SingleStore

Using WebAssembly for in-database Machine Learning

Akmal B. Chaudhri

Agenda

Akmal B. Chaudhri

Senior Technical Evangelist
SingleStore

Agenda

- Introduction
- Machine Learning with a Database System
- Setup local Wasm development environment
- Demo: Large Movie Review

Introduction

SingleStoreDB

SingleStoreDB is a **real-time, distributed SQL database,** designed for modern applications and real-time analytics.

SingleStoreDB features a **unified data engine** that supports transactions, analytics and multiple data models, making it a compelling choice for use cases requiring several special purpose databases.

Machine Learning with a Database System

Apache Spark

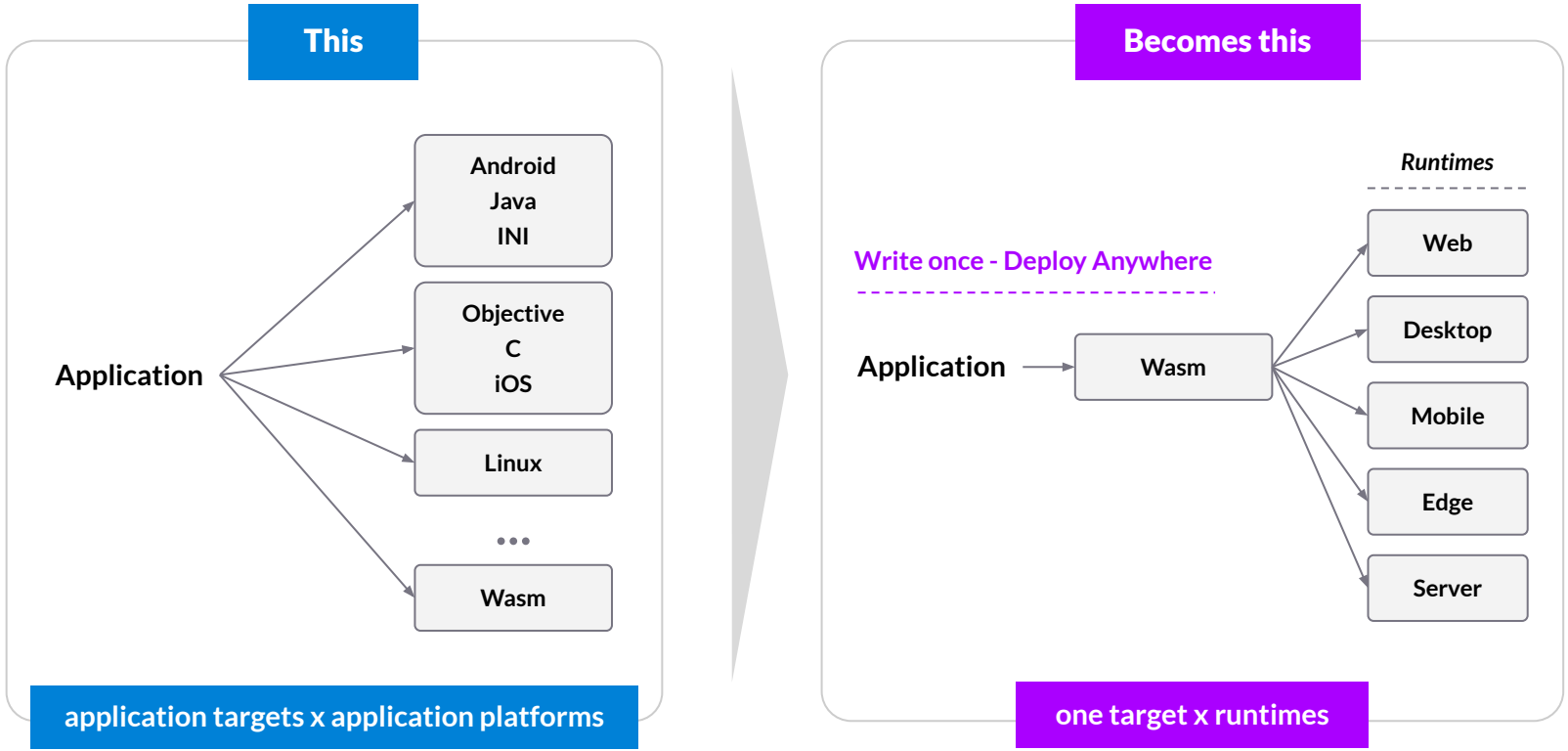
SingleStore Python Client
(+ ML Libraries)

