Workflow management for scientific research computing with Tapis Workflows

Architecture and Design Decisions behind Software for Research Computing Pipelines

Nathan Freeman
Texas Advanced Computing Center
Austin, Texas, USA
nfreeman@tacc.utexas.edu

Joe Stubbs
Texas Advanced Computing Center
Austin, Texas, USA
jstubbs@tacc.utexas.edu

Richard Cardone
Texas Advanced Computing Center
Austin, Texas, USA
rcardone@tacc.utexas.edu

ABSTRACT

Developing research computing workflows often demands significant understanding of DevOps tooling and related software design patterns, requiring researchers to spend time learning skills that are often outside of the scope of their domain expertise. In late 2021, we began development of the Tapis Workflows API to address these issues. Tapis Workflows provides researchers with a tool that simplifies the creation of their workflows by abstracting away the complexities of the underlying technologies behind a user-friendly API that integrates with HPC resources available at any institution with a Tapis deployment. Tapis Workflows Beta is slated to be released by the end of April 2022. In this paper, we discuss the high level system architecture of Tapis Workflows, the project structure, terminology and concepts employed in the project, use cases, design and development challenges, and solutions we chose to overcome them.

CCS CONCEPTS

• **Software and its engineering**;

KEYWORDS

workflows, containers, API, HPC

ACM Reference Format:
Nathan Freeman, Joe Stubbs, and Richard Cardone. 2022. Workflow management for scientific research computing with Tapis Workflows: Architecture and Design Decisions behind Software for Research Computing Pipelines. In *Practice and Experience in Advanced Research Computing (PEARC '22), July 10–14, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3491418.3535142

1 INTRODUCTION

Tapis Workflows is an official Tapis API [17] designed to automate the execution of research computing tasks in the Tapis ecosystem, coordinate their inputs and outputs, and communicate the resultant data or product with external resources. Tapis Workflows offers users the ability to design workflows in which they can build container images, run scientific computing jobs, and trigger subsequent workflow tasks on independent systems with webhook notifications.

In this section, we introduce the terminology and important concepts employed in the project, namely pipelines (workflows), actions (tasks) and their functions, and other entities related to resource ownership and pipeline operation.

1.1 Actions

Actions are discrete tasks performed during the execution of a workflow. They can be represented as nodes on a directed acyclic graph (DAG), with the order of their execution determined by their dependencies, and where all actions without dependencies are executed first.

Tapis Workflows currently supports 3 types of actions—referred to as action primitives—that perform functions common in research computing workflows; The image-build action, webhook notification action, and tapis-job action.

1.1.1 Image Build. The image-build action allows users to build container images from configuration files (Dockerfiles) located in code repositories and push the built image to remote registries. The image build action uses Kaniko [6], a software for building container images that is executed inside of a container or as a Job in a Kubernetes cluster [7].

1.1.2 Webhook Notification. The webhook notification action allows users to make HTTP requests to a specified endpoint with a payload containing data from computation of previous workflow actions. This action can be used to mutate external data sources, or fetch data from another API and make it available to subsequent actions. Currently, the only supported HTTP methods are those associated with CRUD operations (GET, POST, PUT, PATCH, and DELETE).

1.1.3 Tapis Job. The tapis-job action acts as a proxy for the job submission operation of the Tapis Jobs API, a specialized service for running containerized applications across large computational resources [15]. When defining a tapis-job action, users must provide a job submission schema just as they would if they were submitting a job directly with the Jobs Service [8]. If the result of the tapis-job action is important for subsequent actions in the pipeline, users can specify the 'poll-until-complete' behavior, which will block all dependent actions until the submitted job comes to some terminal state.

PEARC '22, July 10–14, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9161-0/22/07…$15.00
https://doi.org/10.1145/3491418.3535142

1.1.4 Future Actions. After the beta release of Tapis Workflows, development will be started on two additional action primitives. The tapis-actor action, container-run action, and the script action. More information on these actions can be found in sections 5.1, 5.2, and 5.3 respectively.

1.2 Groups, Users, and Identities

Groups are collections of users that own workflow resources. For all tasks that require secure access to remote resources—such as the image-build and webhook notification actions—user's can create mappings to external identities along with the credentials required to access those resources.

