Workflow as a Service

A Platform-as-a-Service for Workflow Creation and Orchestration

Chelsea Fernandes (ccfernandes@scu.edu), Gowtham Rajeshshekaran (grajeshshekaran@scu.edu),
Karthik Cheernalli Manjunath (kcheernallimanjunath@scu.edu), Naveen Kumar (nnaveenkumar@scu.edu)

Department of Computer Science and Engineering, Santa Clara University

***Abstract*—WaaS provides a platform-as-a-service (PaaS) for users to create a workflow consisting of tasks (as microservices) in a specified order, which simplifies the process of implementing, deploying, executing and visualizing a user's workflow. The final product has a user interface (UI) to specify the order of execution of the tasks in a workflow, which are then deployed inside a Kubernetes cluster. A scheduler microservice is deployed to orchestrate the workflow and its tasks, and a user interface to visualize the running workflows in real-time.**

I. Introduction

Workflows are used by organizations to define a set of functions within a business domain thereby automating the execution of tasks involved from the start until completion. Workflows can become very complex and lengthy, making it difficult to manually deploy the associated services and keep track of underlying complexities. Our solution is to provide a platform-as-a-service that takes care of the orchestration of tasks and integration of the different workflow services. The user just needs to specify the order of tasks using the user interface and provide the docker images for those tasks and the rest will be handled by this platform. The key part of this product is the scheduler service, which acts as the main coordinator for all the other services (see Section VI for a more detailed explanation). In addition to this, users can also visualize their workflows using the user interface. Existing solutions are for the most part on par with our Workflow-as-a-Service in terms of features, except for the underlying mode of communication, which is discussed further in Section IV.

II. Background

A. Workflows

A Workflow is defined as a sequence of tasks that processes data through a specific path, from start to completion, as seen in Figure 1. While they can be purely sequential, workflows can also branch from the main sequence based on specific conditions (ex. if/then branches). Workflows are integral to business as they define the logic that drives the product. The applications for workflows are endless - Figure 2 shows three example applications of workflows along with specific tasks that could be created - invoice generation, order processing, and notifying a user in-app.

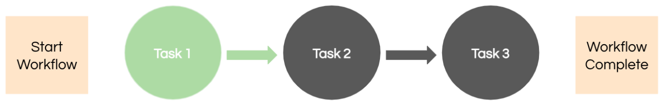


Figure 1. General Workflow Structure

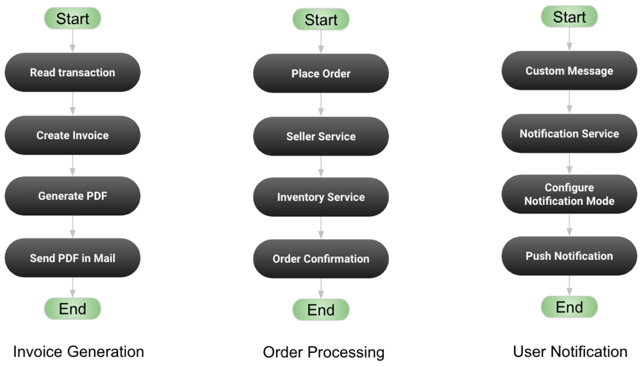


Figure 2. Example Applications of Workflows

III. Technologies Used

A. Cloud Technologies

Following cloud technologies are used in the application:

- Docker Containers - Containerizes each of the services in the application.
- Kubernetes - Provides clusters and pods for deployment.
- Minikube - Allows local deployment of an application using Kubernetes engine.

B. Fullstack

Two different web frameworks are used to create the example workflow which highlights the fact that a given workflow is language agnostic and can be implemented with any language or framework as long as it is containerized. We make use of simple HTTP and REST calls for inter-service communication.

- Web Frameworks
  - Spring Boot - Java framework for creating fullstack applications.
  - Flask - Lightweight Python framework for creating fullstack applications.
- Inter-service Communication
  - HTTP / REST APIs

C. Frontend

- Web Frameworks
  - NodeJS - Javascript framework for creating front-end applications.
- Languages
  - HTML
  - CSS
  - Javascript
- Libraries
  - jQuery
  - Bootstrap

  The above mentioned libraries are used to aid the development of the workflow specification and overall design of the interface.

D. Backend

- Language: Java
- Frameworks: Spring Boot (Java)
- Databases
  - DynamoDB - NoSQL database service that supports key–value and document data stores, from Amazon Web Services (AWS).
  - MongoDB - Document-oriented NoSQL database that uses JSON-like documents with optional schemas.

  NoSQL databases provide the flexibility to store non-relational data that are processed and stored during workflow execution. DynamoDB was used in addition to MongoDB to demonstrate that a cloud database could easily be integrated into the system.

