Fun with Fermyon

Serverless WebAssembly Compiled from Rust

David Wallace Croft, M.Sc.

Presented to the

Dallas Rust User Meetup

2024 Jan 09 Tue

WebAssembly

- WebAssembly (Wasm) is bytecode
  - Like Java bytecode but for the browser
- Many languages compile to Wasm
  - Rust might have the best support
- Wasm also runs on the server
  - Just as Java made the jump from applets
  - Just as JavaScript made the jump to Node.js

Fermyon Spin

- Fermyon Spin runs Wasm components
  - Compiled from Rust and other languages
- Serverless
  - Function as a Service (FaaS)
  - Like Amazon Web Services (AWS) Lambda
  - Fast cold starts

Fermyon Advent of Spin 2023

- Holiday-themed Spin-based code challenges
- Challenge 1
  - Static file server and data persistence
- Challenge 2
  - Knapsack algorithm
- Challenge 3
  - Large Language Model (LLM)
- Challenge 4
  - Bulls and Cows game

Approach

- Multiple code examples available from Fermyon
  - Blogs, Documentation, GitHub, YouTube
- Code assistant
  - Started using Amazon CodeWhisperer
- Submit the code
  - Returns success or failure
- Move on
  - Assume correct once submitted

Challenge 1

- Serve a holiday-themed static webpage
  - Uses a pre-compiled Wasm component
  - One complication due to using Windows
  - Deployed from GitHub CodeSpaces
- Persist data in a key-value store
  - Saves to a local file when testing locally
  - Uses a default store when deployed to Cloud

Challenge 1 Workarounds

- Some workarounds required when on Windows
  - And using a precompiled Wasm component
  - Such as the static fileserver component
- This issue might already be fixed
  - https://github.com/fermyon/spin/issues/2112
- See my README.md files for the workarounds
  - https://github.com/david-wallace-croft/advent-of-spin/tree/main/2023/challenge1
  - https://github.com/david-wallace-croft/advent-of-spin/blob/main/README.md

Challenge 1 spin.toml Excerpt

```
[component.spin-static-fs]
files = [{ source = "assets", destination = "/" }]

# https://github.com/fermyon/spin/issues/2112

source = { url =
"https://github.com/fermyon/spin-fileserver/releases/download/v0.2.1/spin_static_fs.wasm",
digest = "sha256:5f05b15f0f7cd353d390bc5ebffec7fe25c6a6d7a05b9366c86dcb1a346e9f0f" }

# source = "../../../spin-fileserver/target/wasm32-wasi/release/spin_static_fs.wasm"

[[trigger.http]]
route = "/data"
component = "data"

[[trigger.http]]
route = "/..."
component = "spin-static-fs"
```

Challenge 1 Static Assets

```
$ pwd
/c/Users/David/git/croftsoft/rust/advent-of
-spin/2023/challenge1/assets

$ ls
index.html  santa-claus.jpg  stylesheet.css
```

Challenge 1 Code

```
#[http_component]
fn handle_request(
  req: http::Request<Vec<u8>>
) -> anyhow::Result<impl IntoResponse> {
  let store = Store::open_default()?;
  let (status, body) = match *req.method() {
    Method::POST => {
      store.set(req.uri().path(), req.body().as_slice())?;
      (StatusCode::CREATED, None)
    },
[...other REST methods corresponding to CRUD operations]
    _ => (StatusCode::METHOD_NOT_ALLOWED, None),
  };
  let response = Response::builder()
    .body(body)
    .header("Content-Type", "application/json")
    .status(status)
    .build();
  Ok(response)
}
```

Challenge 2

- Knapsack algorithm
  - Maximize value of integer-sized items that fit
  - Dynamic programming
- Implementation provided by AWS CodeWhisperer
  - Automatically wired up inputs to function
  - Concise code
- Tested by submitting
  - Then studied the code for two hours after

Challenge 2 serde-json

```rust
#[derive(Deserialize)]
struct Input {
  capacity: usize,
  kids: Vec<usize>,
  weight: Vec<usize>,
}

#[derive(Serialize)]
struct Output {
  kids: usize,
}

impl IntoBody for Output {
  fn into_body(self) -> Vec<u8> {
    serde_json::to_string(&self).unwrap().into_body()
  }
}
```

Challenge 2 Knapsack

```rust
fn knapsack(
  capacity: usize,
  kids: &[usize],
  weight: &[usize],
) -> usize {
  let mut knapsack = vec![0; capacity + 1];
  for i in 0..kids.len() {
    for j in (weight[i]..=capacity).rev() {
      knapsack[j] = knapsack[j].max(knapsack[j - weight[i]] + kids[i]);
    }
  }
  knapsack[capacity]
}
```

Challenge 3

- Large Language Model (LLM) story generation
  - Generative Artificial Intelligence (AI)
  - Uses Cloud Graphics Processing Units (GPUs)
- Static types required input parameters
  - Not sure what reasonable defaults would be
  - Used Amazon CodeWhisperer suggestions
- Tweaked the prompts that I provided
  - To integrate the inputs from the user

