LFX Mentorship 2023 Fall Term

Compile Move2Kube to WASM/WASI and run it in the browser

Mentors: Harikrishnan Balagopal, Mehant Kammakomati

Name and Contact Information

Name: Prakhar Agarwal

Github: Prakhar-Agarwal-byte

Email: prakharagarwal3031@gmail.com

Alternative Mail: prakhar.agarwal.mec20@itbhu.ac.in

University Name: Indian Institute of Technology (BHU) Varanasi

Current year: 3rd year

Website: prakharcodes.tech

LinkedIn: Prakhar Agarwal | Linkedin

Country: India

Time Zone: IST (UTC+05:30)

Table of Contents

- Name and Contact Information
- Table of Contents
- Overview
    - Abstract
    - Goals
- Approach
    - Implementation and Rationale
- Project Timeline
- Extended Goals
- About Me
    - My learnings and experience in working with WebAssembly.
    - My learnings and experience with Move2Kube tool.
    - My contributions to Konveyor
    - Google Summer of Code 2021 and 2023
    - Summer of Bitcoin 2023
    - Intern at Cisco
    - Have I been a LFX mentee before?
- Motivation
    - Why do I wish to participate in LFX?
    - Why do I wish to work with Konveyor?
    - Why this particular project?
    - Operating systems I work with?
    - Availability
- Why me?
- Research Material and references

Overview

Abstract

Move2Kube is a command-line tool for automating creation of Infrastructure as code (IaC) artifacts. It has inbuilt support for creating IaC artifacts for replatforming to Kubernetes/OpenShift. We want to compile targetting WASM/WASI and run the resulting WASM module in the browser. This will help up showcase Move2Kube for demos and allow users to quickly try out Move2Kube without having to install it or any of its dependencies.

Goals

- Run Move2Kube CLI in the browser using WebAssembly.

Approach

Implementation and Rationale

I already started working on this project so that I can articulate my approach better. I built the Move2Kube source code using Tinygo CLI using the command `tinygo build -o WASM.WASM -target WASM ./main.go`

I tracked down all the errors that I faced and created a list for future reference:-

1. **imports golang.org/x/sys/unix: build constraints exclude all Go files in /home/prakhar/go/pkg/mod/golang.org/x/sys@v0.6.0/unix**

This error is because unix calls are not supported by Tinygo in WASM. This dependency is indirectly imported in the `"github.com/docker/docker/pkg/archive"` package used in utils.go in container package. I looked around for alternative libraries that can handle tar archives, one such was `"archive/tar"` from the standard library, unfortunately this is also not importable in Tinygo WASM. Here is the complete list of packages supported by Tinygo. So I had no option but to remove this file from the build. I used **conditional build tag** feature in Golang for it.

```
//go:build os
// +build os
```

These comments instruct the compiler to only include these files when “os” build tag is included. The error was resolved after this.

2. **# github.com/Masterminds/sprig ../go/pkg/mod/github.com/!masterminds/sprig@v2.22.0+incompatible/network.go:9:18: undefined: net.LookupHost**

This code is imported in utils.go in common package and used as such

```
packageTemplate, err :=
template.New("").Funcs(sprig.TxtFuncMap()).Parse(tpl)
```

It is related to the **networking calls** used in the sprig package. Since network calls are not available in WASM for browsers we will need to

stub it. Sprig package is used to get some text template utility functions out of the box, this package can be easily replaced with custom template functions or using the standard `template/text` package.

3. **github.com/docker/docker/pkg/ioutils ../go/pkg/mod/github.com/docker/docker@v20.10.12+incompatible/pkg/ioutils/fswriters.go:75:15: undefined: os.Chmod**

This is related to the system calls which deals with setting OS level permissions on files. This needs to be removed because we dont have access to it in WASM. I will make use of conditional build tags to ignore this in WASM.

4. **# github.com/go-git/gcfg ../go/pkg/mod/github.com/go-git/gcfg@v1.5.0/set.go:41:20: v.Type().FieldByNameFunc undefined (type reflect.Type has no field or method FieldByNameFunc)**

go-git package currently is not WASM compatible hence its showing this error. I will remove this dependency to make the project first run on WASM and later after discussion find an alternative to this library or raise issue in the repo to make this WASM compatible.