IV. Related Work

There are many similar Workflow-as-a-Service applications already available in the market. Amazon Web Services offers a PaaS called *Simple Workflow Service* which allows users to have full control over implementation and coordination of tasks without worrying much about the underlying complexities involved to track the progress and maintain their state [2]. Google Cloud offers *Workflows*, which provides features similar to AWS's service. Both of these services follow a publisher/subscriber *(pub-sub)* model, which is different from the implementation presented in this report, which makes use of simple HTTP calls for synchronous communication between services. Added to that, they also provide facilities to run these tasks as serverless functions, which is considered a future work for this project.

V. System Architecture

To understand WaaS's System Architecture, two key components and their differences must be comprehended well - Workflow Specification and Workflow Instance.

Specification vs Instance: A specification defines a workflow's attributes like the list of tasks and their order of execution while an instance is a running instance of that specification. A workflow specification provides the blueprint for the workflow against which multiple workflow instances can be created and executed. For example, in Figure 3, we can see a workflow specification that defines a list of four tasks starting with *Process Order*, followed by *Send Order to Vendor*, *Order Delivery* and *Notification*. Below the specification, two running instances of this workflow specification are given. One instance handles an order for an *IPhone 13*, while the other instance handles an order for a *Galaxy S20* phone. Both instances belong to different customers, with different notification and delivery preferences. There can be multiple instances of a workflow following the same path and running the same code, but with different values. It is similar to the object-oriented paradigm, where specification is analogous to a class, while instance is analogous to objects of that class. Similar explanation applies to Task Specification and Task Instance. For example, a *Notification* task specification would define how the notification needs to be sent to the user-provided a set of data that is provided by a real-time running instance of this task.

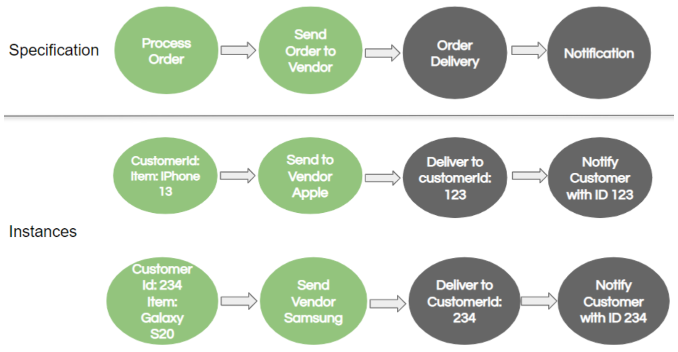

Specification
Process Order
Send Order to Vendor
Order Delivery
Notification
Instances
CustomerId: Item: IPhone 13
Send to Vendor Apple
Deliver to customerId: 123
Notify Customer with ID 123
Customer Id: 234 Item: Galaxy S20
Send Vendor Samsung
Deliver to CustomerId: 234
Notify Customer with ID 234

Figure 3. Workflow specification vs Instance

As a result of this distinction between specification and instance, the architecture is divided into two parts: workflow specification creation and workflow instance creation.

A. Workflow Specification Creation

Workflow specification architecture is presented in Figure 4. The workflow specification service provides users (most likely developers or designers who will create workflows) a user interface to create task and workflow specifications. Using this interface, the user can create a Task Specification by providing its name and Docker image that will be used to execute tasks of this type. After creating multiple tasks, the user can create a workflow specification by configuring the order in which these tasks need to be executed, along with a name for this type of workflow. The workflow specification service will persist these specifications in the database to be used later. It also provides APIs to retrieve these specifications. From the user interface, the user can deploy this workflow specification, which deploys all the Docker images for the tasks inside this workflow specification into a Kubernetes cluster. The tasks can now be executed through these *(Task)* services created inside the Kubernetes cluster. The workflow specification service also adds two *core services* of its own into this cluster: *Scheduler Service* and *UI Service*, which will be used to manage the running instances of this workflow specification.