Challenge 3 spin.toml Excerpt

```
[component.confabulator]
ai_models = ["llama2-chat"]
allowed_outbound_hosts = []
source = "confabulator/target/wasm32-wasi/release/confabulator.wasm"

[component.confabulator.build]
command = "cargo build --target wasm32-wasi --release"
watch = ["src/**/*.rs", "Cargo.toml"]
workdir = "confabulator"

[[trigger.http]]
component = "confabulator"
route = "/..."
```

Challenge 3 Prompt

```rust
fn make_prompt(
  characters: &[String],
  objects: &[String],
  place: &str,
) -> String {
  let mut prompt = "Tell an engaging Christmas story. \
    The story should have a happy ending. \
    The story should have a theme of joy. \
    The story should be between 250 and 500 words long. "
    .to_owned();
  prompt.push_str(&format!(
    "The story should take place in the following location: {}. ",
    place
  ));
  prompt.push_str(&make_include_prompt(characters, "characters", "character"));
  prompt.push_str(&make_include_prompt(objects, "objects", "object"));
  prompt
}
```

Challenge 3 LLM Call

```rust
fn confabulate(
  characters: &[String],
  objects: &[String],
  place: &str,
) -> Output {
  let prompt = make_prompt(characters, objects, place);
  let options = llm::InferencingParams {
    max_tokens: 1000,
    repeat_penalty: 1.2,
    repeat_penalty_last_n_token_count: 0,
    temperature: 0.7,
    top_k: 0,
    top_p: 1.0,
  };
  let infer_result: Result<InferencingResult, spin_sdk::llm::Error> =
    llm::infer_with_options(
      llm::InferencingModel::Llama2Chat,
      &prompt,
      options,
    );
  let result = match &infer_result {
    Ok(inferencing_result) => format!("{:?}",
inferencing_result),
    Err(error) => format!("Error: {:?}", error),
  };
  let story = match infer_result {
    Ok(inferencing_result) => inferencing_result.text,
    Err(_error) => String::new(),
  }
  .trim()
  .to_owned();
  Output {
    prompt,
    result,
    story,
  }
}
```

Challenge 4

- Bulls and Cows game
  - Like the Mastermind board game
  - Make a guess and get a hint
- Goal is to minimize the number of guesses
  - By eliminating hypotheses after each hint
- Scoured the Web and YouTube
  - Python presentations by Adam Forsyth
  - Donald Knuth paper on Mastermind

Challenge 4 Output


```
{
  "rounds": [
    "1: 012 -> (0, 2)",
    "2: 103 -> (0, 1)",
    "3: 240 -> (0, 2)",
    "4: 421 -> (3, 0)"
  ]
}
```

Challenge 4 spin.toml Excerpt

```
[component.bullseye]
allowed_outbound_hosts = ["https://bulls-n-cows.fermyon.app"]
source = "bullseye/target/wasm32-wasi/release/bullseye.wasm"

[component.bullseye.build]
command = "cargo build --target wasm32-wasi --release"
watch = ["src/**/*.rs", "Cargo.toml"]
workdir = "bullseye"

[[trigger.http]]
component = "bullseye"
route = "/..."
```

Challenge 4 Loop

```
  let mut permutations = make_permutations();
[...]
  while let Some(guess) = permutations.pop() {
    let bulls_cows_output: BullsCowsOutput =
      send_guess(&game_id_option, &guess).await?;
[...]
    if solved {
      break;
    }
[...]
    permutations
      .retain(|permutation| output_hint == make_hint(&guess, permutation));
[...]
  }
```

Challenge 4 More Code

- #[serde(alias = "gameId")]
- struct Permutation
- fn has_all_unique_symbols(&self) -> bool
- impl IntoBody for BullseyeOutput
- fn make_hint(guess, secret) -> Hint
- unmatched_secret_symbols.swap_remove(index)
- fn make_permutations() -> Vec<Permutation>

Future

- CroftSoft Spin Prototype
  - Cleaned-up Spin example components
- Authentication (AuthN) / Authorization (AuthZ)
  - OAuth 2.0 / OpenID Connect (OIDC)
- Fullstack Jamstack Serverless Rust
  - Rust-Wasm on frontend using Dioxus
  - Rust-Wasm on backend using Spin
- Wasm archive repositories
  - Like downloading Java Archive (JAR) files

Links

- Fermyon Advent of Spin
  - https://github.com/fermyon/advent-of-spin/tree/main
- CroftSoft Advent of Spin 2023 Solutions
  - https://github.com/david-wallace-croft/advent-of-spin
- CroftSoft Spin Prototype
  - https://github.com/david-wallace-croft/spin-prototype
- Adam Forsyth, "Beating Mastermind: Winning Games, Translating Math to Code, and Learning from Donald Knuth", PyGotham 2018 https://youtu.be/2iCpnWYXPik?si=wxlGsvkOEfVHEfsF

Licenses

- Slides and code are © 2024 CroftSoft Inc
- This slide presentation is available under the terms of the Creative Commons Attribution 4.0 International License https://creativecommons.org/licenses/by/4.0/
- The code is available under the terms of the open source MIT License https://opensource.org/license/mit/