1.3 Pipelines and Directives

A Pipeline is a unique collection of actions (tasks) that define a workflow. Operations on pipelines and associated resources are only available to users that belong to the group that owns them.

Pipelines can be triggered in two ways; Direct API calls and webhook notifications from source control platforms. Users can provide special commands (Directives) in the request body of API calls—or in the case of webhook notification, the commit message—that provide users with a way to invoke optional or custom behavior of pipeline and action executions. Directives enables users to perform mock-runs of their pipelines to ensure their actions run in the correct order, tag their images with custom values during image builds, and enable caching to improve subsequent image build-times.

2 ARCHITECTURE

Tapis Workflows employs a microservices architecture, providing scalability, loose coupling between services, and ease of development. The project consists of 4 main components; an API, pipeline service, message broker, and database. What follows is an explanation of the project's source code structure and the technologies and relationship between the major components that make up Tapis Workflows.

2.1 Project Structure

The source code for each component is contained in a single repository, following the monorepo pattern [11]. Monorepos are a useful project structure for software that uses a microservice architecture and organizations that maintain codebases for many interconnected applications.

This pattern was chosen over the polyrepo pattern for three primary reasons. First, the Pipelines Service and API share common utilities responsible for DAG and request validation. Second, because each service is located in the same repository, it allows us to take a holistic approach to testing to ensure functionality is maintained between services. Third, it simplifies the development process as code changes to multiple services can be encapsulated in a single commit.

The downsides to using monorepo—namely the issues associated with scaling and lack of granular access control on individual services—are not projected to complicate future development efforts.

2.2 Components

The 4 previously mentioned components will be deployed in Kubernetes for the development, staging, and production environments, and docker-compose [14] for local development environments.

2.2.1 API. Tapis Workflows exposes a Python/Django REST API that allows users to create pipelines, actions, and related resources to facilitate research computing workflows. The Django framework was chosen because it comes furnished with a collection of tools that make API development fast and secure, and works well with most major databases [5].

One of the primary reasons for using Django is its built-in object-relational mapping (ORM). It significantly reduces the amount of code necessary to query and mutate data sources, albeit at a cost to performance. Taking into consideration the current and projected use cases, the fact that the API is neither read nor write heavy, and the vast majority of the processing workload will be handled by Pipeline Service, the trade-off of API performance for speed and ease of development was deemed acceptable.

The Tapis Workflows API—like other Tapis APIs—provides an OpenAPI (v3)-compliant specification. This means that it is compatible with the existing Python (tapipy) [9] and Java SDKs for interacting with the service.

2.2.2 Pipelines Service. The Pipeline Service is a containerized component written in Python that is responsible for managing the life-cycle of pipelines, their actions, and validating their inputs and outputs. As previously stated, the order of execution is determined by the dependencies specified on the action object when it is created. The Pipeline Service is capable of executing actions within a single pipeline both serially and concurrently. For all short-running executions such as with webhook notifications, and Tapis job and actor actions—when poll-until-complete behavior is not specified, the processing occurs directly on the Pipeline Service container. For all CPU and I/O bound executions, the action is run in a separate container (i.e. Job in Kubernetes).

This service is triggered by messages containing pipeline specifications sent from the API via RabbitMQ. It then dispatches the initial action(s) and queues up the remaining actions. Once an action finishes executing and the expected output of that action is verified and validated, all queued actions that depend on the completed action are dispatched—provided they are not dependent on an action that is currently running.

2.2.3 Database. The data model for Tapis Workflows contains a mixture of both structured and unstructured data, with a strong bias towards the former. For this reason, a decision was made to use MySQL as the primary database.

3 USE CASES

The following section discusses the early adopter use cases for the Workflows API.

3.1 SCINCO's JupyterHub

Tapis Workflows was initially designed to serve as the CI (continuous integration) system for the TACC's SCINCO JupyterHub platform [16]. When users of this platform publish changes to their

Jupyter Notebook Dockerfiles, the CI system receives a webhook notification from the source control platform, pulls the code containing the updated configuration file to build a new image, pushes that image to a destination registry(Dockerhub), and sends a webhook notification to an endpoint on the Kubernetes cluster hosting the JupyterHub, pulling in the new image and restarting the service.