SingleStore Vector Functions
(e.g. Euclidean Distance)

Code Engine using WebAssembly
(e.g. VADER)

OpenAI

Web Assembly: Write once, deploy anywhere



The Problem Wasm In-DB Functions Solve

- **Complex logic could only be done in app**
  - Sometimes writing it in Procedural SQL is too hard
  - e.g. ML scoring, fuzzy text matching
- **Moving lots of data to app is slow**
- **Coding query logic in app takes lots of developer time**
  - e.g. "find top-10 highest-scoring records"

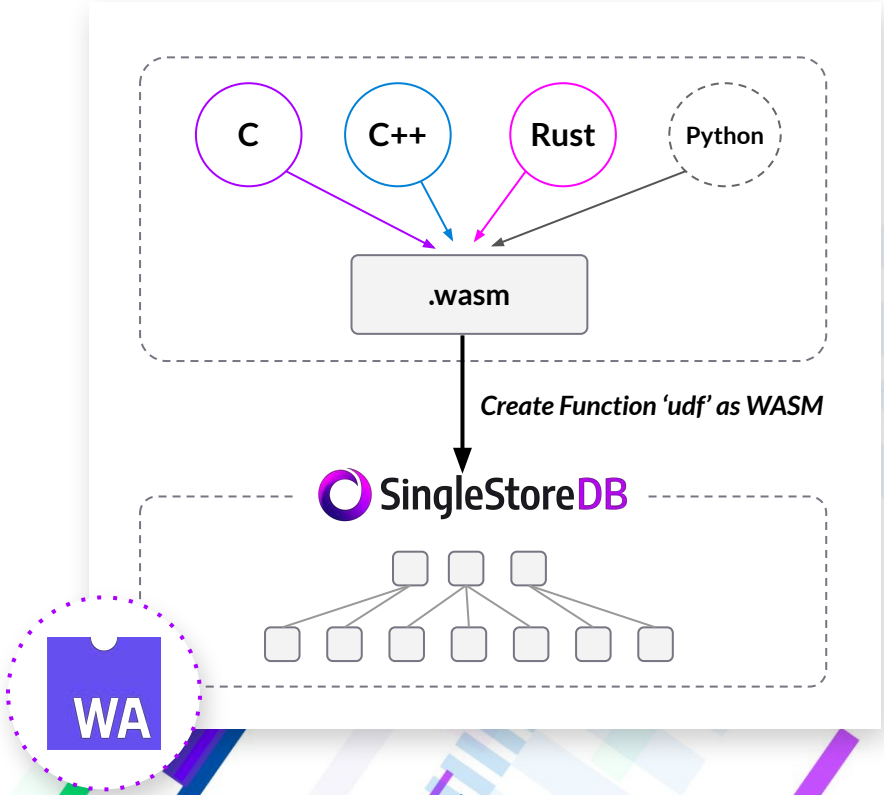
Code Engine - powered by Wasm

Bring or deploy existing libraries

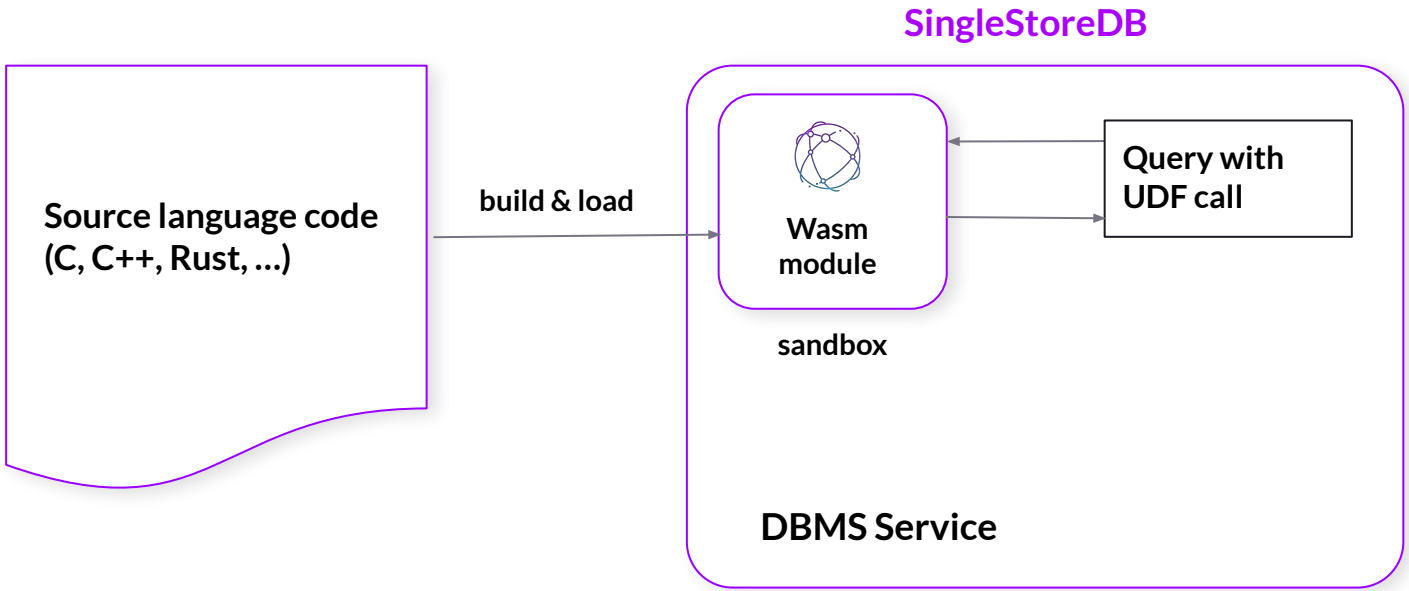
- **C, C++, Rust** programs and User Defined Functions (UDFs) can be brought natively into the database
- **Safe:** Code runs as a module in a sandbox
- **Robust ecosystem**: All major web browsers

Push app logic and compute down to the database tier

- **Fast:** Near-native performance, in-process
- **New real-time and ML use cases** enabled by data proximity to code



How Wasm Extensibility works



Setup local Wasm development environment

Install the Software

- Download `wasi-sdk`

  https://github.com/WebAssembly/wasi-sdk/releases

```
cd /opt
sudo cp /path/to/wasi-sdk-20.0-linux.tar.gz .
sudo tar xzvf wasi-sdk-20.0-linux.tar.gz
export PATH=/opt/wasi-sdk-20.0/bin:$PATH
```

Install the Software

- Install the `Rust` toolchain

```
curl --proto '=https' --tlsv1.2 -sSf \
https://sh.rustup.rs | sh

source "$HOME/.cargo/env"
```

- Install `wit-bindgen`

```
cargo install --git \
https://github.com/bytecodealliance/wit-bindgen \
wit-bindgen-cli
```

Install the Software

- Add `wasm32-wasi` to the `Rust` toolchain

```
rustup target add wasm32-wasi
```

- To deploy Wasm module to database use the pushwasm tool

```
git clone https://github.com/singlestore-labs/pushwasm
cd pushwasm
cargo build --release
export PATH=/path/to/pushwasm/target/release:$PATH
```

- May need to install `libssl`

```
sudo apt install libssl-dev
```

Initialise the source tree

- Create a new directory called `workdir` in our home folder

```
cd
mkdir workdir
cd workdir
```

