Development and demonstration of a proof-of-concept for the integration of programming frameworks for high performance computing into a container-based workflow orchestrator.

2. Project Paper

submitted on October 30, 2023

Faculty for Business and Health

Degree program in Business Informatics

International Management for Business and Information Technology

by

JON ECKERTH

Practice supervisor :

⟨ Hewlett Packard Labs ⟩
⟨ Harumi Kuno, PhD ⟩
⟨ Principal Research Scientist ⟩

DHBW Stuttgart:

⟨ Dominic Viola ⟩

Signature of the practice supervisor:

Statement of Integrity

Hereby I solemnly declare:

1. that this 2. Project Paper titled *"Development and demonstration of a proof-of-concept for the integration of programming frameworks for high performance computing into a container-based workflow orchestrator."* is entirely the product of my own scholarly work, unless otherwise indicated in the text or references, or acknowledged below;
2. I have indicated the thoughts adopted directly or indirectly from other sources at the appropriate places within the document;
3. this 2. Project Paper has not been submitted either in whole or part, for a degree at this or any other university or institution;
4. I have not published this 2. Project Paper in the past;
5. the printed version is equivalent to the submitted electronic one.

I am aware that a dishonest declaration will entail legal consequences.

Dublin, October 30, 2023

Jon Eckerth

Abstract:

TODO

Contents

List of abbreviations

AI Artificial Intelligence

CC Cloud Computing

CI/CD Continuous Integration/Continuous Delivery

CLASP Cloud Application Services Platform

CNCF Cloud Native Computing Foundation

CNI Container Network Interface

CWL Common Workflow Language

FAM Fabric Attached Memory

GASNet Global Address Space Networking

HPC High Performance Computing

HPE Hewlett Packard Enterprise

IaC Infrastructure as Code

IP Interlectual Property

k8s Kubernetes

LCP Loosely Coupled Problems

MAUT Multi-attribute Utility Therory

ML Machine Learning

PFS Pachyderm File System

PoC Proof of Concept

PoC Proof of Concept

PV Persistent Volume

RBAC Role Based Access Control

RDMA Remote Direct Memory Access

ROC Rank order centroid

RR Rank Reciprocal

RS Rank Sum

SMART-ER SMART Exploiting Ranks

SMART Simple Multiattribute Rating Technique

SME Subject Matter Expert

SSO Singele Sign On

TCP Tightly Coupled Problems

UDP User Datagram Protocol

VM Virtual Machine

WSM Weighted Sum Model

List of Figures

List of Tables

1 Introduction

In this section, the underlying motivation of this project is explained. Furthermore, the problems which will be addressed by this project are described, which serve as the basis for the research questions which will guide this project and ultimately result in solutions and further questions which are listed in the contributions section and discussed in the conclusion.

1.1 Motivation

The proliferation of "Big Data" has led to the need to compute, analyse, and visualize ever-increasing amounts of datasets, which themselves are getting more and more complex, has led to an ever increasing demand for more efficient and quicker ways to process data.

Both the High Performance Computing (HPC) and the Cloud Computing (CC) community have been working on solutions to distribute and parallelize computations for decades, both with their own approaches and solutions to their respective problems.

While the HPC community has been putting a lot of effort into developing new and extremely efficient ways to parallelize computations, the CC community has been focusing on improving the flexibility, scalability and resilience of their solutions as well as improving the ease of use for their developers and users.

Both used to be very distinct and separate communities due to their very different usecases, while the HPC community was mostly concerned with scientific computing and simulations of physical phenomena, the CC community is mostly concerned with providing a reliable and easily up and down scalable infrastructure for the industry and businesses.

Now with the advent of Machine Learning (ML) and Artificial Intelligence (AI) the two communities are starting to converge, as the ML and AI community is adopting the tools and techniques of both communities to solve their problems as they see fit.

But this convergence of the two is not without its problems, being developed in two coexisting and separate communities, the tools and techniques of both communities are not always compatible with each other, the goal of this project is to find a way to bridge this gap and to find a way to combine the best of both worlds.

1.2 Problem Statement

The following key problems have emerged from the convergence of High Performance Computing (HPC) and Cloud Computing (CC) communities, especially in the context of Machine Learning (ML) and Artificial Intelligence (AI) research:

- **Workload Resilience and Fault Tolerance in HPC:** HPC systems often lack mechanisms to recover from task failures within larger jobs, running for an extended time. When a component task fails, it can invalidate the entire computation, requiring a restart from scratch. This need for resilient failover and verification strategies as well as the need to avoid computational wastage is a key challenge for HPC systems, especially with every increasing system sizes and complexity.[3]
- **Environment/Package Management in HPC:** HPC systems are notorious for their complex package management systems. As having a shared infrastructure between many users each with their own specific needs and requirements of different versions of packages, libaries and software, all the while sharing a common environment. Many solutions to this problem have been developed, each with their own advantages and disadvantages.[4][5][6][7]
- **Portability Issues with HPC:** Tieing in with the previous point, HPC systems are often designed to be optimized for specific hardware as well as having a very specific software stack. This makes the portability of applications between different HPC systems very difficult and often infeasible.[8] This lack of portability contrasts sharply with the more platform-agnostic nature of CC environments, where the containerization of applications has become the norm for ensuring portability.
- **Scalability and Flexibility in HPC:** Due to its direct access to the hardware and very specific hardware needs, HPC systems are often hard to dynamically scale and inflexible. while CC systems are designed to be easily scalable and flexible and are often designed to be hardware-agnostic and abstract away the underlying hardware. This becomes especially relevant in the context of heterogeneous hardware, where the hardware is not uniform and consists of different types of hardware, which is becoming more and more common in the context of ML and AI research.
- **Lack of Interconnected Problem Solving in CC:** The workloads traditionally deployed on CC systems are often independent of each other, like load balancing, web hosting, etc. This is in stark contrast to the interconnected nature of HPC workloads, where each part of the input data is dependent on the other parts of the input data, such that all nodes of the system need to be able to communicate with each other.

[3] Egwutuoha et al. 2013
[4] Dubois/Epperly/Kumfert 2003
[5] Bzeznik et al. 2017
[6] Gamblin et al. 2015
[7] Hoste et al. 2012
[8] Canon/Younge 2019, p. 50

- **Provenance and Reproducibility:** Another need that is becoming more and more important in the context of ML and AI research is the need for provenance and reproducibility of results. Being able to tell which data was used to train the model, is of ever-increasing importance as the influence the resulting models have on our lives increases as well as the data used to train the model. This is especially important since it is crucial to ensure that the data is not biased, outdated, or otherwise flawed, which could lead to incorrect predictions, decisions, or recommendations. in addition various data sources, from images to text, may have copyright restrictions that, when overlooked, can lead to significant legal complications.
- **Versioning Limitations:** The dynamic nature of ML and AI research necessitates robust versioning solutions for data, configurations and code. While CC has developed many solutions to this problem over the years, making them their own subsection of the ecosystem, namely Continuous Integration/Continuous Delivery (CI/CD) tools for the testing and deployment of applications aswell as Infrastructure as Code (IaC) tools for the deployment of infrastructure. While many solutions have been developed for the one-off deployment of HPC systems, the dynamic nature of CC systems necessitates a more robust solution to this problem, from which the HPC community could benefit aswell.

1.3 Research Questions

To address the aforementioned problems, to bridge the gap between the two paradigms and to combine the best of both worlds, an integration of the two paradigms is needed. This was accomplished by integrating a HPC framework called Arkouda[9] into a container based CC workflow management tool called 'Pachyderm'[10] and integrating both with the supporting infrastructure the CC system enables us to use. This process of integration and prototyping as well as the explanation of the underlying concepts and technologies will be the focus of this project.

[9] Merrill/Reus/Neumann 2019
[10] *Home Page | Pachyderm* 2023

- **RQ1:** *How can a high-performance computing framework be effectively integrated into a container-based workflow management tool?*
- **RQ2:** *What are the opportunities for improving the integration of high-performance computing frameworks with container-based workflow management tools?*

1.4 Contributions

In order to address the problems stated above, find answers to the research questions and to bridge the gap between the two paradigms, the following contributions were made:

- **C1:** *An analysis of the problem space and existing solution, within the constraints of time, resources and businesses needs.*
- **C2:** *A prototype implementation combining the 'Arkouda' framework with the k8s based workflow orchestrator 'Pachyderm' running on the Heydar Cluster*
- **C3:** *Further integrations of tools from both sides of the spectrum, addressing many of aforementioned pain-points*

2 Methodology

2.1 Prototyping

TODO

Needs to have a methodology from the Spectrum of Methodologies for Business information systems[11]

Argumentation why this project is centrally a Prototyping project:

- The research questions are directly inspired bt the needs of the customer
- The limitations and the scope are both defined by the available resources of the business unit as well as the time constraints of the project and the available know-how
- Based[12] can be classified as a presentation prototype in which we do a does a vertical integration of many different sysytems, according to budde this can be described as a vertical interface, as it reaches through the entire stack of technological abstraction[13]
- to create this prototype we will be using Which will be using spiral modle[14]

[11] Wilde/Hess w.y.
[12] Budde et al. 1992, p. 91
[13] Budde et al. 1992, p. 94
[14] Boehm 1988

2.2 Decision Making

As previously described, the methodology of Prototyping benefits from a very tight loop of iterations between the different phases of the project. While this is highly effective in producing a good end result, it can also take many iterations and a lot of experimentation until an adequate tool or solution has been found. Given the constraints of a limited time frame for this project, it becomes crucial to use this time as efficiently as possible. Sometimes, when the time does not permit a thorough exploration of

To ensure that the decisions made are the most optimal within the constraints of the available information, adopting a systematic, replicable, and transparent decision-making process becomes essential. Over the years, various frameworks have been crafted to guide decision-making, particularly when information is complex and multi-dimensional.

2.2.1 Weighted Sum Model

Evangelos Triantaphyllou suggests that the Weighted Sum Model (WSM) is in practice the most used and most relevant decision-making framework[15]. The WSM method, by design, mandates the assignment of specific weights to each criterion based on its relevance. Subsequent to this, every alternative is evaluated based on these weighted criteria, resulting in a cumulative score. The alternative with the highest score is therefore the optimal choice.

$$A_i^{WSM-score} = \sum_{j=1}^{n} w_j a_{ij} \quad for\ i = 1, 2, 3, \ldots, m.$$

Abb. 1: Formula for calculating the WSM score[16]

Where:

- w_j: This represents the weight assigned to the j-th criterion. Weights are determined by the decision-makers based on the relative importance of each criterion. They should be normalized (i.e., the sum of all weights should be 1 or 100

- a_{ij}: This represents the score or rating of the i-th alternative concerning the j-th criterion. This score is an assessment of how well the alternative meets or satisfies the specific criterion.

This method, despite its simplicity and direct approach, isn't without limitations. One notable drawback is its dependence on dimensionless scales. For the weights to properly reflect the criteria's importance, the scores need to be on a common, dimensionless scale, a detail not always feasible or convenient in practice.

[15] Triantaphyllou 2000, p. 1
[16] *Weighted Sum Model* 2022

2.2.2 Simple Multi-Attribute Rating Technique

In contrast to the WSM , which predominantly utilizes a direct mathematical approach to rank alternatives based on their weighted sum scores, the Simple Multiattribute Rating Technique (SMART) methodology offers a more comprehensive approach to multi-criteria decision-making. While WSM is primarily concerned with simple weighted arithmetic sums, the SMART method dives deeper, ensuring that diverse performance values—both quantitative and qualitative are harmonized and placed on a common scale.

The SMART method, grounded in Multi-attribute Utility Theory (MAUT), provides a structured framework that encompasses more than just the weighting of criteria. It involves:

1. Discernment of vital criteria pertinent to the decision in focus.
2. Weight allocation to each criterion in accordance to its significance.
3. Evaluation of each potential alternative against the identified criteria, culminating in a score.
4. Aggregation of these individual scores via their associated weights, yielding a total score for every alternative.

By adhering to the SMART framework, alternatives can be sequenced based on their aggregated weighted scores. This systematic approach equips decision-makers to choose solutions that align closely with their objectives. The computational formula integral to the SMART method is:

$$x_j = \frac{\sum_{i=1}^{m} w_i a_{ij}}{\sum_{i=1}^{m} w_i}, \quad j = 1, \ldots, n.$$

Abb. 2: Formula for calculating the SMART score[17]

Where:

- x_j Is the overall utility socre for alternative j. The higher the score, the better the alternative, in comparison to the other alternatives.
- a_{ij} Is the utility score for alternative j for the criterion i.
- w_i Is the weight of criterion i.

This method's emphasis on utility functions ensures a more nuanced and adaptable approach to decision-making compared to models like WSM, making it suitable for complex scenarios where criteria and alternatives are diverse in nature[18].

[17] Taken from Fülöp 2005, p. 6
[18] Fülöp 2005, p. 6

2.2.3 SMART Exploiting Ranks

The SMART Exploiting Ranks (SMART-ER) method is a variant of the SMART method that attempts to alleviate the largest issue of the original SMART method, namely the problem of a somewhat arbitrary ranking of the options if no numerical values can be derived.