3.1.1 HETDEX. HETDEX (The Hobby-Eberly Telescope Dark Energy Experiment) is a collaborative effort by The University of Texas at Austin, the Pennsylvania State University, Ludwig-Maximilians-Universität München, and Georg-August-Universität Göttingen, aimed at measuring the cosmological state of the early universe to look for potential evolution in dark energy by measuring the clustering of over 1 million distant galaxies using the Hobby-Eberly Telescope at the McDonald Observatory [10]. The HETDEX group maintains a Jupyter Notebook image used for interactive data analysis and visualization. They will employ Tapis Workflows to build the container image and push it to an image registry.

3.2 DARPA SHADE

SHADE(Stabilizing Hostilities through Arbitration and Diplomatic Engagement) is a DARPA AI Exploration initiative that seeks to explore technologies that assist in multi-party decision-making, collaboration, and negotiation [12]. The DARPA SHADE program will leverage Tapis Workflows to rapidly build, test, and iteratively improve AI agents designed to compete in the game of Diplomacy [1].

4 CHALLENGES AND SOLUTIONS

4.1 Existing Technologies vs. Custom Solution

There are existing technologies that offer the same functionality as Tapis Workflows and solve many of the same problems. For the CI Pipeline use case, both Github and Gitlab offer their own solutions; Github Actions and Gitlab CI, respectively. However, they come with limitations.

For academic institutions, access to premium features can be prohibitively expensive. Additionally, the memory and storage allotments for even the most robust plans do not satisfy the requirements for some research computing use cases. For example, TACC's Jupyterhub platform routinely requires image builds that use in excess of 16GB of memory and take 15 to 25 minutes to build. TACC has its own self-hosted instance of Gitlab, but does not meet the minimum memory requirements necessary to run a single image build for any of the previously mentioned use cases. Therefore, a custom solution that runs on dedicated hardware appeared to be the clear option.

4.2 Scalability and Action Scheduling

In its current state, Tapis Workflows adopts a first-in-first-out approach to pipeline and action execution. This is sufficient to satisfy the requirements for the initial use cases, but will inevitably become an issue when users submit multiple pipelines with actions that run over long periods of time or consume large amounts of computational resources(Memory, CPU, etc).

The proposed solution is to develop a mechanism that enables the pipeline service to run actions from multiple pipelines concurrently and prioritize actions based on their estimated resource consumption, execution time, and order in which they are submitted. Additionally, users will be required to specify a TTL(time-to-live) on their action definition that sets a limit on the amount of time it is permitted to run. Actions that exceed their TTL will be terminated.

4.3 Securing Sensitive Data

The safe handling of sensitive data is one of the foremost concerns of Tapis Workflows. For some actions—the image-build action, for example—users are required to provide credentials that permit access to restricted external resources such as code repositories and image registries. In order to reduce the workload inherent in manual authentication for every run of a pipeline, a design decision was made to persist those credentials. A robust solution was needed to ensure their safe storage. For this, the project makes use of Tapis Security Kernel(SK) [3] an authorization and secrets management service backed by HashiCorp Vault [2].

4.4 Action Inputs, Outputs, and Terminal State

One of the biggest challenges we face in the development of Tapis workflows is how to manage the input and output between action dependencies, and how to qualify an action as having executed successfully, particularly those of containerized actions (i.e. container-run).

There are two criteria that can be used when qualifying the terminal state of containerized actions. The container's exit code, and whether the data or products produced by an action match the parameters specified by an actions output definition. Tapis Workflows currently only uses the first criterion. For all actions that do not specify any outputs, a non-zero exit code of the container executing the action is sufficient for qualifying success. For actions that specify output, that output must meet the quantity and type constraints enumerated in the action's output definition.

5 FUTURE WORK

Along with the tapis-job action mentioned above, there are plans to integrate with additional services in the Tapis ecosystem.

5.1 Tapis Actor Action

This action will leverage the Abaco API [4], a container-based function-as-a-service platform. This will enable users to execute function primitives and utilize the results with other actions, including other tapis-actor actions. Abaco writes the results of a function to the stdout of the container in which it ran. If the result reported by Abaco does not conform to the data type specified in the output definition of the action, the action will fail.