- Create a skeletal `Rust` source tree

```
cargo init --vcs none --lib
```

Create the Interface Definition file

- Create a file called `sentimentable.wit`

```
record polarity-scores {
    compound: float64,
    positive: float64,
    negative: float64,
    neutral: float64,
}
sentimentable: func(input: string) ->
list<polarity-scores>
```

Implement and Compile

- Replace the existing contents of `Cargo.toml`

```
[package]

name = "sentimentable"

version = "0.1.0"

edition = "2021"

# See more keys and their definitions at
https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]

wit-bindgen-rust = { git = "https://github.com/bytecodealliance/wit-bindgen.git", rev =
"60e3c5b41e616fee239304d92128e117dd9be0a7" }

vader_sentiment = { git = "https://github.com/ckw017/vader-sentiment-rust" }

lazy_static = "1.4.0"

[lib]

crate-type = ["cdylib"]
```

Implement and Compile

- In the `lib.rs` file, we'll replace the existing contents with the following code:

```
wit_bindgen_rust::export!("sentimentable.wit");

use crate::sentimentable::PolarityScores;

struct Sentimentable;

impl sentimentable::Sentimentable for Sentimentable {


    fn sentimentable(input: String) -> Vec<PolarityScores> {
        lazy_static::lazy_static! {
            static ref ANALYZER: vader_sentiment::SentimentIntensityAnalyzer<'static> =
                vader_sentiment::SentimentIntensityAnalyzer::new();
        }
```

Implement and Compile

- In the `lib.rs` file, we'll replace the existing contents with the following code:

```
let scores = ANALYZER.polarity_scores(input.as_str());

        vec![PolarityScores {

            compound: scores["compound"],

            positive: scores["pos"],

            negative: scores["neg"],

            neutral: scores["neu"],

        }]

    }

}
```

Implement and Compile

- Build the Wasm module

```
cd ..

cargo build --target wasm32-wasi --release
```

- Connect

```
mysql --local-infile -u admin -h <host> -P 3306 \
--default-auth=mysql_native_password -p
```

- Create Database

```
CREATE DATABASE demo;

USE demo;
```

Deploy Wasm

- Use the pushwasm tool to push our Wasm module into SingleStoreDB

```
pushwasm tvf --prompt \
--name sentimentable \
--wit ./sentimentable.wit \
--wasm ./target/wasm32-wasi/release/sentimentable.wasm \
--conn mysql://admin@<host>:3306/demo


Wasm function was created successfully.
```

Run in the Database

- Quick test

```
SELECT * FROM sentimentable('The movie was great');
+--------------------+--------------------+----------+--------------------+
| compound           | positive           | negative | neutral            |
+--------------------+--------------------+----------+--------------------+
| 0.6248933269389457 | 0.5774647887323944 |        0 | 0.4225352112676057 |
+--------------------+--------------------+----------+--------------------+
```

- VADER can consider capitalisation

```
SELECT * FROM sentimentable('The movie was GREAT!');
+--------------------+--------------------+----------+--------------------+
| compound           | positive           | negative | neutral            |
+--------------------+--------------------+----------+--------------------+
| 0.7290259049799065 | 0.6307692307692307 |        0 | 0.3692307692307692 |
+--------------------+--------------------+----------+--------------------+
```

Demo: Large Movie Review

Summary

- Wasm = Web Assembly
- Wasm UDFs provide great power, extensibility
- Extend system in C/C++, Rust, and many more
- Fast
- Safe

Resources

- Turbocharge your application development using WebAssembly with SingleStoreDB
- Bytecode Alliance

SingleStore

Thank You

Offices

San Francisco (HQ)
534 Fourth Street
San Francisco, CA 94107

Additional locations:
Boston, Portland, Seattle, Sunnyvale, Raleigh, London, Dublin, Lisbon, India, Kyiv, and Singapore

Contact

Call us: 1-855-463-6775
Email us: team@singlestore.com