This method addresses the issue by letting the decision maker simply ranking the different criteria in relation to each other and then normalizing the weights[19]. They propose the different weighting curves.

$$w_i(ROC) = \frac{1}{n}\sum_{j=1}^{n}\frac{1}{j}, \quad i = 1, \ldots, n.$$

Abb. 3: Formula for the ROC weights

The ROC takes the centroid of the rank order and uses the reciprocal of the rank as the weight.

$$w_i(\ RS\ ) = \frac{n+1-i}{n(n+1)/2}, \quad i = 1, \ldots, n.$$

Abb. 4: Formula for the RS weights

The RS uses linear curve where weights are normalized by dividing them by the sum of all weights.

$$w_i(RR) = \left(\frac{\frac{1}{i}}{\sum_{j=1}^{n}\frac{1}{j}}\right),\ rank\ i = 1, \ldots, n,\ option\ j = 1, \ldots, n$$

Abb. 5: Formula for the RR weights

The RR emphasizes the most important criteria by using the reciprocal of the rank as the weight, then normalizing each weight by the sum of all reciprocals.

[18] *Multi-Criteria Decision Analysis for Use in Transport Decision Making* 2014, p. 26
[19] Roberts/Goodwin 2002, p. 296

3 State of the Art

3.1 Containerization

Container Solutions

Software defined Infrastructure

Large Scale Container Orchestration

3.2 High Performance Computing Frameworks

3.2.1 Loosely Coupled Problems

Loosely Coupled Problems (LCP) also known in the industry as "embarrassingly parallel"[20] problems are problems that can be broken up into smaller independent tasks that can be executed in parallel.

tools like Mapreduce and Spark

3.2.2 Tightly Coupled Problems

In contrast to LCP problems, Tightly Coupled Problems (TCP) problems are problems that can not be broken up into smaller independent tasks that can be executed in parallel, instead of working independently, each atomic task needs to communicate at least with one other task. A good example of a TCP problem are the n-body problems, where the position of each body is dependent on the position of all other bodies.

Message Passing Interface (MPI) vs Shared Memory (OpenMP) or Partioned Global Address Space (PGAS)[21]

[20] **smtn**
[21] **smtn**

4 Creation of the Artifact

4.1 Initial Goals

As this project was first and foremost a project, designed to interactively explore the problem space from the perspective of the HPC community, all the while being contained by business requirements and time constraints, the initial goals of this project were very broad and open-ended. At first the initial goal was simply to create a Proof of Concept (PoC) of a realistic workflow engine using the "Arkouda" project, in order to present the Customer with an easily graspable example of its capabilities.

While we are approaching the problem from the perspective of the HPC community, the intended end user of this tool are the data scientists and Subject Matter Experts (SMEs) that are working with the HPC systems, and therefore the tool needs to be designed and selected with the fact in mind that the end user will most likely not be knowledgeable in the field of HPC or the underlying infrastructure.

In the first iteration of the project a preselection of possible Workflow management tools was given from the business side, with the option to increase the scope if the presented tools were not sufficient.

Therefore, the goals of the first iteration of this project was twofold, first to determine which, if any, of the presented tools were suitable for the task at hand, and to determine what would make an adequate PoC for the customer.

The following iterations are split into the tree main aspects of the project and will be discussed in their own subsections. While these steps where happening concurrently, they each address a different aspect of the project and therefore underwent their own iterative processes.

4.2 Overall Structure

As can be seen in figure 6, the artifact is composed of 3 main components, the **Central Workflow Engine** which is responsible for the orchestration of the workflows (center) and interfaces directly with the underlying infrastructure, the **HPC Framework** which is responsible for the execution of TCP workloads (left) and the **Supplementary Services** which aim at improving the usability and accessibility for the end user (right).

All this is build on top of a hardware-agnostic k8s cluster, which is responsible for the orchestration of the different components and the underlying infrastructure.

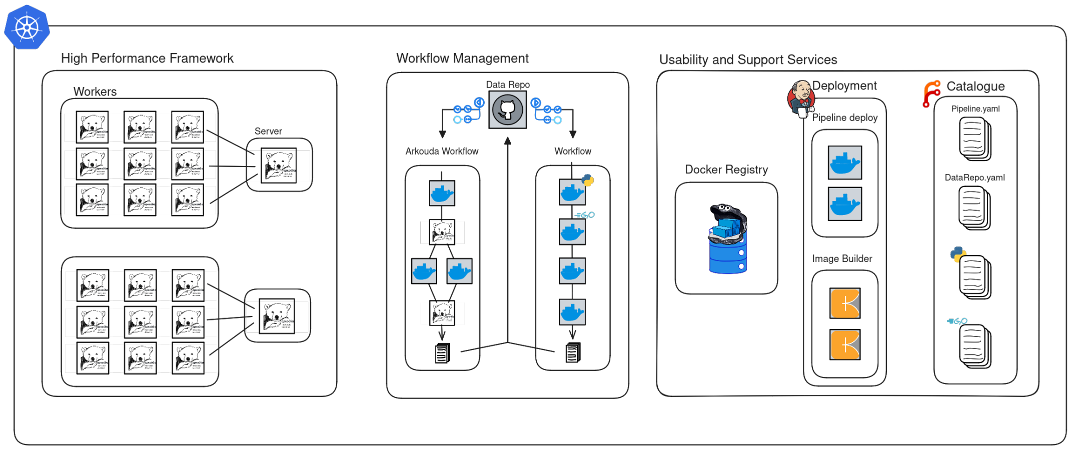

Abb. 6: Pachykouda high level infrastructure diagramm

4.3 Selection of Workflow Management Tools

As described in section 4.1, the first iteration of this project was to determine which, if any, of the presented tools were suitable for the task at hand. The following section will describe the process of selecting the tools and the criteria that were used to evaluate them. Because the time frame does not allow for a full integration and testing of all the presented tools in depth we will be using a decision making framework to evaluate the tools, as described in the Methodologies 2.2 to determine which tools will be most suitable for an initial PoC and will serve as a good starting point for the project and future iterations.

- **Pachyderm:** A k8s based Workflow manager, written in go which was recently aquired by Hewlett Packard Enterprise (HPE).
- **Argo:** A k8s based Workflow manager , written in go, which is a Cloud Native Computing Foundation (CNCF) project[22].
- **Cloud Application Services Platform (CLASP):** An in-house developed workflow manager, written in Java, utilizing Serverlet to execute workflows[23].
- **Snaplogic:** A commercial low-code/no-code workflow manager with a focus on data integration and data engineering[24].

[22] *Argoproj/Argo-Workflows* 2023
[23] Sayers, C. et al. 2015
[24] *iPaaS Solution for the Enterprise* 2023

But given that it was possible to select projects outside of the initial selection, the following projects also need to be considered:

- **Airflow:** A Python-based workflow manager under the CNCF umbrella, known for its easy-to-use interface and extensibility[25].
- **Kubeflow:** A k8s-native platform for deploying, monitoring, and running ML workflows and experiments, also a CNCF project, streamlining ML operations alongside other Kubernetes resources[26].
- **Knative:** An open-source k8s-based platform to build, deploy, and manage modern serverless workloads, simplifying the process of building cloud-native applications[27].
- **Luigi:** An open-source Python module created by Spotify to build complex pipelines of batch jobs, handling dependency resolution, workflow management, and visualization seamlessly[28].
- **Common Workflow Language (CWL):** An open-standard for describing analysis workflows and tools in a way that makes them portable and scalable across a variety of software and hardware environments, from workstations to cluster, cloud, and high-performance computing environments.

Selection Criteria

Due to this extensive list of diverse tools, a set of criteria was established to determine which tool would be the most suitable for the task at hand. The following list of criteria was established to evaluate the tools:

- **Ease of use:** As the inted endusers of the tool are not primarily HPC experts, the tool needs to be easy to use and understand, and should not require the enduser to have a deep understanding of the underlying infrastructure. While we can expect that the administration of the infrastructure will be done by adequately trained personnel, the enduser should be spared having to adapt to the underlying infrastructure as much as possible.
- **Extensibility:** One significant constraint of the project is the restricted number of available work-hours. Given that the project's environment predominantly centers around HPC (High Performance Computing) workloads, it's essential for the tool to be easily expandable without requiring extensive modifications to the underlying system. Ideally this property would be transferred to the enduser, allowing them to easily extend the developed tool further to their needs.

[25] Haines 2022
[26] *Kubeflow* 2023
[27] *Home - Knative* 2023
[28] *Spotify/Luigi* 2023

- **Community, Support and Documentation:** It is not enough that the software technically permits extensibility, the software also needs to be adequately documented and a support framework needs to be in place. Be it a community of users or a dedicated support team, the enduser and the developers need to be able to rely on the software being maintained and updated as well as being able to find expert help in case of problems.
- **Maturity:** With the boom of AI and ML in recent years[29], the number of tools and frameworks has exploded, and while this is a good thing it also means that a lot of these tools are still paving their way and are developing rapidly. While this is not necessarily a bad thing, it does mean that the tool might not be ready for production use and might not be able to provide the stability and reliability that is required for a production environment or are lacking in documentation and support.
- **Strategic alignment with HPE:** As this project is being developed within the context of HPE, it is important to consider the strategic alignment of the tool with HPE. HPE has is a large company with a diverse portfolio of products and services, and this project intersects with many different parts of the company. Therefore it is important to consider the strategic alignment of the tool with HPE and its products and services.
- **License:** While this PoC is not a commercial product in itself but rather an exploration of the problem space and a demonstration of what a final commercial product might be like, it is important to consider the licenses of the tools that are being used. Having to strip out a tool later on because of licensing issues would be a significant setback and therefore needs to be considered.
- **Cost:** Time is not the only constraint of this project, as the project is being developed within the context of HPE it is important to consider the cost of the tools that are being used.

Weigting of the Criteria

An integral part of the SMART methodology is the weighting of the criteria, as described in section 2.2. In order to rank the criteria themselves, as they are quite hard to quantify, We will be using the weighing methodology as described in the SMART-ER methodology 2.2.3.

The first step of which is the ranking of the criteria from most important to least important.

1. **Extensibility** As this is first and foremost a prototyping project, the actual development it at least for the first couple steps of the highest importance.
2. **Community, Support & Docs** This also applies for the external support available to the development team as if they are stuck, no developed can proceed, no matter the other factors.

[29] *24 Top AI Statistics & Trends In 2023 – Forbes Advisor* 2023

3. **License** This criterion has to weighted carefully, as a highly restrictive license might be a dealbreaker, but a license that is too permissive might conflict with the strategic alignment with HPE.

4. **Strategic alignment with HPE** As this is developed by and for HPE their requirements need to be consider aswell.

5. **Ease of Use** While the ease of use is important as this should eventually become a product, for now the central aspect is to create a PoC therefore the usability is a priority, but not the highest.

6. **Cost** As this is a PoC and not a commercial product, the cost is not the highest priority as this will be of small scale and therefore the cost will be negligible in most cases.

7. **Maturity** While the maturity of the tool is important, as this is a PoC and not a commercial product, if the maturity of the tool does not impact the extensibility of the tool or the development process, it is not the highest priority.

As all these criteria are quite important, the weighting function selected for the criteria is the RS function, as described in section 2.2.3, as it does not rank the criteria too harshly. The lookup tables for the weighting function can be found in the appendix 14.

Criteria	**Weight**
Extensibility	0.2500
Community, Support and Documentation	0.2143
License	0.1786
Strategic alignment with HPE	0.1429
Ease of use	0.1071
Maturity	0.0714
Cost	0.0357

Tab. 1: Weighting of the criteria

Evaluation of the Tools

Now that we have established the criteria aswell as their weighing, we can beginn to evaluate the tools based on the criteria. Here we will be using a mix of Methodologies, as some of these criteria can simply be indexed via analogous values, while others are of a more non specific nature. The discussion of which values will be used on which weighing scale for the tools comparison can be found in the apendix under

The following table shows the evaluation of the tools which where chosen for their relevance to the problem space, based on the criteria and the weighting of the criteria:

Criteria	Pachyderm	Argo	CLASP	Snaplogic
Ease of use	TBD	TBD	TBD	TBD
Extensibility	TBD	TBD	TBD	TBD
Community, Support & Docs	10	2.32	2.5	5.03
Maturity	TBD	TBD	TBD	TBD
Strategic alignment	TBD	TBD	TBD	TBD
License	10	7.5	10	0
Cost	TBD	TBD	TBD	TBD

Tab. 2: Evaluation of the suggested tools

Criteria	Airflow	Kubeflow	Knative	Luigi	CWL
Ease of use	TBD	TBD	TBD	TBD	TBD
Extensibility	TBD	TBD	TBD	TBD	TBD
Community, Support & Docs	10	2.25	0.74	2.29	0.22
Maturity	TBD	TBD	TBD	TBD	TBD
Strategic alignment	TBD	TBD	TBD	TBD	TBD
License	7.5	7.5	7.5	7.5	7.5
Cost	TBD	TBD	TBD	TBD	TBD

Tab. 3: Evaluation of the additional tools

Conculsion of the Selection Process

TODO: Write conclusion of the selection process

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

4.4 Implementation of the Artifact

This section will describe the iterative process of implementing the larger artifact and is broken up into 3 subsections. While these steps where happening concurrently, they each address a different aspect of the project and therefore mostly underwent their own iterative processes.