5.2 Container Run Action

The container-run action will allow users to run containers using their own images stored in remote repositories. Once the container has finished running, its files and data written to stdout will be validated against the action's output definition to determine whether the action executed successfully.

5.3 Script Action

The script-action will enable users to write arbitrary code in the "command" property of an action definition following the OpenWDL syntax [13]. Users will choose from a selection of programming languages and runtimes in which to execute that code. This provides them with the ability to transform the output data of one action into a structure that conforms to the input definition of another, increasing the re-usability of existing actions. For security reasons, this action will be wholly restricted from access to external networks.

6 CONCLUSION

A solution to managing research computing pipelines in HPC ecosystems has long been sought after, and Tapis Workflows seeks to lay the groundwork for that solution to be realized. We will continue to work closely with various teams at TACC and other institutions to understand their workflows, and develop functionality to meet their use cases. In this paper, we have covered how Tapis Workflows approaches the problem of workflow management, the concepts and technologies that drive its development, and the solutions to challenges we expect to face as this project progresses.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation Office of Advanced CyberInfrastructure, the Tapis Framework:[1931439 and 1931575]

REFERENCES

[1] 2009. *The Invention of Diplomacy.* https://web.archive.org/web/20090910012615/http://www.diplom.org/~diparch/resources/calhamer/invention.htm Last access: 2022-04-04.
[2] 2019. *HashiCorp Vault.* https://www.vaultproject.io/ Last access: 2022-4-4.
[3] 2019. *Tapis Security Kernel.* https://tapis.readthedocs.io/en/latest/technical/security.html Last access: 2022-4-1.
[4] 2020. *Actors.* https://tapis.readthedocs.io/en/latest/technical/actors.html Last access: 2022-3-31.
[5] 2020. *Django: Supported Databases.* https://docs.djangoproject.com/en/4.0/ref/databases/ Last access: 2022-03-31.
[6] 2020. *Kaniko.* https://github.com/GoogleContainerTools/kaniko Last access: 2022-03-31.
[7] 2020. *Kubernetes.* https://kubernetes.io/ Last access: 2022-3-31.
[8] 2020. *Submitting a Tapis Job.* https://tapis.readthedocs.io/en/latest/technical/jobs.html#the-job-submission-request Last access: 2022-03-31.
[9] 2020. *Tapipy.* https://github.com/tapis-project/tapipy Last access: 2022-03-31.
[10] 2021. *The Hobby-Eberly Telescope Dark Energy Experiment (HETDEX) Survey Design, Reductions, and Detections.* https://ui.adsabs.harvard.edu/abs/2021ApJ...923..217G/abstract Last access: 2022-04-05.
[11] 2021. *Monorepo vs. Polyrepo.* https://github.com/joelparkerhenderson/monorepo-vs-polyrepo Last access: 2022-03-31.
[12] 2021. *SHADE: Federal Contract Opportunity.* https://govtribe.com/opportunity/federal-contract-opportunity/shade-darpapa210403 Last access: 2022-03-31.
[13] 2022. *OpenWDL: Defining Commands.* https://github.com/openwdl/wdl/blob/main/versions/1.0/SPEC.md#command-section Last access: 2022-3-31.
[14] 2022. *An Overview of Docker Compose.* https://docs.docker.com/compose/ Last access: 2022-03-31.
[15] 2022. *Tapis Jobs API.* https://tapis.readthedocs.io/en/latest/technical/jobs.html Last access: 2022-03-31.
[16] Joe Stubbs et al. 2020. Integrating Jupyter into Research Computing Ecosystems. Proceedings of the Practice and Experience on Advanced Research Computing, PEARC 2020.
[17] Joe Stubbs, Richard Cardone, Mike Packard, Anagha Jamthe, Smruti Padhy, Steve Terry, Julia Looney, Joseph Meiring, Steve Black, Maytal Dahan, Sean Cleveland, and Gwen Jacobs. 2020. Tapis: An API Platform for Reproducible, Distributed Computational Research. (2020). submitted.