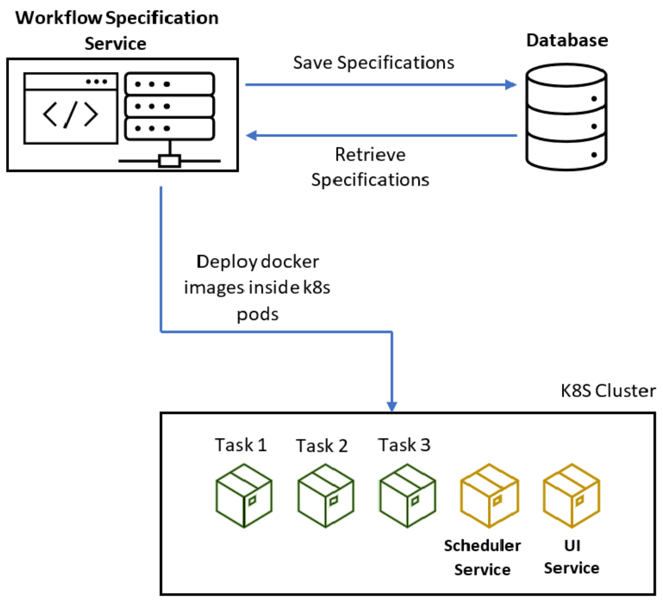


Figure 4. Workflow specification Architecture

B. *Workflow Instance Creation*

Figure 5 shows the Workflow Instance Architecture. The Scheduler service provides a REST API to create new instances from a workflow specification. To create instances, the details of the workflow specification has to be provided, which is queried from the workflow specification service's data store. In real-world scenarios, the creation of instances is most likely to be triggered via events. For example, when the user orders food from a delivery application, the web/mobile application can trigger this workflow on behalf of the user. But the instances can also be created from the Workflow UI Service that comes bundled with other task services. This can be useful especially if the developers want to test their workflows directly from the user interface. The Scheduler Service will create workflow instances and then manage the complete life-cycle of those instances and also the life-cycle of the task instances that belong to these workflow instances. It will begin execution of each of these tasks one by one by sending execution requests to task services via a webclient called RestTemplate provided by the Spring framework. Once a task complete its execution, it will notify the scheduler service about the same. Scheduler service will now mark the task as complete and will trigger the next task as per the ordering provided from the workflow specification. The entire workflow execution is managed this way and once the workflow is completed fully, its status will be updated as complete. The workflow UI service provides a visualization of the workflow instance to its users, where the users can see the tasks that are completed and the ones that are still pending.

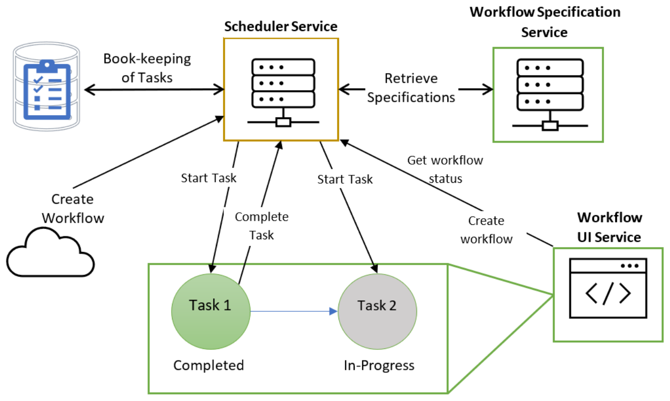


Figure 5. Workflow Instance Architecture

VI. Software Architecture

Our system follows a service-based architecture, where the components involved in the system are developed as standalone microservices. This enables each of the services to be designed, developed, deployed and scaled independently based on their workload. The services are language agnostic where different technology stacks are used and are coupled with separate data stores to maintain the state of the service. The services are packaged into Docker images and are deployed as separate pod instances into a Kubernetes cluster. Kubernetes helps us in achieving automating deployment, scaling, and management of the containerized services. It also provides us with a simpler service configuration which is used for communication and service discovery (via kube-dns [10]). Each of these services and their core functionalities are explained in detail in the following sections.

A. *Workflow Specification Service*

Workflow specification service holds the web, application and data layer for the functionalities related to tasks and workflow specification. The tasks created by the users and the workflow specification along with the task ordering details are processed and stored in the MongodB data store attached to this service. Figure 6 shows the architecture of this service. This service uses spring MVC [4] to expose REST API endpoints to create, retrieve and deploy these specifications. It makes use of spring-data MongoRepository interface to persist and query the database. Listing 1 shows a task specification, which defines the field dockerImage for executing this task, amongst other fields. Listing 2 shows a workflow specification, which defines the order of task specifications for the execution of this workflow specification. The specification service uses fabric8 kubernetes java client [3] to deploy these Docker images inside a kubernetes cluster (minikube in this case). The MongoDB database is a StatefulSet kubernetes resource [5] to keep the data persistent even if a pod restarts.