4.4.1 Infrastructure

First iteration - Minikube

As the decision of the Workflow management tool was made, it was obvious that a dedicated k8s infrastructure was needed to run the tool[30]. The Pachyderm documentation gave two recommendations for setting up an initial development environment, preferably Docker Desktop or alternatively Minikube[31]. Due to the exclusive license of Docker-Desktop[32], which prevents large companies free usage of the product[33] the choice fell on Minikube for an initial test setup.

In addition to the underlying k8s Pachyderm also needs an external S3 Storage Bucket for its Pachyderm File System (PFS) for which we used MinIO, a self-hostable S3 compliant object storage[34], which was also based on recommendations by the Pachyderm documentation.

The persistent storage requirements for the Pachyderm itself was fulfilled by manually creating two Persistent Volume (PV)'s on the hosts local hard drive. Using the Helm packagemanager[35] for k8s the at that point the newest version 2.6.4 was installed from the official Artifacthub repository[36].

The host system of this iteration was a single ProLiant DL385 Gen10 Plus running Ubuntu 22.04.3 LTS x86_64. During the setup every step was diligently noted and put into a repository[37], alongside the needed scripts. The instructions can be found in the appendix at 3.

Learnings from the first iteration

The shortcomings of this naive first iteration became apparent very quickly, which was to be expected, as the goal of this iteration was to create a minimal working example to get a better understanding of the tooling and the underlying infrastructure.

The first and foremost issue where the limitations imposed by Minikubes' reliance on an Internal Virtual Machine (VM), during testing the inability to on the fly increase the resources of the VM became a significant bottleneck. At some point during the testing of 4.4.2 the VM was so overloaded that the installation was irreparably damaged which was seen as a sign to move on to the next iteration.

Another more subtle issue was the discrepancy between the experience a small scale k8s installation within Minikube and a large scale k8s cluster like the one that would be used in later steps

[30] *Pachyderm Docs - On-Prem Deploy* 2023
[31] *Pachyderm Docs - Local Deploy* 2023
[32] *Docker Terms of Service | Docker* 2022
[33] *Docker FAQs | Docker* 2021
[34] Inc 2023
[35] *Helm Docs Home* 2023
[36] *Artifacthub Pachyderm 2.6.4* 2023
[37] Eckerth 2023

of the project. Therefore, it was decided that a more realistic k8s cluster would be needed for the next iteration, which became the Heydar cluster.

Second iteration - Heydar Cluster

Improving upon the shortcomings of the first iteration, the second iteration was based in the attempt to create a more realistic k8s cluster. To achieve this 20 ProLiant DL360 Gen9 Servers, running Ubuntu 22.04.3 LTS x86_64 where used to create a bare metal k8s cluster, using kubeadm as it provides deep integration with the underlying infrastructure[38].

But a bare metal cluster also comes with its own set of challenges, as the cluster needs to be provisioned and configured manually. In order to automate this process, the Ansible automation tool was used to set up all the nodes in parallel and to ensure that the all the nodes are in the same state. Ansible is a declarative tool which allows for the automation of the provisioning and configuration of the cluster[39], by specifying the desired state of the cluster in a playbook and then applying it to the cluster. The Ansible playbook used for the setup of the cluster can be found at Appendix 4/1.

Which unknowingly caused conflict between the Ansible playbook and the maintenance scripts of the cluster as the Heydar machines. As k8s needs very specific configurations on the underlying infrastructure like the deactivation of swap space[40].

This was resolved by consulting with the maintainer of the cluster and adjusting the Ansible playbook as well as the maintenance config for the cluster nodes accordingly, after we had identified the issue.

One important aspect of a production like cluster is the networking, as k8s does not natively manage communication on a cluster level, but instead relies on so called Container Network Interface (CNI)s to manage and abstract the underlying network infrastructure[41].

Here we are spoiled for choice once again, as there are a multitude of different CNIs available, each with their own advantages and disadvantages. The Kubernetes documentation provides a non-exhaustive list of 17 different CNIs[42], which all fulfill this essential task in different ways. As the needs regarding the network plugin where not very specific at this point, the choice fell on Calico, as surface level research showed that it was a popular choice for bare metal clusters[43], provided security and enterprise support as well having a wide range of features[44]. But Calico proved to be more difficult to set up than expected, after consulting with a college who set up a different cluster with Calico, it was decided to use Flannel as a CNI instead. Flannel turned out

[38] *Creating a Cluster with Kubeadm* 2023
[39] *Ansible* 2023
[40] *Installing Kubeadm* 2023
[41] *Cluster Networking* 2023
[42] *Kubernetes CNI Plugins* 2023
[43] *Explore Network Plugins for Kubernetes* 2023
[44] Mehndiratta 2023

to be much easier to set up and configure, as it is a very lightweight CNI which is designed for bare metal clusters[45], and foregoes the more advanced security features of Calico.

The Flannel configuration used for the cluster can be found at Appendix 4/2 it is closely based on the example configuration provided by the Flannel documentation[46].

Learnings from the second iteration

The second iteration was a significant improvement over the first iteration, as it provided a much more realistic environment for the development of the artifact. But it also came with its own set of challenges, as the bare metal cluster needed to be provisioned and configured manually, which was a significant time investment.

What became apparent very quickly was that the solution for the provisioning of the PV was nowhere near scalable, as it relies on the local hard drive of the host machine and therefore must host the container on the same machine as the PV which defeats the purpose of a multi node cluster in the first place. Therefore, a more scalable solution needs to be implemented for the next iteration. A possible solution could be the use of distributed storage solutions like Ceph[47] or GlusterFS[48] in combination with the Rook project[49]. Which will need to be explored in future iterations.

As described in section 4.4.2 a service hosting Fabric Attached Memory (FAM) will be needed in future iterations as well.

[45] *Flannel* 2023
[46] *Flannel Install Config* 2023
[47] *Ceph.Io — Home* 2023
[48] *Gluster* 2023
[49] *Rook* 2023

4.4.2 Tightly Coupled HPC Workloads

As described in section 3.2.2 TCP problems are a large part of the HPC world, but seem to lack native support in Pachyderm. Pachyderm as it exists as of writing this thesis, is centralized around LCP problems, as it is designed to work with large amounts of data but with each so called "datum" being independent of each other. This is a very good fit for LCP problems, and ties into their concepts of data lineage, versioning and providence.

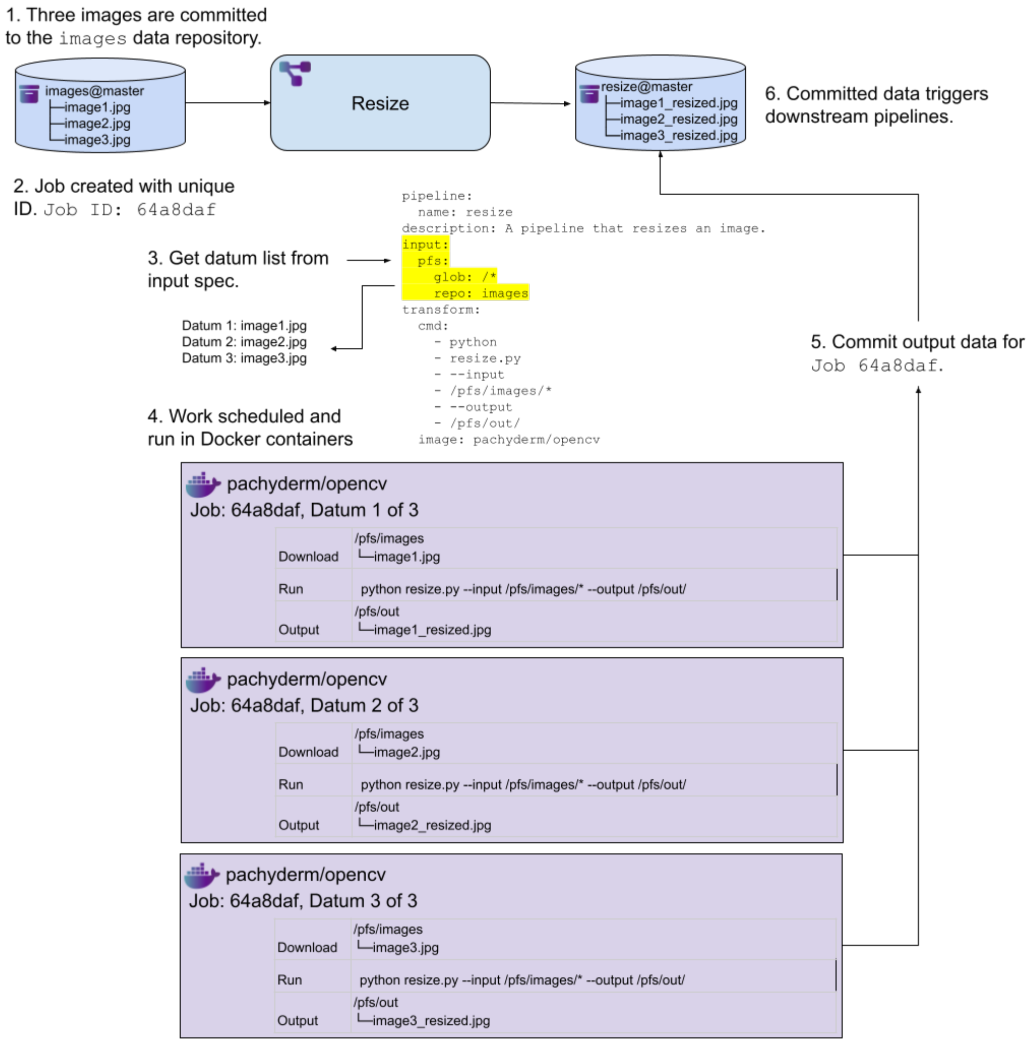

Abb. 7: Pachykouda datum distribution amongst workers [50]

Diagram 7 shows Pachyderms approach to distribute their datums amongst workers, given an already defined pipeline. Once Data files are added to the input repository, Pachyderm will determine Based on a glob pattern wether the files are relevant datums for the pipeline. If the newly added data fits the pattern each of the files will be supplied to its own instantiation of a worker, all originating from the same image, which will then process the data concurrently and

[50] Taken from: *Intro to Pipelines* 2023

independently of each other. After the worker has finished its task, the resulting datums are then collected in their own repository of data. A more detailed swim lane diagram of this process can be found in the appendix at 14

This approach is very well suited for LCP problems, as the datums are independent of each other and can be processed in parallel without any issues. But it is not well suited for Large TCP problems, if the computation of the data can not be split into distinct independent datum files, or the computation is reliant on the intercommunication of the datums. If the datasets are small enough, this does not really present a problem as one can simply take all the data into a single workernode and process it there. But as a single worker node can only utilize the resources of a single physical compute node, this does not scale well with the size of the dataset and defeats the purpose of a distributed system in the first place.

So our goal for this section is a way to find a way to enable pachyderm to pool the entire resources of the cluster, in oder to solve a TCP problem.

First iteration - PachyKouda

As a first attempt to address this issue, it was decided that the integration of a TCP framework into Pachyderm on the container level would be the best approach. So the first iteration is based on the idea of a Pachyderm conforming client container, which is able to interface with an external TCP framework, which can handle the reception of the data, the distribution of the data amongst the workers and the collection of the results to reintegrate them into the PFS.

The first iteration of this idea was called PachyKouda, as it was based on the Arkouda TCP framework[51], which itself is a python binding for the Chapel programming language[52].

For that step an Arkouda worker was installed bare metal on the headnode of the Heydar cluster, in order to verify the feasibility of the idea, with the goal of moving the worker into the cluster in the next iteration.

The client container was based on the official User Datagram Protocol (UDP)-based build by the Arkouda team[53]. The container was then modified to be able to communicate with the Arkouda worker on the headnode of the cluster, it can now send data to the worker and receive the results.

Learnings from the first iteration

The first iteration was a total success, as it proved the feasibility of being able to use a client container to forward the data processing to an external Arkouda worker. As described earlier,

[51] *Arkouda Gituhb Repository* 2023
[52] *Chapel-Lang* 2023
[53] *Arkouda-Contrib/Arkouda-Docker at Main · Bears-R-Us/Arkouda-Contrib* 2023

the goal of the next iteration is to move the Arkouda worker into the cluster, in order to be able to utilize the full resources of the cluster.