All these errors are mainly related to **network and system calls** which are inherently not supported in WASM. First priority is to make Move2Kube run on WASM, even if we have to drop some features for it. Once it is successful, I will start to work on supporting more Move2Kube features on WASM.

Proper filesystem in WASM

Move2Kube needs a proper filesystem in order to function, since we won't have an actual OS, we will need to emulate a filesystem in browser. One way to accomplish this is to use the FileSystem API. This interface is available in most modern browsers and can be accessed via javascript. We can create entire directory structures in the browser, just like the example below.

```
// Taking care of the browser-specific prefixes.
window.requestFileSystem =
  window.requestFileSystem || window.webkitRequestFileSystem;

// ...

// Opening a file system with temporary storage
window.requestFileSystem(
  TEMPORARY,
  1024 * 1024 /*1MB*/,
  (fs) => {
    fs.root.getFile(
      "log.txt",
      {},
      (fileEntry) => {
        fileEntry.remove(() => {
          console.log("File removed.");
        }, onError);
      },
      onError,
    );
  },
  onError,
);
```

Now since this function is available in JS, I will make bindings in Go Code so that I can call it from there. Here's an example of manipulating DOM using WASM

```
<!-- index.html -->
<html>
<head>
    <script src="WASM_exec.js"></script>
    <script>
        const go = new Go();
        WebAssembly.instantiateStreaming(fetch("main.WASM"),
go.importObject).then((result) => {
            go.run(result.instance);
        });
    </script>
</head>
<body>
    <div id="output"></div>
</body>
</html>
```

```
// main.go
package main

import (
    "syscall/js"
)

func main() {
    c := make(chan struct{}, 0)

    js.Global().Get("document").Call("getElementById",
"output").Set("innerHTML", "Hello from Go!")

    <-c
}
```

We can also make use of the WASI FS proposal to write the filesystem but since it is used to access the host's filesystem, we dont need this much privilege and our goal can be accomplished with the Filesystem API only. Later if its implemented in wasmer, wasmedge then we can try and experiment with it but since it's in planning phase at the moment, we are better off using the Filesystem API.

I will create a **separate package for WASI Filesystem** that will be only compiled when the build target matches with WASM/WASI i.e. **conditional build** or else the original package will be used. This way we can create alternative codebase needed for running Move2Kube in WASM without modifying much of the original code.

Here's an example of where I have modified the the replicateDeletionCallback in filesystem package using the Filesystem API from Go for WASM:

```
package main

import (
    "fmt"
    "syscall/js"
)

func replicateDeletionCallBack(source, destination string,
config js.Value) error {
    si := js.Global().Get("fs").Call("statSync", source)

    // Check for errors
    if si.IsNull() {
        errMsg := fmt.Sprintf("Unable to stat %s", source)
        return js.Error{Value: js.ValueOf(errMsg)}
    }
```

```
    js.Global().Get("fs").Call("rmdirSync", destination,
js.ValueOf(map[string]interface{}{"recursive": true}))

    err := js.Global().Get("fs").Call("mkdirSync",
destination, si.Get("mode"))
    if !err.IsNull() {
        errMsg := fmt.Sprintf("Unable to create directory
%s", destination)
        return js.Error{Value: js.ValueOf(errMsg)}
    }

    js.Global().Get("fs").Call("chmodSync", destination,
si.Get("mode"))

    return nil
}
```

Call it from the main function

```
func main() {
    js.Global().Set("replicateDeletionCallBack",
js.FuncOf(replicateDeletionCallBack))

    c := make(chan struct{}, 0)
    <-c
}
```

Create an HTML file and JavaScript glue code

```
<!-- index.html -->
<html>
<head>
    <script src="WASM_exec.js"></script>
    <script>
```

```
        const go = new Go();
        WebAssembly.instantiateStreaming(fetch("main.WASM"),
go.importObject).then((result) => {
            go.run(result.instance);
        });

        function replicateDeletion(source, destination,
config) {
            return replicateDeletionCallBack(source,
destination, config);
        }
    </script>
</head>
<body>
    <div id="output"></div>
</body>
</html>
```

Support for Network calls in WASM

When WebAssembly is executed in a web browser, it doesn't have direct access to network calls or other system-level operations. Instead, it can communicate with the host environment (the browser) through JavaScript functions and Web APIs.

