Report DataCloud project

Íñigo Aréjula Aísa

October 23, 2023

1 Background

My expertise in undertaking this project stems from my extensive familiarity with cloud-native applications. I possess a deep understanding of their intricacies, encompassing deployment strategies and effective management techniques. Additionally, I am proficient in utilizing various workflow tools such as Apache Airflow. Furthermore, I have garnered significant experience in working with containers, employing tools such as Docker, Docker Compose, Podman, and Kubernetes.

2 Part 1

In the initial phase of the project, I was tasked with programming my own workflow orchestrator. After careful consideration, I opted for Golang due to its robust built-in mechanism for Remote Procedure Calls (RPC). I have designed each workflow step as an individual RPC service, which the orchestrator will invoke seamlessly. These steps will be encapsulated within containers utilizing Docker technology. Notably, I have ensured the inclusion of a *Volume*, enabling all containers to access a shared folder. This shared space serves as a repository for storing the products of each step, in our case, the generated files.

The orchestrator invokes each procedure as needed, leveraging the synchronous nature of RPC, which facilitates the creation of a sequential Directed Acyclic Graph (DAG). While constructing a straightforward DAG is relatively simple, handling complex workflows introduces intricacies. Below is an example of the code required to call a service in this context

```
    func arango(filename string) error {
// Conect to  RPC server  at port 1234
client, err := jsonrpc.Dial("tcp", "converter:1234")
if err != nil {
fmt.Println("Error while conecting to RPC server:", err)
return err
}
defer client.Close()
```

```
// Call Split
response := datacloud.Response{}
request := datacloud.Request{Filename: filename}
err = client.Call("ArangoService.Arango", &request, &response)
if err != nil {
fmt.Println("Error calling remote method:", err)
return err
}
return nil

}
```

I invested a total of 2 hours and 13 minutes in this project. The majority of the time was allocated to configuring the RPC protoco. the process of creating the Docker Compose setup was remarkably swift and efficient. Additionally, I dedicated time to crafting the necessary functions and corresponding Dockerfiles for each of the workflow steps. I did not need time for learning or read documentation in this step.

3 Part 2

In this phase of the project, I employed Argo Workflow. Despite my prior lack of experience with Argo, its seamless integration with Kubernetes made the configuration process surprisingly intuitive. I dedicated 40 minutes to reading documentation. After that I watched two instructional videos on YouTube to grasp the core features and comprehend how to establish dependencies between different workflow steps; this learning phase took me 37 minutes. Following this, I constructed the entire workflow, investing a total of one hour and ten minutes.

The fundamental concept behind my workflow design involved creating a DAG, outlining the dependencies between individual steps and specifying their respective inputs and outputs.

Following the DAG definition, my next task was to articulate detailed templates for each step of the workflow. These templates encapsulated the specific instructions, parameters, and configurations necessary for the successful execution of each step. By providing explicit guidance on inputs, outputs, and any required resources, these templates facilitated a consistent and reliable workflow execution. This is an example of the unzip step defined in the cargo file.

```
    - name: unzip-step
  inputs:
    parameters:
      - name: file_path
  container:
    image: unizip_datacloud:latest
```

```
command: [main]
source: |
    echo "{{inputs.parameters.file_path}}"
```

It is also needed to say that running this requires a Kubernetes cluster, which I do not cover in the task, however, I added a brief script for deploying Argo in the cluster.

4 Part 3

This phase proved to be unexpectedly straightforward. Although I initially faced challenges due to the sparse documentation, experimenting with the portal proved to be an effective learning method. Understanding the tool's functionality was rapid once I delved into it.

Initially, I encountered a hurdle when I discovered that the portal didn't function optimally in Firefox, I spent around 10 minutes before switching to Chrome. Upon transitioning to Chrome, I spent an additional 7 minutes thoroughly exploring the portal's features. With this knowledge in hand, creating the actual workflow was remarkably swift, taking only 2 minutes to create. The resulting workflow is shown in Figure 1.

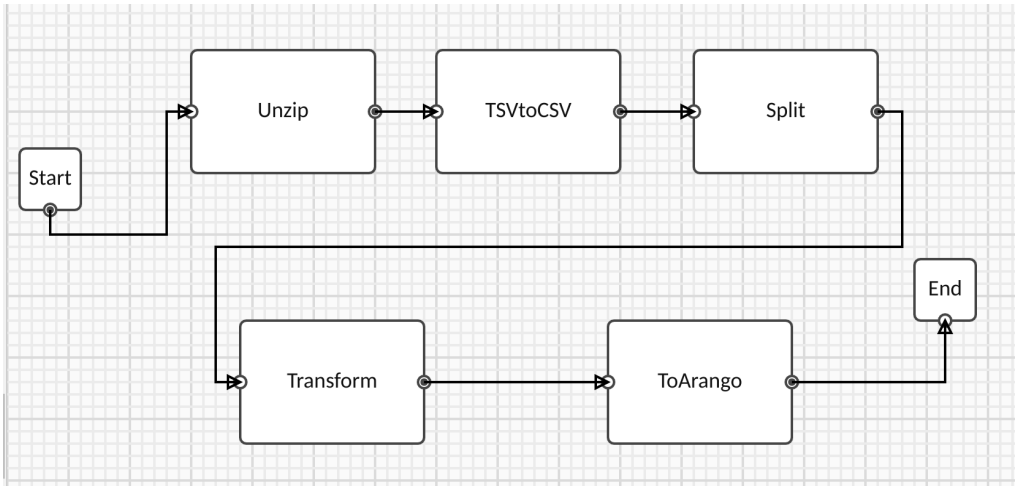

Figure 1: Workflow