Second iteration - Kymera

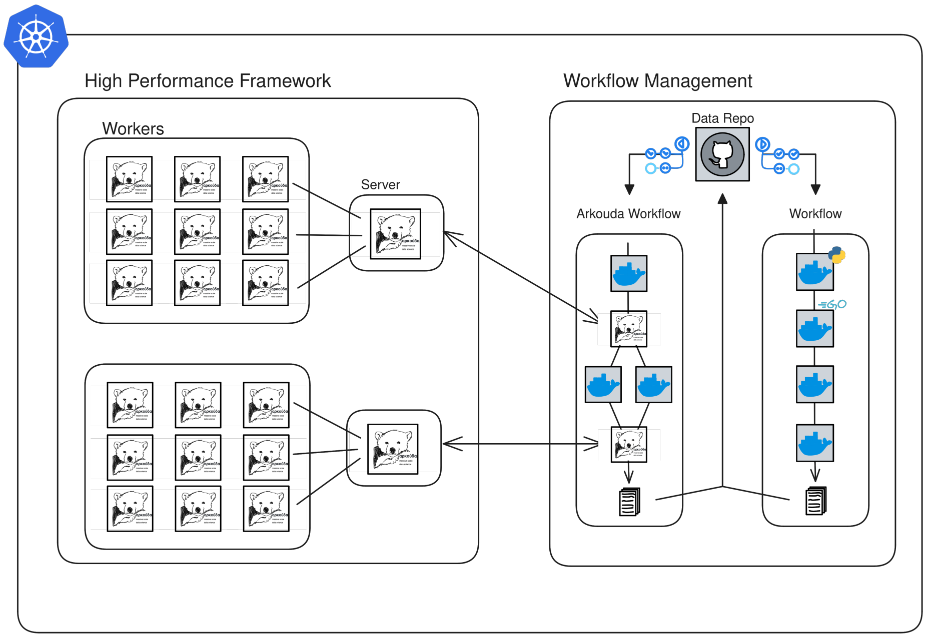

Abb. 8: Arkouda workers on the Heydar cluster

Diagram 8 above shows a high level overview of how the workers interface with the client container in the workflow. The Arkouda container which is part of the workflow is still the same as in the first iteration, but now instead of interfacing with an external worker it is interfacing with a worker swarm hosted across the cluster.

The Swarm is split into two parts, one central Arkouda server, facilitating the communication between the client container and the workers and the workers also called locales themselves. The locales and the server are based on the helm charts provided by the Arkouda-Contrip repo[54],

A detailed walk through the setup of the Role Based Access Control (RBAC), Secrets and deployments for the Heydar Cluster can be found in the appendix at Appendix 4/4 which in turn is based on the official Arkouda documentation[55].

[54] *Bears-R-Us/Arkouda-Contrib/Arkouda-Helm-Charts* 2023
[55] *Arkouda-Contrib/Arkouda-Docker at Main · Bears-R-Us/Arkouda-Contrib* 2023

Learnings from the second iteration

As Arkouda does not currently provide multi tenancy of their Server, meaning that they can only be connected a single client at a time, so if multiple pipelines need to solve a TCP at the same time, they would not be able to share the same worker swarm. Instead they would need to spawn their own worker swarm.

Another issue is that there are currently going trough the standard Pod to pod communication configuration of flannel, which means that the entire traffic between the client container and the Arkouda server as well as the traffic between the workers is all happening over emulated overlay network which enables the containers on the different nodes to communicate with each other as if they where on the same network, no matter of the actuall infrastructure below it. The communication protocol of the Arkouda servers is UDP based Global Address Space Networking (GASNet), which provides the Remote Direct Memory Access (RDMA) needed for the Arkouda framework to work, but this incurs a significant overhead in the form of the encapsulation of the UDP packets into TCP packets.

Also the containers are currently not compatible with the OpenFAM project[56], which is being developed as an integration to Arkouda and Chapel by the Hewlett Packard Systems Architecture Lab[57], it extends the Arkouda framework with the ability to use FAM as banks of RDMA enabled memory, which can be accessed by the Arkouda workers. This would proof to be a significant improvement as it has the potential to reduce the overall overhead of the communication[58] amongst the workers as well as to the server, by cutting down the overall amount of network traffic.

The pachyderm platform itself might also benefit from the integration of FAM, as it could be used to store the datums in the PFS, providing the running pipeline processes with a much faster access to the data.

Third iteration - FAM

While significant efforts have already been made to successfully integrate Arkouda and FAM, these have so far been focussing on bare metal installations, for that reason, in order to integrate the FAM enabled Arkouda working from within a containerized environment the tools would need to be customly recompiled matching the new environment. Therefore we needed to:

1. Compile OpenFAM in the Container
2. Compile custom Chapel in the Container with OpenFAM
3. Compile custom Arkouda in the Container with the OpenFAM enabled Chapel

[56] Keeton/Singhal/Raymond 2019
[57] Byrne et al. 2023
[58] Chou et al. 2019

4. Rebuild the Arkouda container with the new Arkouda binary

5. Reweite the k8s deployment to make use of OpenFAM

This section was quite challenging as it required a deep understanding of the PoC implementations of the OpenFAM, Arkouda and Chapel projects and was cut short by the time constraints of the project and was therefore not brought to a successful conclusion. The current state of the project can be found in the PoC repository[59] and in the appendix at ??. But this showed us that there is a lot of potential in the integration of FAM into the container based HPC world, as it could provide a significant performance boost to the overall system and should be explored further in future iterations.

4.4.1 Supplementary Services

As the other branches of the prototyping where happening, the need for additional services and infrastructure arose to support the development of the prototype as well as to increase the general usability of the prototype. This section will especially describe the services which help to make this prototype a more complete solution.

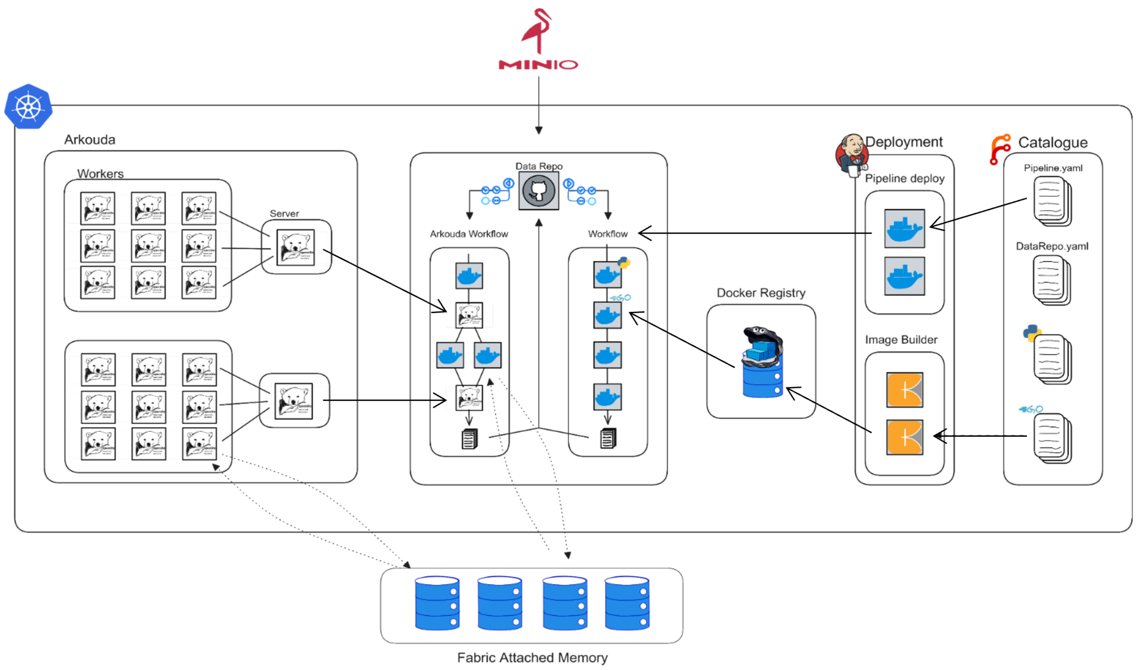

Abb. 9: Pachyderm High-Level Architecture

[59] **eckerthPoCRepository2023**

Docker Registry

One thing that was quite apparent from the getgo, was the need for a central docker registry. As Pachyderm does not manage the docker images itself, but relies on the user to provide them somehow externally.

During the first iterations when the development was being done on Minikube as described in 4.4.1, the internal Registry of the node was enough. But as soon as we moved over to the Heydar system keeping the Hosts internal registries in sync was of course not feasible, Therefore we added a private docker registry to the cluster[60]. The deployment config is based on the official docker registry helm chart[61] and can be found at Appendix 4/5.

Frogejo Catalogue

One if the main goals of this project was to create a tool that is accessible to the end user, and therefore the need for a catalog of available workflows arose. The idea was to create something similar to the Clasp catalogue[62] but for the Pachyderm ecosystem. Meaning that users can share, search and deploy workflows from a central catalogue, without having to worry about the underlying infrastructure.

Having HPC software in a completely contained and version system directly addresses many of the originial problem statements, described in **??**

Jenkins CI/CD Pipeline

4.5 Evaluation of the Artifact

[60] Kumar 2020
[61] *Docker-Registry 1.10.0 · Phntom/Phntom* 2023
[62] Sayers, Craig et al. 2015

5 Conclusion

6 Summary and Outlook

6.1 Summary

6.2 Outlook

Appendix

Appendix Index

Appendix 1: Discussion of Tool Evaluation and Weighing

Appendix 1/1: Extensibility

Appendix 1/2: Community, Support & Docs

This section assesses the level of external support provided for each project. To evaluate this support, we will focus on three distinct aspects and combine them into a single score. Firstly, we will examine the size of the community, as a substantial community often indicates project maturity and the availability of extensive support. As proxies for community size, we will consider two central metrics: the number of stars on GitHub and the quantity of questions on Stack Overflow.

Tab. 4: Comparison of Project Popularity

Project	GitHub Stars	Stack Overflow Questions
Pachyderm	6,000	6
Argo	14,500	136
Clasp	0	0
Snaplogic	0	57
Airflow	32,200	10,218
Kubeflow	13,100	434
Knative	4,100	204
Luigi	16,900	346
CWL	1,400	6

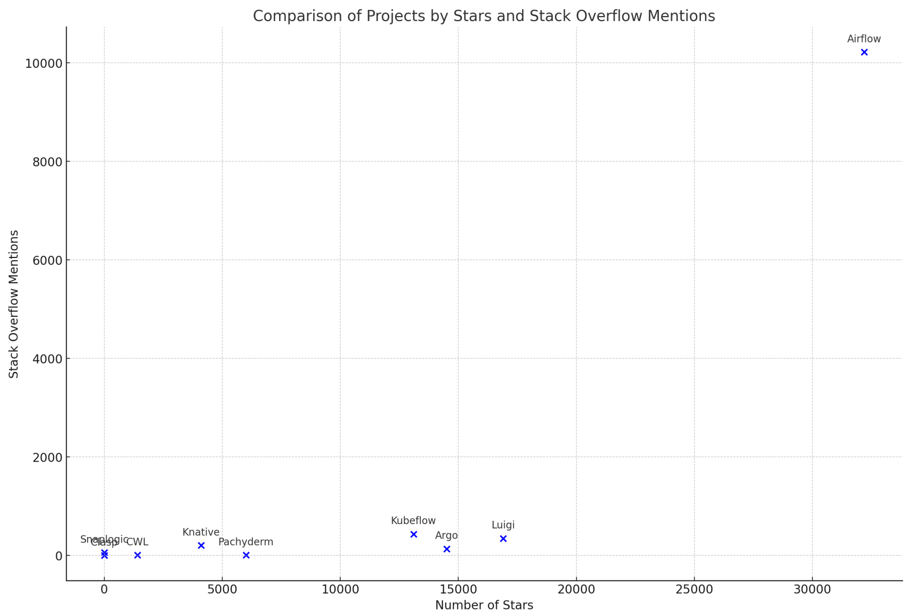

Abb. 10: Stars and Stack Overflow Questions Comparison

To gauge the level of support and community engagement surrounding these projects, we have devised a composite score that normalizes and combines the GitHub stars and Stack Overflow questions metrics. The calculation of this score involves the following methodology:

Each project is represented as a point $P_i = (x_i, y_i)$ in a two-dimensional space, with x_i and y_i being the number of GitHub stars and Stack Overflow questions, respectively, for the i-th project. The composite score S_i for each project is computed by normalizing these values to a scale of 0-10 and then taking their average.

Additionally, we acknowledge that some commercial tools, as well as certain open-source projects, offer enterprise support, reducing the reliance on the community for assistance. Similarly, projects developed in-house often have access to the original development team for support. Therefore, we will apply a flat bonus of 5 points to the scores of projects offering enterprise support and a flat bonus of 2.5 points to projects developed in-house.

$$S_i = \frac{1}{2}\left(\frac{x_i - \min(x)}{\max(x) - \min(x)} \times 10 + \frac{y_i - \min(y)}{\max(y) - \min(y)} \times 10\right) + B_i$$

Here, $\min(x)$, $\max(x)$, $\min(y)$, and $\max(y)$ represent the minimum and maximum values of GitHub stars and Stack Overflow questions across all projects, respectively. The final scores S_i, along with the respective bonuses B_i, provide a comprehensive metric for comparing project popularity, community engagement, and the availability of additional support options, all on the same scale.

