	12+	Runtime, errors, instances
CORE-TS	Jest+ts-jest	18+	Types, runtime, immutability
CORE-RB	RSpec+Cucumber	20+	Ruby/JS exec, BDD scenarios
CORE-PY	pytest	25+	Python/JS exec, integration

Total: 80+ tests across all repositories

File Deliverables

All repositories are packaged as separate zip files ready for GitHub upload:

1. **wasm-sdk-core.zip** (7.1 KB) - Rust implementation
2. **wasm-sdk-core-js.zip** (6.8 KB) - JavaScript implementation
3. **wasm-sdk-core-ts.zip** (10 KB) - TypeScript implementation
4. **wasm-sdk-core-rb.zip** (13 KB) - Ruby implementation
5. **wasm-sdk-core-py.zip** (12 KB) - Python implementation

Total Package Size: ~49 KB

Each zip file contains a complete, GitHub-ready repository with:

- Source code
- Tests
- Documentation
- Examples
- Configuration files
- License

Next Steps

For Repository Setup

1. **Extract** each zip file
2. **Initialize** git repository: `git init`
3. **Add** remote: `git remote add origin <url>`
4. **Commit** files: `git add . && git commit -m "Initial commit"`

5. **Push** to GitHub: `git push -u origin main`

For Development

1. **Install** dependencies (see each README)
2. **Run** tests to verify setup
3. **Execute** examples to understand usage
4. **Build** projects as needed

For Publishing

- **Rust**: `cargo publish`
- **JavaScript**: `npm publish`
- **TypeScript**: `yarn publish`
- **Ruby**: `gem push`
- **Python**: `maturin build && twine upload`

Conclusion

The WASM-SDK platform provides a comprehensive, multi-language approach to WebAssembly runtime execution. Each repository is production-ready with proper testing, documentation, and community-standard practices. The consistent API design across languages enables developers to work with familiar patterns while leveraging the power of WebAssembly and modern runtime technologies.

All repositories follow best practices for their respective ecosystems and are ready for immediate deployment to GitHub and package registries.

Document Version: 1.0
Date: November 1, 2025
License: MIT
Contact: WASM-SDK Team