Listing 1. Task Specification

```
{
    "taskName": "Email Notification",
    "serviceName": "email-notification",
    "dockerImage": "user/email-
       notification:v1",
    "cpuLimit": "0.5",
    "memoryLimit": "256Mi"
}
```

Listing 2. Workflow Specification

```
{
    "name": "Registration",
    "taskOrderList": [
        {
            "taskId": "62789477e11dea2af1
               e2f7fb",
            "order": "1"
        },
        {
            "taskId": "62789498e11dea2af1
               e2f7fc",
            "order": "2"
        }
    ]
}
```

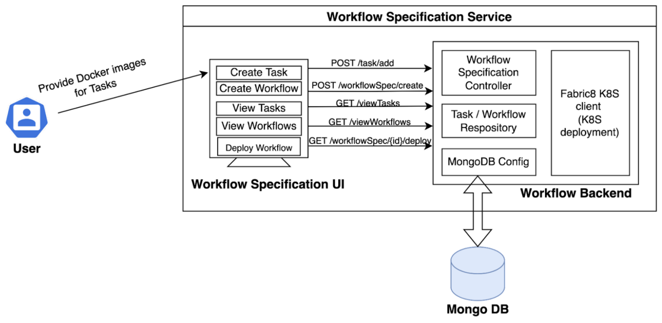

Workflow Specification Service
Provide Docker images for Tasks
User
Create Task
Create Workflow
View Tasks
View Workflows
Deploy Workflow
POST /task/add
POST /workflowSpec/create
GET /viewTasks
GET /viewWorkflows
GET /workflowSpec/{id}/deploy
Workflow Specification Controller
Task / Workflow Respository
MongoDB Config
Fabric8 K8S client (K8S deployment)
Workflow Specification UI
Workflow Backend
Mongo DB

Figure 6. Workflow Specification Service

The workflow specification service provides the following REST APIs:

- POST /workflowSpec/create - creates a workflow specification
- GET /workflowSpec/:workflowSpecId - Gets a specification from its ID
- GET /workflowSpec/getAllWorkflows - Gets all workflow specifications, including Task Specification details.
- GET /workflowSpec/:workflowSpecId/deploy - Deploys a workflow specification, given its ID
- POST /task/add - Adds a new task specification
- GET /task/:taskSpecId - Gets a task specification from ID
- GET /task/:taskSpecId/deploy - Deploys an individual task specification, given its ID
- GET /task/:taskSpecId/deploymentStatus - Gets the deployment status of a task specification

B. Scheduler Service

The scheduler service is the core microservice which communicates with all other services in the system. It handles the scheduling of each of the tasks and drives workflow execution until completion. Figure 7 shows the architecture of the scheduler service. All the inter-service communications are handled through the RestTemplate web client provided by the Spring framework. Under the hood this client uses HTTP protocols for communication. The request and response between the UI interface and this service are handled through simple HTTP REST APIs. The exposed REST endpoints are used by the client to create a workflow instance, view the instances created so far and their statuses. The data is persisted into the DynamoDB data store provided by AWS. We make use of the DynamoDB Mapper SDK (software development toolkit) as the ORM (object–relational mapping) to query the database. Listing 3 shows a sample workflow instance using JSON, which shows workflow name and its status. Each task also has its own status and the status can be Pending, In Progress, Completed or Failed. The tasks also have the property *order* to specify their place in the execution order of all tasks of this workflow. The workflow instance also stores a set of key-value pairs, called attributes which can be used to pass data to tasks. Each task can add or update these attributes, which can then be available to any other tasks that are executed later in the workflow.

Listing 3. Workflow Instance

```
{
    "workflowSpecId": "628530954adcf0577
       ee4872c",
    "name": "User Reg 2",
    "workflowStatus": "IN_PROGRESS",
    "taskInstanceList": [
        {
            "taskId": "6284454184817f3566
               0b740d",
            "serviceName": "registration-
               form",
            "order": 1,
            "taskName": "Registration
               Form",
            "status": "COMPLETED",
            "url": "http://{{HOST}}:30009
               /registration/5d7b6837-37b
               9-4053-b138-9c59585e9584/6
               284454184817f35660b740d",
        },
        {
            "taskId": "62842a64fadc2c42
               feb4a6db",
            "serviceName": "email-
               notification",
            "order": 2,
            "taskName": "Email
                Notification",
            "status": "FAILED",
        }
    ],
    "attributes": {
        "email": "abc@gmail.com"
    },
    "createdAt": "2022-05-18T18:45:42.495
        Z",
    "updatedAt": "2022-05-18T18:46:06.003
        Z"
}
```