Project	**Composite Score**	**Enterprise Bonus**	**Inhouse Bonus**	**Final Score**
Airflow	10.00	0	0	10.00
Pachyderm	0.93	5	2.5	8.43
Snaplogic	0.03	5	0	5.03
Luigi	2.79	0	0	2.79
Clasp	0.00	0	2.5	2.5
Argo	2.32	0	0	2.32
Kubeflow	2.25	0	0	2.25
Knative	0.74	0	0	0.74
CWL	0.22	0	0	0.22

Tab. 5: Composite scores of Workflow managers, sorted by final score

Appendix 1/3: License

As discussed in section 4.3 the tools in consideration should not be to restrictive. To evaluate the criteria we will employ a 4 bucket system:

- **Ideal Situation (Score: 10):** This refers to cases where either the tool is in the public domain (and therefore not subject to copyright restrictions) or where our organization possesses a direct ownership or significant influence over the licensing terms. This situation provides the most flexibility, allowing for extensive modification, redistribution, and proprietary use without concern for licensing infringements.

- **Permissive License (Score: 7.5):** Tools under licenses like MIT, BSD, or Apache 2.0 fall into this category. These licenses are highly permissive and generally allow for broad freedom, including modification, distribution, and private use, with minimal restrictions, often limited to liability and warranty.
- **Restrictive or Reciprocal Licenses (Score: 2.5):** Licenses such as the GPL or AGPL are more restrictive, requiring any changes to be open-sourced or contributions to be made back to the community. These "copyleft" licenses can be problematic in proprietary settings where modifications or integrations need to remain confidential.
- **Unacceptable Licenses (Score: 0):** This includes licenses that impose burdensome conditions or high costs, proprietary software where the source code is unavailable, or situations where the licensing terms make it impractical to use within our projects. For instance, licenses that mandate the purchase of additional software, restrict certain types of use, or pose potential legal risks would fall into this category.

Now we will evaluate the licenses of the tools in question, and assign them a score based on the above criteria.

- **Pachyderm** The licensing model of Pachyderm follows a model which has similarities with the "Open Core model"[63]. Which means that while the core functionalities are published as the "COMMUNITY EDITION" with a permissive source-available License (Apache License 2.0)[64]. Functionality like Singele Sign On (SSO) or the ability to create more than 16 pipelines are part of a different distribution under a Commercial License.

  But in our case this is of no concern, as the startup behind the Pachyderm softwarem, including its Interlectual Property (IP) was aquired by HPE. Giving us a free hand to modify without needing to worry.

- **Argo**Argo's adoption of the Apache License 2.0[65] aligns with common practices for open-source projects, affording users considerable freedom. This permissive license simplifies the use, modification, and redistribution of the software, an aspect that's particularly beneficial for collaborative development or integration into proprietary software. Given our requirements and operational context, this offers us the flexibility needed for adaptation and potential enhancements without stringent restrictions, streamlining any developmental efforts we undertake with Argo.

[63] *Pahcyderm -Pricing* 2022
[64] *Pachyderm/LICENSE at Master · Pachyderm/Pachyderm* 2023
[65] *Argo-Cd/LICENSE at Master · Argoproj/Argo-Cd* 2023

- **CLASP** is not a published software and therefore not under any specific license. But similar considerations as the ones of Pachyderm apply here aswell, as it is an internal project the IP also completely belongs to HPE
- **Snaplogic** is an entirely commercial product which does not provide insight into nor the right to modify their Software[66]. But as they might agree this is not a total knockout criterion for this entire project, but in regards to the licensing it will be weighted with 0.
- **Airflow** is licensed under the Apache License 2.0.[67]
- **Kubeflow** is licensed under the Apache License 2.0.[68]
- **Knative** is licensed under the Apache License 2.0.[69]
- **Luigi** is licensed under the Apache License 2.0.[70]
- **CWL** is licensed under the Apache License 2.0.[71]

Appendix 1/4: Strategic alignment

Appendix 1/5: Ease of Use

Appendix 1/6: Maturity

Appendix 1/7: Cost

This section aims to compare the relative cost of the products in relation to each other. We previously factored in the enterprise features, so when enterprise support is available and applicable we will take this into consideration. Here we have three categories of products first those which are completely free and without any enterprise support, secondly those which are free but offer enterprise support and lastly those which operate on a subscription basis.

[66] *SnapLogic – Master Subscription Agreement* 2023
[67] *License — Airflow Documentation* 2023
[68] *Kubeflow/LICENSE at Master · Kubeflow/Kubeflow* 2023
[69] *Knative Docs/LICENSE at Main · Knative/Docs* 2023
[70] *Luigi/LICENSE at Master · Spotify/Luigi* 2023
[71] *Cwl-Utils/LICENSE at Main · Common-Workflow-Language/Cwl-Utils* 2023

Appendix 2: Diagrams

Appendix 2/1: Lookup table weighing functions

Appendix 2/2: Pipeline Communication Swim Lane Diagram

	Attributes								
Rank	2	3	4	5	6	7	8	9	10
1	0.7500	0.6111	0.5208	0.4567	0.4083	0.3704	0.3397	0.3143	0.2929
2	0.2500	0.2778	0.2708	0.2567	0.2417	0.2276	0.2147	0.2032	0.1929
3		0.1111	0.1458	0.1567	0.1583	0.1561	0.1522	0.1477	0.1429
4			0.0625	0.0900	0.1028	0.1085	0.1106	0.1106	0.1096
5				0.0400	0.0611	0.0728	0.0793	0.0828	0.0846
6					0.0278	0.0442	0.0543	0.0606	0.0646
7						0.0204	0.0334	0.0421	0.0479
8							0.0156	0.0262	0.0336
9								0.0123	0.0211
10									0.0100

Abb. 11: ROC weights [72]

	Attributes								
Rank	2	3	4	5	6	7	8	9	10
1	0.6667	0.5455	0.4800	0.4379	0.4082	0.3857	0.3679	0.3535	0.3414
2	0.3333	0.2727	0.2400	0.2190	0.2041	0.1928	0.1840	0.1767	0.1707
3		0.1818	0.1600	0.1460	0.1361	0.1286	0.1226	0.1178	0.1138
4			0.1200	0.1095	0.1020	0.0964	0.0920	0.0884	0.0854
5				0.0876	0.0816	0.0771	0.0736	0.0707	0.0682
6					0.0680	0.0643	0.0613	0.0589	0.0569
7						0.0551	0.0525	0.0505	0.0488
8							0.0460	0.0442	0.0427
9								0.0393	0.0379
10									0.0341

Abb. 12: RR weights [73]

Appendix 2/3: Pipeline Communication Swim Lane Diagram

[72] Taken from: Roberts/Goodwin 2002
[73] Taken from: Roberts/Goodwin 2002
[74] Taken from: Roberts/Goodwin 2002
[75] Taken from: *Intro to Pipelines* 2023

	Attributes								
Rank	2	3	4	5	6	7	8	9	10
1	0.6667	0.5000	0.4000	0.3333	0.2857	0.2500	0.2222	0.2000	0.1818
2	0.3333	0.3333	0.3000	0.2667	0.2381	0.2143	0.1944	0.1778	0.1636
3		0.1667	0.2000	0.2000	0.1905	0.1786	0.1667	0.1556	0.1455
4			0.1000	0.1333	0.1429	0.1429	0.1389	0.1333	0.1273
5				0.0667	0.0952	0.1071	0.1111	0.1111	0.1091
6					0.0476	0.0714	0.0833	0.0889	0.0909
7						0.0357	0.0556	0.0667	0.0727
8							0.0278	0.0444	0.0545
9								0.0222	0.0364
10									0.0182

Abb. 13: RS weights [74]

Appendix 3: Minikube installation instructions

```
# Pachyderm

## Installation

These instructions are based upon the excellent guide by
    ↪ [Pachyderm](https://docs.pachyderm.com/latest/set-up/on-prem/)

### Proxy

If you are in the HPE internal network, you will need to set up the proxy.
Simply execute the following command:

```bash
export HTTP_PROXY=http://web-proxy.corp.hpecorp.net:8080
export HTTPS_PROXY=http://web-proxy.corp.hpecorp.net:8080
```

If you want to make this permanent, add these lines to the `~/.bashrc` or
    ↪ equivalent file.

kubectl

Simply following the instructions on the [kubernetes
    ↪ website](https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/)
    ↪ should be sufficient.
But for the sake of completeness, here is what I did:

```bash
curl -LO "https://dl.k8s.io/release/$(curl -L -s
    ↪ https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

If the proxy is giving you grief one can simply download the binary elsewhere
    ↪ and copy it to the target machine. (not recommended)
```

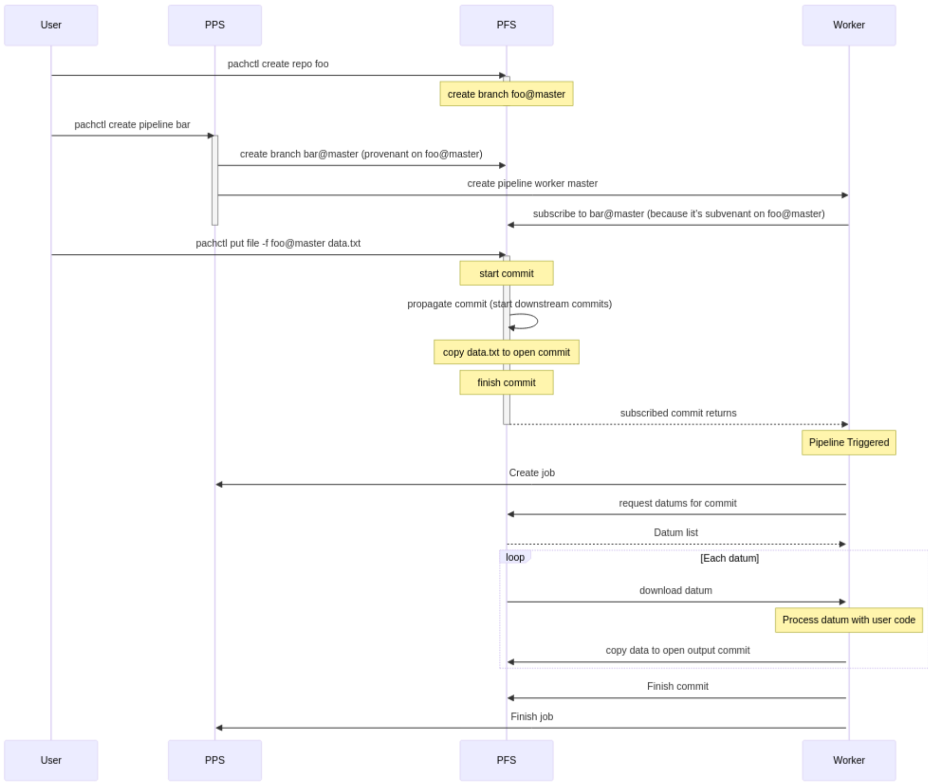

Abb. 14: Swim lane Diagram of the communication between the user and Pachyderm[75]

~~~

Installing minikube

The same things apply for minikube as for kubectl.
The propper instructions can be found on the [minikube website](https://minikube.sigs.k8s.io/docs/start/)
But here is what I did anyway:

```bash
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64.deb
sudo dpkg -i minikube_latest_amd64.deb
```

We can then test the installation by running:

```bash
minikube start
kubectl cluster-info
```

If you are getting an error stating that it is not able to connect to the cluster you might need to set the following environment variable:

```bash
~~~

````
export NO_PROXY=localhost,127.0.0.1,10.96.0.0/12,192.168.59.0/24,192.168.49.0/24,192.168.
```

### Installing [helm](https://helm.sh/docs/intro/install/)

Same procedure as every year...

```bash
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null
sudo apt-get install apt-transport-https --yes
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

[Persistent Storage](https://kubernetes.io/docs/tasks/configure-pod-container/configure-persis

We need to create a persistent volume for etcd and the postgres database.
Therefore we need to create a directory for each of them.

```bash
mkdir -p /mnt/pachyderm/etcd
mkdir -p /mnt/pachyderm/postgres
```

We then create the configuration files for the persistent volumes.

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: etcd-pv
labels:
    type: local
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: manual
  local:
    path: /mnt/pachyderm/etcd

---

````

````
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-pv
labels:
    type: local
spec:
    capacity:
        storage: 10Gi
    accessModes:
        - ReadWriteOnce
    storageClassName: manual
    local:
        path: /mnt/pachyderm/postgres
```

And then the corresponding persistent volume claims.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: etcd-pvc
spec:
    storageClassName: manual
    accessModes:
        - ReadWriteOnce
    resources:
        requests:
        storage: 10Gi

---

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc
spec:
    storageClassName: manual
    accessModes:
        - ReadWriteOnce
    resources:
        requests:
        storage: 10Gi
```

Then we add the storage class to the cluster.

```bash
kubectl apply -f filename.yaml
```
````
````

````
We then take note of the storage class name because we will add it to the helm values file later. \
In this case it is `manual`.