To perform network calls in a WebAssembly module within a browser context, I will expose JavaScript functions that handle network operations. The WebAssembly code will call these JavaScript functions, and the JavaScript functions will then use browser-provided APIs (like fetch for HTTP requests) to perform the actual network communication.

Since TinyGo **doesn't support the net/http** package for WebAssembly, you'll need to use JavaScript to make the HTTP

request. Here's an example of how I modified the [download content from URL function](#).

```
// Replace http.Get with JavaScript fetch API
js.Global().Get("fetch").Invoke(downloadOptions.ContentURL).C
all("then", js.FuncOf(func(this js.Value, args []js.Value)
interface{} {
    resp := args[0]
    return resp.Call("arrayBuffer")
})).Call("then", js.FuncOf(func(this js.Value, args
[]js.Value) interface{} {
    arrayBuffer := args[0]
    js.CopyBytesToGo(arrayBuffer, contentBytes) // Read the
response into a Go byte slice
    // ... rest of the code ...
    return nil
}))
```

Project Timeline

Project Period(September 4, 2023 - November 30, 2023)	
Sep 4, 2023 - Sep 15, 2023	• Setup development environment, familiarise myself with the codebase. • Introduction with mentors, decide upon a weekly meeting time
Sep 16, 2023 - Sep 30, 2023	• Identify all the dependencies that don't work with WASM • Finalize the approach for

	using Filesystem after discussion with mentors
Oct 1, 2023 - Oct 15, 2023	• Iteratively work converting the codebase to be WASM compatible
Oct 16, 2023 - Oct 30, 2023	• Demo Move2Kube running in the browser • Plan out more features to implement in WASM
Oct 31, 2023 - Nov 14, 2023	• Keep working on the features as planned • Take feedback from community • Discuss any stretch goals
Nov 15, 2023 - Nov 22, 2023	• Write tests to catch any regressions • Document the process for reference in future development • Work on stretch goals
Nov 23, 2023 - Nov 30, 2023	• The final week has been left free for completing any remaining work (if any). This provides sufficient cushion for making sure that the timeline is followed. If everything gets completed smoothly before this period then this will be utilized for incorporating community feedback.

Extended Goals

- Improve support for WASM comptaibility as more features get introduced in the future
- Explore more products by Konveryor like the analyzer, tackle
- Keep on contributing and engaging with the Konveyor community

About Me

My contributions to Konveyor

I am actively contributing to Konveyor Move2Kube from the past 2 weeks. My contributions include writing tests for the filesystem package, fixing documentation and refactoring code.

PR	Description	Status
#1066	Added tests for replicateProcessFileCallBack	Merged
#1065	Update the documentation link in Readme file	Merged
#1071	Added test for replicateAdditionCallback	Under Review
#1072	Refactor "test_data" to "testdata" for consistency	Under Review

I also take part in discussions in the slack channel, attend the bi-weekly community meetings and am proactive for any suggestion or feedback.

My learnings and experience in working with WebAssembly

I have written Go programs and compiled them to WASM using the official Go compiler and the Tinygo version as well. I understood that there are differences between both these compilers as some **standard library packages are not supported** by Tinygo. The output binary produced by Tinygo is very **small in size**, because it was originally designed to work in microcontroller chips. WebAssembly allows us to run code at near-native speeds in web browsers. It's designed to be highly efficient, making it suitable for performance-intensive tasks. I also experimented with Move2Kube codebase and tried to compile it to WASM using Tinygo, and listed out some interesting findings, that will help in this project. WebAssembly code is executed in a sandboxed environment within the browser, ensuring that it doesn't have direct access to the underlying system. So we will need to remove stub/remove that code which requires this access. It is supported by major web browsers, including Chrome, Firefox, Safari, and Edge. This cross-browser compatibility ensures that applications built using WebAssembly can reach a wide audience without compatibility issues.

My learnings and experience in working with Move2Kube tool

I built the Move2Kube CLI from source and tried to compile various apps written in several languages using it. I got interesting analysis reports and auto generated YAML files which I could then deploy to my cluster. I started exploring and contributing to the codebase,

which further helped me understand the internal working of the tool. Meanwhile, I also attended the bi-weekly community meetings and took part in slack discussions, this helped me engage with the community and learn more about the various aspects of the tool. Some of them are:

Discovery and Analysis: Move2Kube can automatically analyze existing applications, regardless of their complexity or architecture, to determine their dependencies, components, and runtime characteristics. This analysis is crucial for understanding how to effectively migrate applications.