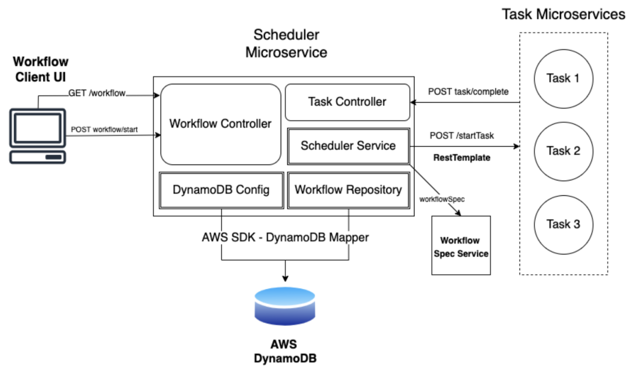


Figure 7. Scheduler Service

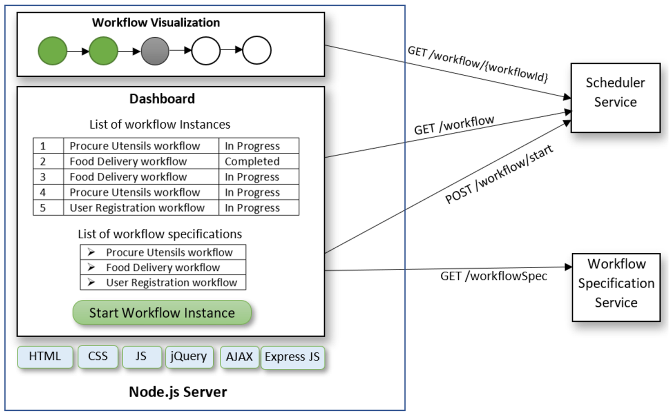
The scheduler service provides the following REST APIs:

- POST /task/complete - Completes a task and updates any attributes returned by the task.
- POST /task/retry - Retries a failed task
- POST /workflow/start - used to start the particular workflow, provided a workflow specification ID and an optional set of workflow-instance-specific key-value attributes
- GET /workflow/:workflowId - Gets the status of a workflow instance from the workflow instance ID
- POST /workflow/:workflowId/update - Updates attributes of a workflow instance

C. Workflow UI Service

Figure 8 shows the Workflow UI Service Architecture. The interface is built using HTML, CSS and javascript. In addition to these, the service uses JQuery library [7] for handling events and AJAX requests. It also uses Bootstrap CSS [8] for a few UI components. These resources are hosted on a NodeJS server, which is a javascript based web application framework. Using the workflow-ui service, users can manage their Workflows and Tasks. In the Dashboard screen, the user can see the list of all Workflow Instances created, their type, their last updated time, and their current status. Dashboard also provides the option to create and start a new Workflow Instance. Once the workflow instance is started, its progress can be tracked from the workflow UI service. Let's say if any task has failed, it can be retried directly from the UI, which then triggers the Scheduler service to execute that task again and complete it if successful. This support provides the necessary error handling mechanism and ensures the completion of Workflow. Workflow UI service gets all the required information from Workflow Specification Service and Scheduler Service using the below HTTP APIs.

- GET /workflowSpec - used to get all the workflow details from the Workflow Specification service
- GET /workflow - used to get all the status of the workflow i.e. Completed/In Progress/Failed
- POST /workflow/start - used to start the particular workflow
- GET /workflow/:workflowId - to get the status of a workflow instance



1	Procure Utensils workflow	In Progress
2	Food Delivery workflow	Completed
3	Food Delivery workflow	In Progress
4	Procure Utensils workflow	In Progress
5	User Registration workflow	In Progress

Figure 8. Workflow UI Service

D. Workflow Tasks

Figure 9 shows the Workflow Tasks. These are the set of tasks which the user defines for a particular Workflow and their implementations are provided as Docker Images. It is completely up to the users to design and develop their tasks according to their requirements. For example, it can be with or without UI, or some backend activity. The user has the freedom to choose any programming language for implementing the Task (i.e. Task is independent of the Programming Language). However, the user has to follow these 3 simple contract guidelines for the task:

- Provide Docker image for the task to be run.
- Implement POST /startTask API, which will be used by the Scheduler to start this task.
- Invoke POST /task/complete API which is exposed by the scheduler service to mark the task as complete.

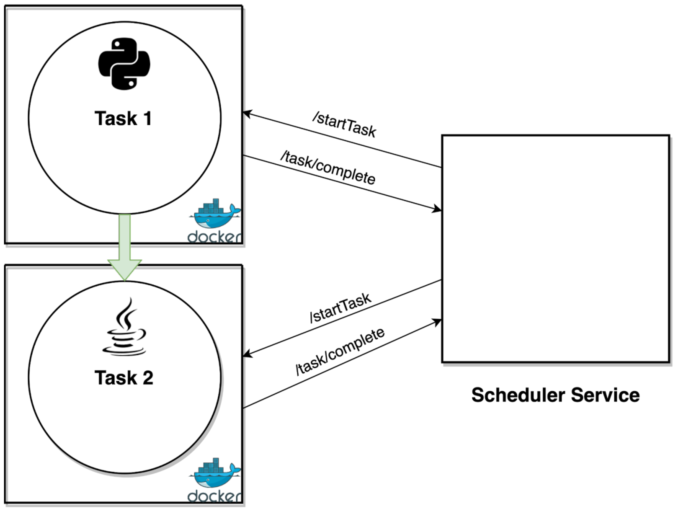


Figure 9. Workflow Tasks

As explained in the scheduler service architecture, the tasks can share data using key-value attributes. The request for startTask provides these attributes to the task, and the tasks can add or update attributes through the completeTask API.