Installing [MinIO](https://min.io/docs/minio/linux/index.html)

We now install an S3 compatible storage system. Which one does not really matter, but I chose MinIO because it is easy to install and configure.

```bash
wget https://dl.min.io/server/minio/release/linux-amd64/archive/minio_20230619195250.0.
-O minio.deb
sudo dpkg -i minio.deb

mkdir -p /mnt/pachyderm/minio

# to manually start the server
minio server /mnt/pachyderm/minio --console-address :9001
```

The standard password is `minioadmin:minioadmin`

Then you can access the web interface at `http://localhost:9001` where you should login, change the password and create a bucket. \
The access credentials for the bucket will be added to the helm values file later, so take note of them.

Installing [Pachyderm](https://docs.pachyderm.com/latest/set-up/on-prem/)

First we need to add the Pachyderm helm repository:

```bash
helm repo add pachyderm https://helm.pachyderm.com
helm repo update
```

We then get the values file from the repository and edit it to our liking.\
My setup is based on the version `2.6.4-1`, so it might be different for future versions.

```bash
wget https://raw.githubusercontent.com/pachyderm/pachyderm/2.6.x/etc/helm/pachyderm/val
```

MinIO

First we change the deploy target at line `L7`

````

````
```yaml
# Deploy Target configures the storage backend to use and cloud provider
# settings (storage classes, etc). It must be one of GOOGLE, AMAZON,
# MINIO, MICROSOFT, CUSTOM or LOCAL.
deployTarget: "MINIO"
...
```

This does not need to be set when using something else but with MinIO we also have to set `L544` to "MINIO"

```yaml
...
storage:
    # backend configures the storage backend to use.  It must be one
    # of GOOGLE, AMAZON, MINIO, MICROSOFT or LOCAL. This is set automatically
    # if deployTarget is GOOGLE, AMAZON, MICROSOFT, or LOCAL
    backend: "MINIO"
    ...
```

A little further down (`L635`) we find the MinIO configuration. We need to set the endpoint, access key and secret key.

This point was a little tricky as I had MinIO installed on the same machine as Pachyderm, but it would take no other value than the outward facing IP address of the machine.

```yaml
...
    minio:
      # minio bucket name
      bucket: "<bucket name>"
      # the minio endpoint. Should only be the hostname:port, no http/https.
      endpoint: "10.X.X.X:9000"
      # the username/id with readwrite access to the bucket.
      id: "<id>"
      # the secret/password of the user with readwrite access to the bucket.
      secret: "<secret>"
      # enable https for minio with "true" defaults to "false"
      secure: "false"
      # Enable S3v2 support by setting signature to "1". This feature is being deprecated
      signature: ""
      ...
```

Storage classes

Now we add the storage classes we created earlier to the Postgres at `L784`

````

````
```yaml
...
    # AWS: https://docs.aws.amazon.com/eks/latest/userguide/storage-classes.html
    # GCP: https://cloud.google.com/compute/docs/disks/performance#disk_types
    # Azure: https://docs.microsoft.com/en-us/azure/aks/concepts-storage#storage-classes
    storageClass: manual
    # storageSize specifies the size of the volume to use for postgresql
    # Recommended Minimum Disk size for Microsoft/Azure: 256Gi  - 1,100 IOPS https://azure.microsoft.com/en-us/pricing/details/managed-disks/
...
```

and for the etcd at around `L144`

```yaml
...

  # GCP: https://cloud.google.com/compute/docs/disks/performance#disk_types
  # Azure: https://docs.microsoft.com/en-us/azure/aks/concepts-storage#storage-classes
  #storageClass: manual
  storageClassName: manual

  # storageSize specifies the size of the volume to use for etcd.
  # Recommended Minimum Disk size for Microsoft/Azure: 256Gi  - 1,100 IOPS https://azure.microsoft.com/en-us/pricing/details/managed-disks/

...
```

SSL Certificates

My setup refuses to work without SSL certificates, so I had to generate some.

```bash
openssl genrsa -out <CertName>.key 2048
openssl req -new -x509 -sha256 -key <CertName>.key -out <CertName>.crt

kubectl create secret tls <SecretName> --cert=<CertName>.crt --key=<CertName>.key
```

We then edit the `values.yaml` file at around `L683` to use the certificates.

```yaml
...
  tls:
    enabled: true
    secretName: "<SecretName>"
    newSecret:
````
````

````
      create: false
    ...
```

### CLI

To directly interact with the cluster we need to install the Pachyderm CLI.

```bash
curl -o /tmp/pachctl.deb -L https://github.com/pachyderm/pachyderm/releases/download/v2.6.5/pachctl_2.6.5_amd6 && sudo dpkg -i /tmp/pachctl.deb
```

Deploy

Now that the values file is ready we can install Pachyderm.

```bash
helm install pachyderm pachyderm/pachyderm \
  -f ./values.yml pachyderm/pachyderm \
  --set postgresql.volumePermissions.enabled=true \
  --set deployTarget=LOCAL \
  --set proxy.enabled=true \
  --set proxy.service.type=NodePort  \
  --set proxy.host=localhost \
  --set proxy.service.httpPort=8080

```

Now you might want to connect to the dashboard. This can be done by port-forwarding the service.

```bash
pachctl port-forward
```

:tada: Now we should be able to access the dashboard at `http://localhost:4000` :tada:
````
````

Appendix 4: Kubernetes setup scripts

Appendix 4/1: Ansible setup script

```
---
- hosts: heydar_nodes
  become: yes
  tasks:
    - name: Setting up environment variables
      lineinfile:
        path: /etc/environment
        line: "{{ item }}"
      with_items:
        - "https_proxy=http://proxy.its.hpecorp.net:80"
        - "HTTP_PROXY=http://proxy.its.hpecorp.net:80"
        - "http_proxy=http://proxy.its.hpecorp.net:80"
        - "NO_PROXY=localhost,127.0.0.1,10.0.0.0/8,172.16.0.0/16,10.93.246.68/28"

    - name: Update and install necessary packages
      apt:
        name: "{{ packages }}"
        update_cache: yes
      vars:
        packages:
          - apt-transport-https
          - ca-certificates
          - curl

    - name: Add Kubernetes apt-key
      shell: |
        curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | gpg --yes --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg
        echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | tee /etc/apt/sources.list.d/kubernetes.list
        apt-get update -y
        apt-get install -y kubelet kubeadm kubectl containerd
        apt-mark hold kubelet kubeadm kubectl

    - name: Enable necessary kernel modules and sysctl parameters
      shell: |
        modprobe br_netfilter
        echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
        echo '1' > /proc/sys/net/ipv4/ip_forward
        sysctl -p

    - name: Disable swap
      shell: |
```

```
      swapoff -a
      sed -i -e '/ swap / s/^/#/' -e '/\/swap.img/ s/^/#/' /etc/fstab #
          ↪ comments out swap in fstab

  - name: Join the Kubernetes cluster
    shell: |
      kubeadm join 10.93.246.87:6443 --token 0v7aoq.65ib2v0g70a6em49
          ↪ --discovery-token-ca-cert-hash
          ↪ sha256:9cb5e62dd86cd7e94718c866575cd023c98cc89f2849dad3d25dfd75b13d1b72
```

Appendix 4/2: Flannel configuration

```yaml
apiVersion: v1
kind: Namespace
metadata:
  labels:
    k8s-app: flannel
    pod-security.kubernetes.io/enforce: privileged
  name: kube-flannel
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: flannel
  name: flannel
  namespace: kube-flannel
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    k8s-app: flannel
  name: flannel
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - nodes/status
  verbs:
  - patch
- apiGroups:
  - networking.k8s.io
  resources:
  - clustercidrs
  verbs:
  - list
```

```yaml
  - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    k8s-app: flannel
  name: flannel
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: flannel
subjects:
- kind: ServiceAccount
  name: flannel
  namespace: kube-flannel
---
apiVersion: v1
data:
  cni-conf.json: |
    {
      "name": "cbr0",
      "cniVersion": "0.3.1",
      "plugins": [
        {
          "type": "flannel",
          "delegate": {
            "hairpinMode": true,
            "isDefaultGateway": true
          }
        },
        {
          "type": "portmap",
          "capabilities": {
            "portMappings": true
          }
        }
      ]
    }
  net-conf.json: |
    {
      "Network": "172.16.0.0/16",
      "Backend": {
        "Type": "vxlan"
      }
    }
kind: ConfigMap
metadata:
  labels:
    app: flannel
    k8s-app: flannel
```

```
    tier: node
  name: kube-flannel-cfg
  namespace: kube-flannel
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    app: flannel
    k8s-app: flannel
    tier: node
  name: kube-flannel-ds
  namespace: kube-flannel
spec:
  selector:
    matchLabels:
      app: flannel
      k8s-app: flannel
  template:
    metadata:
      labels:
        app: flannel
        k8s-app: flannel
        tier: node
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/os
                operator: In
                values:
                - linux
      containers:
      - args:
        - --ip-masq
        - --kube-subnet-mgr
        command:
        - /opt/bin/flanneld
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: EVENT_QUEUE_DEPTH
          value: "5000"
```

```
        image: docker.io/flannel/flannel:v0.22.0
        name: kube-flannel
        resources:
          requests:
            cpu: 100m
            memory: 50Mi
        securityContext:
          capabilities:
            add:
            - NET_ADMIN
            - NET_RAW
          privileged: false
        volumeMounts:
        - mountPath: /run/flannel
          name: run
        - mountPath: /etc/kube-flannel/
          name: flannel-cfg
        - mountPath: /run/xtables.lock
          name: xtables-lock
      hostNetwork: true
      initContainers:
      - args:
        - -f
        - /flannel
        - /opt/cni/bin/flannel
        command:
        - cp
        image: docker.io/flannel/flannel-cni-plugin:v1.1.2
        name: install-cni-plugin
        volumeMounts:
        - mountPath: /opt/cni/bin
          name: cni-plugin
      - args:
        - -f
        - /etc/kube-flannel/cni-conf.json
        - /etc/cni/net.d/10-flannel.conflist
        command:
        - cp
        image: docker.io/flannel/flannel:v0.22.0
        name: install-cni
        volumeMounts:
        - mountPath: /etc/cni/net.d
          name: cni
        - mountPath: /etc/kube-flannel/
          name: flannel-cfg
      priorityClassName: system-node-critical
      serviceAccountName: flannel
      tolerations:
      - effect: NoSchedule
        operator: Exists
      volumes:
```

```
      - hostPath:
          path: /run/flannel
        name: run
      - hostPath:
          path: /opt/cni/bin
        name: cni-plugin
      - hostPath:
          path: /etc/cni/net.d
        name: cni
      - configMap:
          name: kube-flannel-cfg
        name: flannel-cfg
      - hostPath:
          path: /run/xtables.lock
          type: FileOrCreate
        name: xtables-lock
```

Appendix 4/3: Bash verification script

```bash
#!/bin/bash

# Define color codes
RED='\033[0;31m'
GREEN='\033[0;32m'
NC='\033[0m' # No Color

# Initialize error flag
error_flag=0

# Function to print info messages
info() {
    echo -e "${GREEN}[INFO] $1${NC}"
}

# Function to print error messages
fail() {
    echo -e "${RED}[ERROR] $1${NC}"
    error_flag=1
}

# Checking installation of necessary packages
dpkg -l | grep -qw apt-transport-https || fail "apt-transport-https is not installed"
dpkg -l | grep -qw ca-certificates || fail "ca-certificates is not installed"
dpkg -l | grep -qw curl || fail "curl is not installed"
dpkg -l | grep -qw kubelet || fail "kubelet is not installed"
dpkg -l | grep -qw kubeadm || fail "kubeadm is not installed"
dpkg -l | grep -qw kubectl || fail "kubectl is not installed"
dpkg -l | grep -qw containerd || fail "containerd is not installed"

# Check Kubernetes APT source list
grep -q "https://apt.kubernetes.io/ kubernetes-xenial main" /etc/apt/sources.list.d/kubernetes.list || fail "Kubernetes APT source list is not configured correctly"

# Check if swap is disabled
swapon --summary | grep -q swap && fail "Swap is not disabled"

# Check containerd configuration
grep -q 'SystemdCgroup = true' /etc/containerd/config.toml || fail "SystemdCgroup is not enabled in containerd configuration"

# Check sysctl parameters
[ "$(cat /proc/sys/net/bridge/bridge-nf-call-iptables)" == "1" ] || fail "bridge-nf-call-iptables is not enabled"
[ "$(cat /proc/sys/net/ipv4/ip_forward)" == "1" ] || fail "ip_forward is not enabled"

```

```
# Check proxy settings for services
[ -f /etc/systemd/system/containerd.service.d/http-proxy.conf ] || fail "Proxy settings for containerd service is not configured"
[ -f /etc/systemd/system/kubelet.service.d/http-proxy.conf ] || fail "Proxy settings for kubelet service is not configured"

# Check Kubernetes node status
if command -v kubectl &> /dev/null; then
    kubectl get nodes || fail "Failed to get Kubernetes nodes. Check if the node has joined the cluster successfully"
else
    info "kubectl command not found. Skipping Kubernetes node check"
fi

# Check status of services
if systemctl --all --type=service --state=active | grep -qw containerd; then
    systemctl is-active --quiet containerd || fail "containerd service is not running"
else
    info "containerd service not found. Skipping service status check"
fi

if systemctl --all --type=service --state=active | grep -qw kubelet; then
    systemctl is-active --quiet kubelet || fail "kubelet service is not running"
else
    info "kubelet service not found. Skipping service status check"
fi

# Print summary
if [ $error_flag -eq 0 ]; then
    info "All checks passed successfully."
else
    echo -e "${RED}Some checks failed. Please check the error messages above.${NC}"
fi
```

Appendix 4/4: Arkouda Setup

````

# Arkouda

Based on the helm charts in the [Arkouda Contrip repository](https://github.com/Bears-R-Us/arkouda-contrib/tree/main/arkouda-helm-c
we can now start to deploy Arkouda in our kubernetes Kluster.
These installation instructions are based on the readme of the same repo.

```bash
git clone git@github.com:Bears-R-Us/arkouda-contrib.git
```

## Namespace

For this we create its own namespace.

```bash
kubectl create namespace arkouda
```

If you want to make your live a little bit easier and work with many differnt namespaces, you can add the following alias to your `.bashrc` or `.zshrc` file.

```bash
alias kark='kubectl --namespace arkouda'
```

This keeps you from having to type `--namespace arkouda` or `-n arkouda` every time you want to interact with the arkouda namespace.