Containerization: One of Move2Kube's core functions is converting monolithic applications into containerized microservices. It can break down applications into smaller, manageable components, making it easier to maintain, scale, and update.

Technology Agnostic: Move2Kube is designed to work with a variety of programming languages, frameworks, and application types. It's not tied to any specific technology stack, enabling it to modernize applications regardless of their origin.

Reduced Time and Effort: Move2Kube accelerates the migration process by automating many tasks that would otherwise be time-consuming. This reduces the effort required for migration and lowers the risk of errors.

Migration Path Planning: The tool provides insights into the steps and resources required for successful migration, aiding in planning and resource allocation.

Google Summer of Code 2021 and 2023

I have been working on open source for almost **three years**. I did my Google Summer of Code (GSoC) under Purr Data in 2021 and again in 2023. In this project, I redesigned the **UI/UX** using the best design principles, making it more intuitive to use. I kept working and being in touch with my org and this year too, I was selected where I developed an **Autosave feature** that regularly saves the project files at a configurable interval, this helps to prevent data loss in caseof sudden app crashes.

Summer of Bitcoin 2023

Later I went on to exploring the Bitcoin technology, and was selected in the Summer of Bitcoin 2023 under Padawan Wallet. Here I worked on developing the **Padawan Bitcoin Wallet**. I developed features such as OP_RETURN, implemented BIP32 address paths, BIP84 template for generating wallet addresses, BIP21 for generating easy to use payment links.

Intern at Cisco

Through my oncampus internships, I was hired by Cisco as a summer intern, where I got my first taste working with **cloud native technologies**. There I worked extensively on Kafka and created **Kafka Recap tool** in **Golang** to record and replay Kafka messages amongst several microservices. I also used **Docker and Kubernetes** for building and deploying services, there I got to learn about **CRDs and YAML configs**. This tool helped in system analysis, bug triaging and greatly improved the development velocity.

Have I been a LFX mentee before?

No, this is my first time applying for LFX hence I am eligible for the mentorship.

Motivation

Why do I wish to participate in LFX?

I have been contributing to the open source software since 2020 when I was in the **first year** of my college. Since then I have been looking out for opportunities from where I can get to learn and expand my skills. I got selected for **GSoC and SOB**, where I got to learn a diverse skillset from my highly experienced mentors. Now I wish to dive deep into **cloud native ecosystem** of technologies and I believe that **LFX** provides me with this exciting opportunity of learning it directly from the very **best mentors** out there.

Why do I wish to work with Konveyor?

Konveyor is involved in creating tools and technologies that facilitate the **migration and modernization** of applications to cloud-native environments. By working with Konveyor, I will have the opportunity to contribute real-world challenges related to application migration, containerization, and cloud-native architectures. This hands-on experience will greatly **enhance my technical skills** and make me more proficient in these technologies.

Why this particular project?

Working on a complex project involving both Move2Kube and WebAssembly/WASI can provide a significant learning opportunity. It will also demonstrate my **proficiency in multiple areas**, including software development, WebAssembly technology, and cloud computing. This kind of accomplishment will **enhance my portfolio** and open up **new opportunities** in the future.

Operating systems I work with?

I primarily work with **Ubuntu** in a virtual machine for development purposes. At the same time, I also have Windows 11 as the base operating system, which lets me test programs on both OS.

Availability

I have no other commitment other than my university. I will be solely dedicating the time to this project, so I will be able to devote a minimum of **35-40 hours per week**. I am also available for weekly meetups with my mentors, which we can schedule as suitable.

Why me?

I've invested time in understanding the principles and capabilities of WebAssembly, and Move2Kube. My track record of creative problem-solving and the ability to approach challenges from multiple angles will be invaluable in ensuring a smooth transition of Move2Kube to the browser environment. I thrive on finding **innovative solutions to technical hurdles**. This unique capability sets me apart from others. I am excited about the opportunity to

work on this project and will work as hard as I have to make this project a **grand success**.

Research Material and References

- Github issue
- Guide to webassembly in Tinygo
- Slack discussions
- Mozilla filesystem docs
- WASI FS Proposal
- List of packages supported by Tinygo in WASM
- Example of go app in WASM
- Move2Kube Codebase and Docs