VII. Final Product

To run this project on local minikube cluster, Workflow Specification Service needs to be deployed. It can be done by applying kubernetes deployment descriptors.

```
$ kubectl apply -f app-config-map.yaml
$ kubectl apply -f service-account.yaml
$ kubectl apply -f db-deployment_pv.yaml
$ kubectl apply -f app-deployment.yaml
```

By default this service is configured to run on 30000 node port. After the service is up and running, it can be accessed using minikube's host-only IP. You can get this IP by using the command:

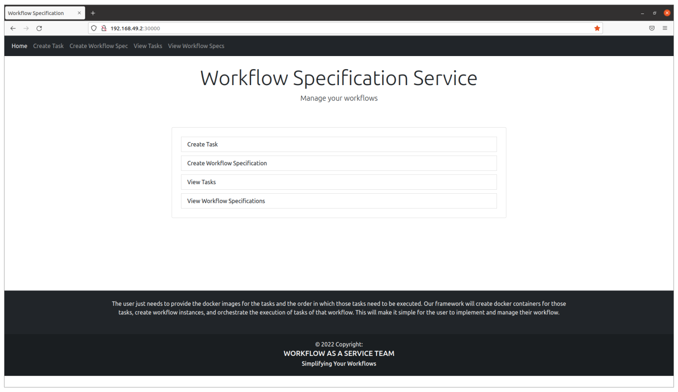
```
$ minikube ip
```

Using this IP address, the application can be accessed (replace Minikube IP with the IP received from above command):

```
http://<Minikube IP>:30000
```

Figure 10 shows the landing page, from where you can start creating task and workflow specifications and deploy them on your local minikube directly from this user interface.

A. User Interface


Workflow Specification
192.168.49.2:30000
Home Create Task Create Workflow Spec View Tasks View Workflow Specs

Workflow Specification Service

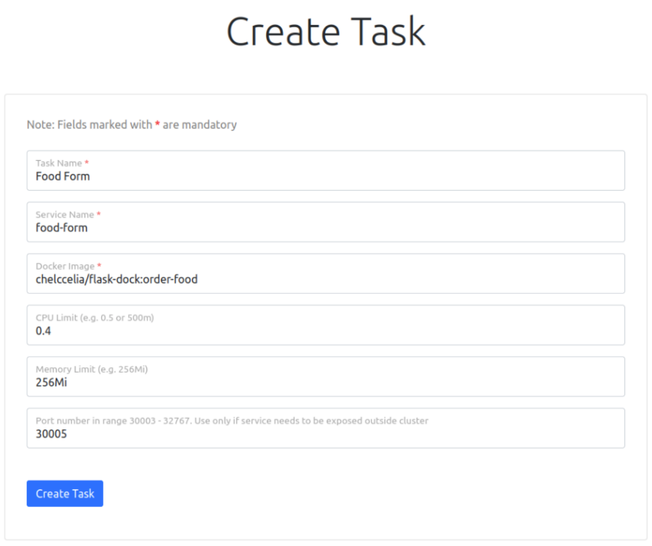
Manage your workflows

Create Task
Create Workflow Specification
View Tasks
View Workflow Specifications

The user just needs to provide the docker images for the tasks and the order in which those tasks need to be executed. Our framework will create docker containers for those tasks, create workflow instances, and orchestrate the execution of tasks of that workflow. This will make it simple for the user to implement and manage their workflow.