## Secrets

To get the containers to to talk to each other and to interface with the kubernetes api we need to create some secrets.

### SSH

The first secret we create is the ssh secret. This is used to connect to the pods and to the kubernetes api. \
As requested by the [dokumentation](https://github.com/Bears-R-Us/arkouda-contrib/tree/3e4050bfef2bf2a this ssh key needs to be created while impersonating a user with the `ubuntu` username.

```bash
adduser ubuntu --disabled-password --gecos ""
su ubuntu -c "ssh-keygen -t rsa -b 4096 -C \"ubuntu@arkouda\" -f ~/id_rsa -q -N \"\""
````
````

````

then we create the secret
kark create secret generic arkouda-ssh --from-file=id_rsa=./id_rsa --from-file=id_rsa.pub=./id_rsa.pub
```

### SSL

The second secret we need is a ssl secret. This is used to connect to the Kubernetes API. \
This secret is created by generating a self signed certificate.

```bash

# we start by generating the certificate
# note do not change the name of the certificate, as it is hardcoded in the yaml file
openssl genrsa -out tls.key 2048

# creating the certificate signing request
openssl req -new -key tls.key -out tls.csr -subj "/CN=arkouda/O=group1"


# now we create a CSR object in the kubernetes api

cat <<EOF | kark apply -f -
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: arkouda
spec:
  request: $(cat tls.csr | base64 | tr -d '\n')
  signerName: kubernetes.io/kube-apiserver-client
  usages:
  - digital signature
  - key encipherment
  - client auth
EOF

# and get it approved by an admin
kark certificate approve arkouda


# from this we get the certificate
kark get csr arkouda -o jsonpath='{.status.certificate}' | base64 --decode > tls.crt

# now we can verify whether the certificate is valid (this is specific to minikube)
curl --cacert /home/<your username>/.minikube/ca.crt --cert ./tls.crt --key ./tls.key https://$(minikube ip):8443/api/
````

```


# and create the secret
kark create secret generic arkouda-tls --from-file=tls.crt=./tls.crt
    ↪ --from-file=tls.key=./tls.key
```
```

Cluster Role

The following section is an excerpt of the [Arkouda UDP Server
    ↪ documentation](https://github.com/Bears-R-Us/arkouda-contrib/tree/3e4050bfef2bf2a2

ClusterRoles

The Kubernetes API permissions are in the form of a ClusterRole (scoped to all
    ↪ namespaces). For the purposes of this demonstration, the ClusterRoles
    ↪ are as follows. Corresponding Role definitions only differ in that that
    ↪ the Kind field is Role and metadata has a namespace element.

GASNET udp Integration

The arkouda-udp-server deployment discovers all arkouda-udp-locale pods on
    ↪ startup to create the GASNET udp connections between all Arkouda
    ↪ locales. Accordingly, Arkouda requires Kubernetes pod list and get
    ↪ permissions. The corresponding ClusterRole is as follows:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: arkouda-pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

This ClusterRole is bound to the arkouda Kubernetes user as follows:

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: arkouda-pod-reader-binding
subjects:
- kind: User
  name: arkouda
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: pod-reader
```
```

````
  apiGroup: rbac.authorization.k8s.io
```

### Service Integration

Arkouda-on-Kubernetes integrates with Kubernetes service discovery by creating a Kubernetes service upon arkouda-udp-server startup and deleting the Kubernetes service upon teardown. Consequently, Arkouda-on-Kubernetes requires full Kubernetes service CRUD permissions to enable service discovery. The corresponding ClusterRole is as follows:

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: service-endpoints-crud
rules:
- apiGroups: [""]
  resources: ["services","endpoints"]
  verbs: ["get","watch","list","create","delete","update"]
```

This ClusterRole is bound to the arkouda Kubernetes user as follows:

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: arkouda-service-endpoints-crud
subjects:
- kind: User
  name: arkouda
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: service-endpoints-crud
  apiGroup: rbac.authorization.k8s.io
```

Locale-Pods

Now we can edit the `arkouda-udp-locale.yaml` file to match our needs. \
For reference, the following is the configuration on my test setup.

```yaml
####################### Pod Settings #######################

imageRepository: bearsrus
releaseVersion: v2023.05.05
imagePullPolicy: IfNotPresent

````

````
resources:
  limits:
    cpu: 1000m
    memory: 1024Mi
  requests:
    cpu: 1000m
    memory: 1024Mi

################ Arkouda Locale Configuration ################

server:
  port: # Arkouda port, defaults to 5555
  memTrack: true
  numLocales: 4
  threadsPerLocale: 4
external:
  persistence:
    enabled: false
    path: /arkouda-files # pod directory path, must match arkouda-udp-server
    hostPath: /mnt/arkouda # host directory path, must match arkouda-udp-server
secrets:
  tls: arkouda-tls # name of tls secret used to access Kubernetes API
  ssh: arkouda-ssh # name of ssh secret used to launch Arkouda locales
```

These can be deployed by moving into the `arkouda-helm-charts` dir and running the following command:

```bash
helm install -n arkouda arkouda-locale arkouda-udp-locale/
```

### Arkouda-Server

Same goes for the `arkouda-udp-server.yaml` file. \
For reference, the following is the configuration on my test setup.
(to find out what the `k8sHost` is, run `kubectl cluster-info`)

```yaml
resources:
  limits:
    cpu: 1000m
    memory: 1024Mi
  requests:
    cpu: 1000m
    memory: 1024Mi

####################### Pod Settings #######################

imageRepository: bearsrus
releaseVersion: v2023.05.05
````

```
imagePullPolicy: IfNotPresent

################ Arkouda Driver Configuration ################

server:
  numLocales: 1 # total number of Arkouda locales = number of arkouda-udp-locale pods + 1
  authenticate: false # whether to require token authentication
  verbose: true # enable verbose logging
  threadsPerLocale: 5 # number of cpu cores to be used per locale
  memMax: 2000 # maximum bytes of RAM to be used per locale
  memTrack: true
  logLevel: LogLevel.DEBUG
  service:
    type: ClusterIP # k8s service type, usually ClusterIP, NodePort, or LoadBalancer
    port: # k8s service port Arkouda is listening on, defaults to 5555
    nodeport: # if service type is Nodeport
    name: # k8s service name
  metrics:
    collectMetrics: false # whether to collect metrics and make them available via  k8s service
    service:
      name: # k8s service name for the Arkouda metrics service endpoint
      port: # k8s service port for the Arkouda metrics service endpoint, defaults to 5556
      targetPort: # k8s targetPort mapping to the Arkouda metrics port, defaults to 5556
locale:
  appName: arkouda-locale
  podMethod: GET_POD_IPS
external:
  persistence:
    enabled: true
    path: /opt/locale # pod directory path, must match arkouda-udp-locale
    hostPath: /mnt/arkouda # host machine path, must match arkouda-udp-locale
  k8sHost: https://192.168.49.2:8443
  namespace: arkouda # namespace Arkouda will register service
  service:
    name: arkoudaserver # k8s service name Arkouda will register
    port: # k8s service port Arkouda will register, defaults to 5555
metricsExporter:
  imageRepository: bearsrus
  releaseVersion: v2023.05.05 # prometheus-arkouda-exporter release version
  imagePullPolicy: IfNotPresent
  service:
    name: # prometheus-arkouda-exporter service name
    port: 5080 # prometheus-arkouda-exporter service port, defaults to 5080
  pollingIntervalSeconds: 5
secrets:
  tls: arkouda-tls # name of tls secret used to access Kubernetes API
```

~~~~
  ssh: arkouda-ssh # name of ssh secret used to launch Arkouda locales
```

Which can be deployed by moving into the `arkouda-helm-charts` dir and running the following command:

```bash
helm install -n arkouda arkouda-server arkouda-udp-server/
```

Horray! We now have a working Arkouda cluster running in our kubernetes cluster.

# Pachykouda - Client

Now we have to create an image which enables pachyderm to send messages to the arkouda cluster.
To accomplish this we need to create a docker image which contains the arkouda client, takes the arkouda server ip and arbitrary arkouda commands as arguments and then executes the commands on the server.

## Local Registry

To be able to develop and deploy this image locally, we need to set up a local docker registry within the kubernetes cluster.

```bash
sudo mkdir -p /mnt/registry/certs

# create the certificate

sudo openssl req -newkey rsa:4096 -nodes -sha256 -keyout /mnt/registry/certs/registry.key -addext "subjectAltName = DNS:master-node-k8" -x509 -days 365 -out /mnt/registry/certs/registry.crt

sudo chown -R nobody:nogroup /mnt/registry
```

Now if you want to push or pull from this repository you need to add the certificate to your trusted certificates.

```bash
sudo -S bash -c 'openssl s_client -showcerts -connect heydar20.labs.hpecorp.net:31320 </dev/null 2>/dev/null | openssl x509 -outform PEM > /tmp/heydar20.labs.hpecorp.net.pem && mkdir -p /etc/docker/certs.d/heydar20.labs.hpecorp.net:31320 && cp /tmp/heydar20.labs.hpecorp.net.pem /etc/docker/certs.d/heydar20.labs.hpecorp.net:31320/ca.crt && systemctl restart docker'

```
~~~~

Appendix 4/5: Docker Registry Deployment

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: pachyderm
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: registry-pv
  labels:
    type: local
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: registry
  local:
    path: /mnt/registry/
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - heydar20.labs.hpecorp.net
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: registry-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: registry
  selector:
    matchLabels:
      type: local
---
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: cluster-registry
  labels:
    app: cluster-registry
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cluster-registry
  template:
    metadata:
      labels:
        app: cluster-registry
    spec:
      volumes:
      - name: certs-vol
        hostPath:
          path: /mnt/registry/certs
          type: Directory
      - name: registry-vol
        persistentVolumeClaim:
          claimName: registry-pvc

      containers:
        - image: registry:2
          name: cluster-registry
          imagePullPolicy: IfNotPresent
          env:
          - name: REGISTRY_HTTP_TLS_CERTIFICATE
            value: "/certs/registry.crt"
          - name: REGISTRY_HTTP_TLS_KEY
            value: "/certs/registry.key"
          ports:
            - containerPort: 5000
          volumeMounts:
          - name: certs-vol
            mountPath: /certs
          - name: registry-vol
            mountPath: /var/lib/registry
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: cluster-registry
  name: cluster-registry
spec:
  ports:
  - port: 5000
    nodePort: 31320
    protocol: TCP
    targetPort: 5000
```

```
  selector:
    app: cluster-registry
  type: NodePort
```

List of literature

24 Top AI Statistics & Trends In 2023 – Forbes Advisor (2023). URL: `https://www.forbes.com/advisor/business/ai-statistics/` (retrieval: 10/07/2023).

Ansible (2023). Ansible. URL: `https://github.com/ansible/ansible` (retrieval: 10/22/2023).

Argo-Cd/LICENSE at Master · Argoproj/Argo-Cd (2023). GitHub. URL: `https://github.com/argoproj/argo-cd/blob/master/LICENSE` (retrieval: 10/21/2023).

Argoproj/Argo-Workflows (2023). Argo Project. URL: `https://github.com/argoproj/argo-workflows` (retrieval: 10/07/2023).

Arkouda Gituhb Repository (2023). Bears-R-Us. URL: `https://github.com/Bears-R-Us/arkouda` (retrieval: 10/23/2023).

Arkouda-Contrib/Arkouda-Docker at Main · Bears-R-Us/Arkouda-Contrib (2023). GitHub. URL: `https://github.com/Bears-R-Us/arkouda-contrib/tree/main/arkouda-docker` (retrieval: 10/23/2023).

Artifacthub Pachyderm 2.6.4 (2023). URL: `https://artifacthub.io/packages/helm/pachyderm/pachyderm/2.6.4` (retrieval: 10/18/2023).

Bears-R-Us/Arkouda-Contrib/Arkouda-Helm-Charts (2023). GitHub. URL: `https://github.com/Bears-R-Us/arkouda-contrib/tree/main/arkouda-helm-charts/arkouda-udp-server` (retrieval: 10/25/2023).

Boehm, B. W. (1988): A Spiral Model of Software Development and Enhancement. In: *Computer* 21.5, pp. 61–72. ISSN: 0018-9162. DOI: `10.1109/2.59`. URL: `http://ieeexplore.ieee.org/document/59/` (retrieval: 10/18/2023).

Budde, R./Kautz, K./Kuhlenkamp, K./Züllighoven, H. (1992): What Is Prototyping? In: *Information Technology & People* 6.2/3, pp. 89–95. ISSN: 0959-3845. DOI: `10.1108/EUM0000000003546`. URL: `https://doi.org/10.1108/EUM0000000003546` (retrieval: 10/02/2023).

Byrne, J./Kuno, H./Ghosh, C./Shome, P. (2023): Coupling Chapel-Powered HPC Workflows for Python. In.

Bzeznik, B./Henriot, O./Reis, V./Richard, O./Tavard, L. (2017): Nix as HPC Package Management System. In: *Proceedings of the Fourth International Workshop on HPC User Support Tools*. HUST'17. New York, NY, USA: Association for Computing Machinery, pp. 1–6. ISBN: 978-1-4503-5130-0. DOI: `10.1145/3152493.3152556`. URL: `https://dl.acm.org/doi/10.1145/3152493.3152556` (retrieval: 10/03/2023).

Canon, R. S./Younge, A. (2019): A Case for Portability and Reproducibility of HPC Containers. In: *2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*. 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC). Denver, CO, USA: IEEE, pp. 49–54. ISBN: 978-1-72816-028-3. DOI: `10.1109/CANOPIE-HPC49598.2019.00012`. URL: `https://ieeexplore.ieee.org/document/8950982/` (retrieval: 10/04/2023).

Ceph.Io — Home (2023). URL: `https://ceph.io/en/` (retrieval: 10/22/2023).

Chapel-Lang (2023): *Chapel-Lang/Chapel: A Productive Parallel Programming Language*. URL: `https://github.com/chapel-lang/chapel` (retrieval: 10/23/2023).

Chou, C. C./Chen, Y./Milojicic, D./Reddy, N./Gratz, P. (2019): Optimizing Post-Copy Live Migration with System-Level Checkpoint Using Fabric-Attached Memory. In: *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. 2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC). Denver, CO, USA: IEEE, pp. 16–24. ISBN: 978-1-72816-007-8. DOI: 10.1109/MCHPC49590.2019.00010. URL: https://ieeexplore.ieee.org/document/8946189/ (retrieval: 10/29/2023).

Cluster Networking (2023). Kubernetes. URL: https://kubernetes.io/docs/concepts/cluster-administration/networking/ (retrieval: 10/18/2023).

Creating a Cluster with Kubeadm (2023). Kubernetes. URL: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/ (retrieval: 10/18/2023).

Cwl-Utils/LICENSE at Main · Common-Workflow-Language/Cwl-Utils (2023). GitHub. URL: https://github.com/common-workflow-language/cwl-utils/blob/main/LICENSE (retrieval: 10/21/2023).

Docker FAQs | Docker (2021). URL: https://www.docker.com/pricing/faq/ (retrieval: 10/18/2023).

Docker Terms of Service | Docker (2022). URL: https://www.docker.com/legal/docker-terms-service/ (retrieval: 10/18/2023).

Docker-Registry 1.10.0 · Phntom/Phntom (2023). URL: https://artifacthub.io/packages/helm/phntom/docker-registry (retrieval: 10/30/2023).

Dubois, P./Epperly, T./Kumfert, G. (2003): Why Johnny Can't Build [Portable Scientific Software]. In: *Computing in Science & Engineering* 5.5, pp. 83–88. ISSN: 1558-366X. DOI: 10.1109/MCISE.2003.1225867. URL: https://ieeexplore.ieee.org/abstract/document/1225867 (retrieval: 10/03/2023).

Eckerth, J. (2023): Installation Instructions Minikube Commit 69a05a755facc8d387e031ff5991c20d46dbe
URL: https://github.com/Jo-Eck/ProjectPaper-2/tree/69a05a755facc8d387e031ff5991c20d46db
project/Pachyderm (retrieval: 10/18/2023).

Egwutuoha, I. P./Levy, D./Selic, B./Chen, S. (2013): A Survey of Fault Tolerance Mechanisms and Checkpoint/Restart Implementations for High Performance Computing Systems. In: *J Supercomput* 65.3, pp. 1302–1326. ISSN: 1573-0484. DOI: 10.1007/s11227-013-0884-0. URL: https://doi.org/10.1007/s11227-013-0884-0 (retrieval: 10/03/2023).

Explore Network Plugins for Kubernetes (2023): Explore Network Plugins for Kubernetes: CNI Explained | TechTarget. IT Operations. URL: https://www.techtarget.com/searchitoperations/tip/Explore-network-plugins-for-Kubernetes-CNI-explained (retrieval: 10/19/2023).

Flannel (2023). flannel-io. URL: https://github.com/flannel-io/flannel (retrieval: 10/19/2023).

Flannel Install Config (2023). URL: https://github.com/flannel-io/flannel/blob/master/Documentation/kube-flannel.yml (retrieval: 10/19/2023).

Fülöp, J. (2005): Introduction to Decision Making Methods. In: *BDEI-3 Workshop, Washington*, pp. 1–15. URL: https://www.academia.edu/download/43447287/decisionmakingmethods.pdf (retrieval: 10/12/2023).

Gamblin, T./LeGendre, M./Collette, M. R./Lee, G. L./Moody, A./de Supinski, B. R./Futral, S. (2015): The Spack Package Manager: Bringing Order to HPC Software Chaos. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15. New York, NY, USA: Association for Computing Machinery, pp. 1–12. ISBN: 978-1-4503-3723-6. DOI: 10.1145/2807591.2807623. URL: https://dl.acm.org/doi/10.1145/2807591.2807623 (retrieval: 10/03/2023).

Gluster (2023). URL: https://www.gluster.org/ (retrieval: 10/22/2023).

Haines, S. (2022): 'Workflow Orchestration with Apache Airflow'. In: *Modern Data Engineering with Apache Spark: A Hands-On Guide for Building Mission-Critical Streaming Applications*. Springer, pp. 255–295.

Helm Docs Home (2023). URL: https://helm.sh/docs/ (retrieval: 10/18/2023).

Home - Knative (2023). URL: https://knative.dev/docs/ (retrieval: 10/07/2023).

Home Page | Pachyderm (2023). URL: https://www.pachyderm.com/ (retrieval: 10/27/2023).

Hoste, K./Timmerman, J./Georges, A./De Weirdt, S. (2012): EasyBuild: Building Software with Ease. In: *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, pp. 572–582. DOI: 10.1109/SC.Companion.2012.81. URL: https://ieeexplore.ieee.org/abstract/document/6495863 (retrieval: 10/03/2023).

Inc, M. (2023): MinIO | MinIO for Kubernetes. MinIO. URL: https://min.io (retrieval: 10/18/2023).

Installing Kubeadm (2023). Kubernetes. URL: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/ (retrieval: 10/22/2023).

Intro to Pipelines (2023). URL: https://docs.pachyderm.com/latest/learn/intro-pipelines/ (retrieval: 10/23/2023).

iPaaS Solution for the Enterprise (2023). SnapLogic. URL: https://www.snaplogic.com/ (retrieval: 10/07/2023).

Keeton, K./Singhal, S./Raymond, M. (2019): The OpenFAM API: A Programming Model for Disaggregated Persistent Memory. In: *OpenSHMEM and Related Technologies. OpenSHMEM in the Era of Extreme Heterogeneity*. Ed. by Swaroop Pophale/Neena Imam/Ferrol Aderholdt/Manjunath Gorentla Venkata. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 70–89. ISBN: 978-3-030-04918-8. DOI: 10.1007/978-3-030-04918-8_5.

Knative Docs/LICENSE at Main · Knative/Docs (2023). GitHub. URL: https://github.com/knative/docs/blob/main/LICENSE (retrieval: 10/21/2023).

Kubeflow (2023). Kubeflow. URL: https://www.kubeflow.org/ (retrieval: 10/07/2023).

Kubeflow/LICENSE at Master · Kubeflow/Kubeflow (2023). GitHub. URL: https://github.com/kubeflow/kubeflow/blob/master/LICENSE (retrieval: 10/21/2023).

Kubernetes CNI Plugins (2023). Kubernetes. URL: https://kubernetes.io/docs/concepts/cluster-administration/addons/ (retrieval: 10/19/2023).

Kumar, P. (2020): How to Setup Private Docker Registry in Kubernetes (K8s). URL: https://www.linuxtechi.com/setup-private-docker-registry-kubernetes/ (retrieval: 10/29/2023).

License — Airflow Documentation (2023). URL: `https://airflow.apache.org/docs/apache-airflow/stable/license.html` (retrieval: 10/21/2023).

Luigi/LICENSE at Master · Spotify/Luigi (2023). GitHub. URL: `https://github.com/spotify/luigi/blob/master/LICENSE` (retrieval: 10/21/2023).

Mehndiratta, H. (2023): Comparing Kubernetes Container Network Interface (CNI) Providers | Kubevious.Io. URL: `https://kubevious.io/blog/post/comparing-kubernetes-container-network-interface-cni-providers` (retrieval: 10/19/2023).

Merrill, M./Reus, W./Neumann, T. (2019): Arkouda: Interactive Data Exploration Backed by Chapel. In: *Proceedings of the ACM SIGPLAN 6th on Chapel Implementers and Users Workshop*. CHIUW 2019. New York, NY, USA: Association for Computing Machinery, p. 28. ISBN: 978-1-4503-6800-1. DOI: `10.1145/3329722.3330148`. URL: `https://dl.acm.org/doi/10.1145/3329722.3330148` (retrieval: 10/04/2023).

Multi-Criteria Decision Analysis for Use in Transport Decision Making (2014). Red. by Michael Bruhn Barfod/Steen Leleur. DTU Lyngby: DTU Transport.

Pachyderm Docs - Local Deploy (2023). Pachyderm Docs - Local Deploy. URL: `https://docs.pachyderm.com/2.6.x/set-up/local-deploy/` (retrieval: 10/18/2023).

Pachyderm Docs - On-Prem Deploy (2023). Pachyderm Docs - On-Prem Deploy. URL: `https://docs.pachyderm.com/latest/set-up/on-prem/` (retrieval: 10/18/2023).

Pachyderm/LICENSE at Master · Pachyderm/Pachyderm (2023). URL: `https://github.com/pachyderm/pachyderm/blob/master/LICENSE` (retrieval: 10/21/2023).

Pahcyderm -Pricing (2022). Pachyderm. URL: `https://www.pachyderm.com/pricing/` (retrieval: 10/21/2023).

Roberts, R./Goodwin, P. (2002): Weight Approximations in Multi-Attribute Decision Models. In: *Journal of Multi-Criteria Decision Analysis* 11.6, pp. 291–303. ISSN: 1099-1360. DOI: `10.1002/mcda.320`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/mcda.320` (retrieval: 10/16/2023).

Rook (2023). URL: `https://rook.io/` (retrieval: 10/22/2023).

Sayers, C./Laffitte, H./Reddy, P./Ozonat, K./Sayal, M./Simitsis, A./Singhal, S./Koutrika, G./Das, M./Aji, A./Bosamiya, H./Riss, M./Wilkinson, K./Lucio, J./Cantal, A./Carvalho, C. (2015): Cloud Application Services Platform (CLASP): User Guide, Introduction, and Operation. In: pp. 1–60.

Sayers, Craig/Laffitte, H./Reddy, P./Ozonat, K./Sayal, M./Simitsis, A./Singhal, S./Koutrika, G./Das, M./Aji, A./Bosamiya, H. A./Riss, M./Wilkinson, K. (2015): CLoud Application Services Platform. In.

SnapLogic – Master Subscription Agreement (2023). SnapLogic. URL: `https://www.snaplogic.com/master-subscription-agreement` (retrieval: 10/21/2023).

Spotify/Luigi (2023). Spotify. URL: `https://github.com/spotify/luigi` (retrieval: 10/12/2023).

Triantaphyllou, E. (2000): 'Introduction to Multi-Criteria Decision Making'. In: *Multi-Criteria Decision Making Methods: A Comparative Study*. Ed. by Evangelos Triantaphyllou. Applied Optimization. Boston, MA: Springer US, pp. 1–4. ISBN: 978-1-4757-3157-6. DOI: `10.1007/978-`

1-4757-3157-6_1. URL: https://doi.org/10.1007/978-1-4757-3157-6_1 (retrieval: 10/12/2023).

Weighted Sum Model (2022). In: *Wikipedia*. URL: https://en.wikipedia.org/w/index.php?title=Weighted_sum_model&oldid=1114224941 (retrieval: 10/16/2023).

Wilde, T./Hess, T. (w.y.): Methodenspektrum der Wirtschaftsinformatik: Überblick und Portfoliobildung. In.