© 2022 Copyright:
WORKFLOW AS A SERVICE TEAM
Simplifying Your Workflows

Figure 10. Landing Page of Workflow-as-a-Service. From here, users have the option to create a task, workflow specification, view loaded tasks, and view previously configured workflow specifications

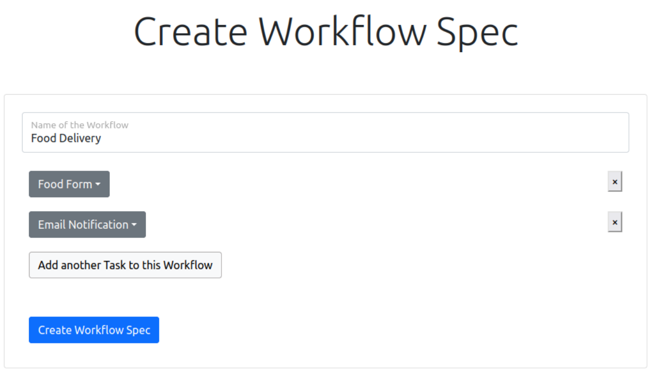

Create Task

Note: Fields marked with * are mandatory

Task Name *: Food Form
Service Name *: food-form
Docker Image *: chelccelia/flask-dock:order-food
CPU Limit (e.g. 0.5 or 500m): 0.4
Memory Limit (e.g. 256Mi): 256Mi
Port number in range 30003 - 32767. Use only if service needs to be exposed outside cluster: 30005

Create Task

Figure 11. Create Workflow Task. In this page, users can upload their workflow tasks as docker images, and define any system requirements, like CPU and memory.

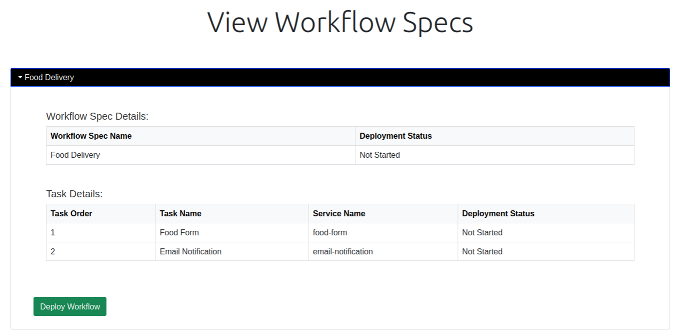

Create Workflow Spec

Name of the Workflow: Food Delivery

Food Form ▾ ×
Email Notification ▾ ×
Add another Task to this Workflow

Create Workflow Spec

Figure 12. Configuring Workflow Order. In this page, users can configure the order of the uploaded docker images and create the workflow.


View Workflow Specs

▾ Food Delivery

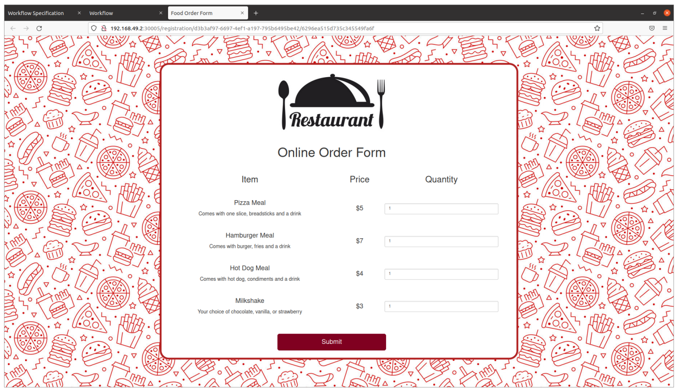
Workflow Spec Details:

Workflow Spec Name	Deployment Status
Food Delivery	Not Started

Task Details:

Task Order	Task Name	Service Name	Deployment Status
1	Food Form	food-form	Not Started
2	Email Notification	email-notification	Not Started

Figure 13. View of a Given Workflow Specification. This page provides all details related to a specific workflow, including the workflow name, deployment status, and list of tasks that are currently part of the workflow. The example in the figure shows a workflow specification for a Food Delivery Order Service.



Online Order Form

Item	Price	Quantity
Pizza Meal Comes with one slice, breadsticks and a drink	$5	1
Hamburger Meal Comes with burger, fries and a drink	$7	1
Hot Dog Meal Comes with hot dog, condiments and a drink	$4	1
Milkshake Your choice of chocolate, vanilla, or strawberry	$3	1

Figure 14. Example Task for a Food Delivery Workflow. The workflow could begin by displaying a form similar like in the figure it the user, and only continue to the next task in the flow once the form is submitted.

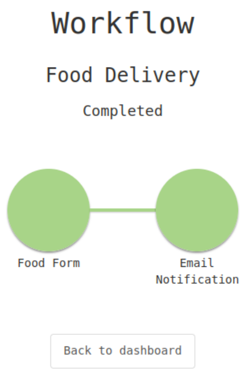


Figure 15. Workflow Complete Visual. When a task is complete, its corresponding circle visual turns green as seen by the two green circles in the figure.

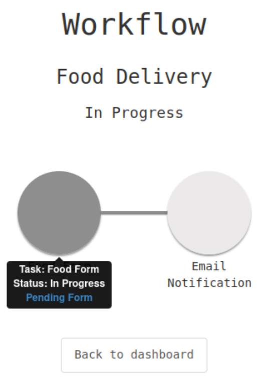


Figure 16. Task In-Progress Workflow Visual. If a task is waiting, the corresponding circle is a light grey color. When in progress, the task circle turns dark grey, and reveals a pop-up informing the user of the status of the task and any blockers that need to be addressed, such as the pending completion of a form as shown in the figure.

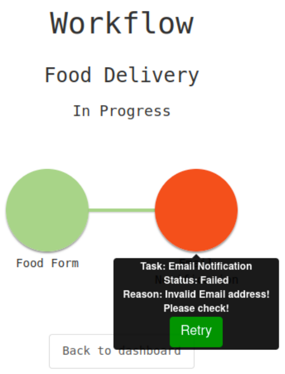


Figure 17. Task Error Workflow Visual. When a task is complete, its corresponding circle visual turns green as seen by the left circle in the figure. However, if the task fails to complete or reaches an error, the corresponding circle becomes red, and provides the user with a potential reason for the error.

VIII. Analysis

Since all the services are deployed inside a kubernetes cluster, they can benefit from the advantages that kubernetes offer. The services are fault tolerant because the kubernetes cluster always maintains the required state of the system. Even if a pod stops, another pod starts running. The services are also scalable. Although we did not implement auto-scaling in this project, one can easily add a kubernetes HorizontalPodAutoscaler [9] to enable auto-scaling. Moreover, the user also gets to configure the CPU and Memory requirements at the task specification level. So, resource intensive tasks can be assigned more resources. We have not added any price related options, but usually cloud platforms have interdependent pricing and resource/auto-scaling options. Another benefit of our application architecture is that each user gets their own separate instances of scheduler-service and ui-service. This gives the flexibility to provide different UI versions to different users, or customize scheduler-service according to user requirements if the need arises.

IX. Future Work

1. Currently the user specifies the tasks in a sequential order. It can be extended to have multiple branches in a workflow which could be executed based on conditions given (i.e. if/else branches).

2. Dependencies can be provided between the workflows. There are many scenarios where two different workflow types may be dependent on each other for their completion. For example: there may a workflow for a product ordered by customer in which delivery is shown as one single step. But in the back-end, this delivery step may be a workflow on its own.

3. As of now tasks are created through provided docker images. This is good for long-lived tasks and services that need to be running forever. However, the product can also be extended to create Tasks through serverless functions (e.g. OpenFaaS [6]) which will be good for short-lived tasks.

4. Providing a software development toolkit (SDK) for easier and quicker development and integration. This will eliminate boilerplate code required for implementing startTask API and calling completeTask API. The user can then use these SDKs and focus completely on the logic of tasks rather than worrying about its integration with the system.

5. Provide auto-scaling options for tasks. The current implementation does not auto-scale since the resources are separately assigned by the user. This can be modified to bring in the flexibility to configure auto-scaling, thereby making the services elastic. Since kubernetes is used, this can be done using kubernetes HorizontalPodAutoscaler [9].

X. Conclusion

Workflow-as-a-Service solves the general problem of dealing with the complexities of deploying and orchestrating workflow with numerous services. Through the use of a microservice architecture and REST APIs, this platform application allows users to seamlessly deploy workflows and specify the order of tasks in a given workflow. By making use of this platform, users can focus on the development of the microservices rather than having to worry about the underlying complexities of the workflow, greatly saving development and deployment time. The user interface solution makes it simple to configure tasks and workflows and visualize the workflow instances as they progress. While the application supports only simple workflows as of now, there is scope to support more complex workflows and a wider range of cloud technologies which can be considered in the future versions.

XI. Github Repository

https://github.com/ccfernandes/workflow-as-a-service

Only workflow-specification service needs to be deployed. All other services can be deployed through the workflow-specification service user interface. Instructions to deploy workflow-specification service can be found inside the README file in workflow-specification directory.

References

[1] "What Is a Workflow?: Types, Benefits and Overview" Kissflow, 23 May 2022, https://kissflow.com/workflow/what-is-a-workflow/.

[2] "What Is Amazon Simple Workflow Service?" Amazon Simple Workflow Service Developer Guide, Amazon Web Servics, https://docs.aws.amazon.com/amazonswf/latest/developerguide/swf-welcome.html.

[3] fabric8io java client: https://github.com/fabric8io/kubernetes-client

[4] https://spring.io/guides/tutorials/rest/

[5] https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

[6] https://www.openfaas.com/

[7] https://jquery.com/

[8] https://getbootstrap.com/

[9] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale

[10] https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/