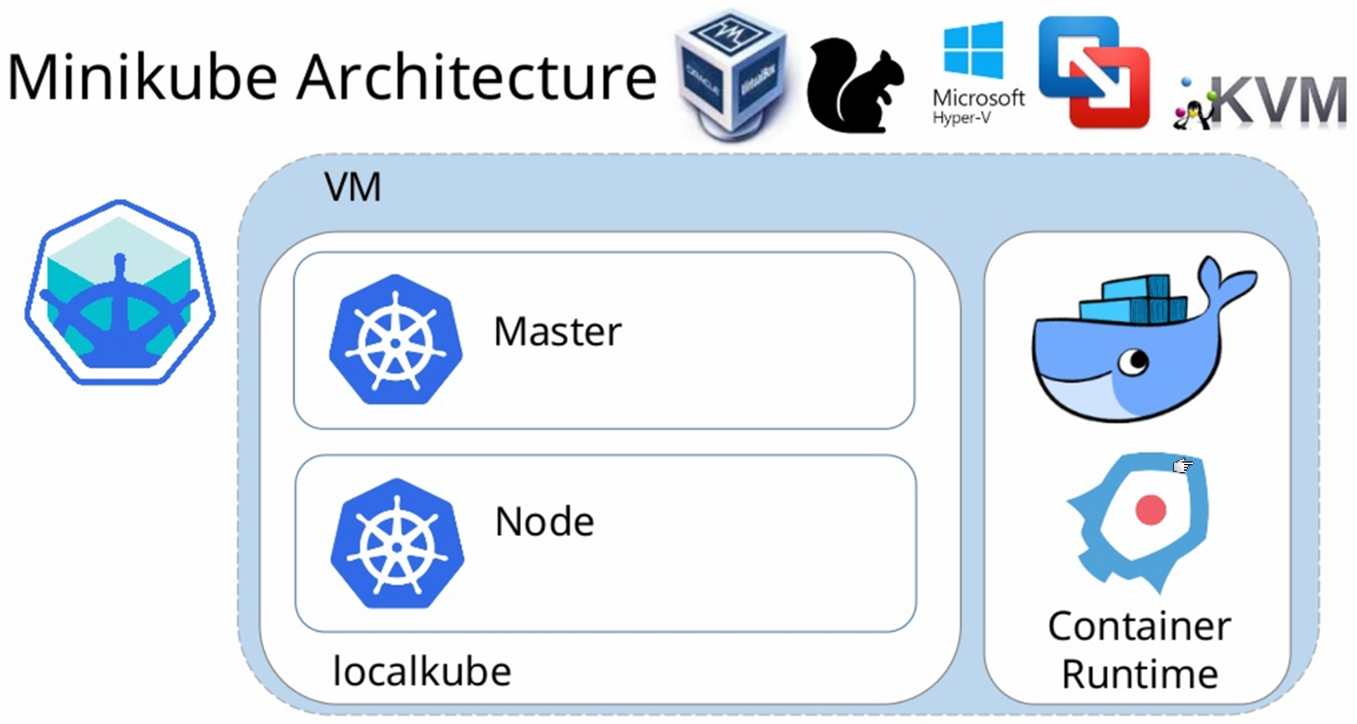
# Introduction

With new advancements in object detection and classification, interest in improving image processing speed and workflow has grown. Along the processing path, numerous techniques may be applied to an image before it reaches the desired state, making image processing a good distributed processing problem. We can improve the processing workflow by treating each processing stage as a micro-service, and stages can be arranged, rearranged, added or even removed from the data processing pipeline all together. This paper will demonstrate a distributed system that colorizes black and white film using Kubernetes for cluster management, Pachyderm for distributed storage and processing, and Docker for hosting jobs.

Architecture

Minikube

The Distributed Video Colorization System uses Minikube for cluster management. Though it can be scaled to other physical or virtual machines, Minikube is a Kubernetes tool for running a single node cluster locally. Minikube runs on a VM running Linux, and for every pipeline we create, a new pod or Docker container is spun up in the Minikube VM. More on pipelines later.

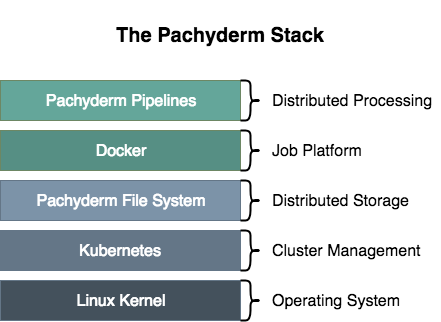


Docker

Containers are lighter and easier to startup up when compared to a VM, which makes them great for executing multiple micro-applications at the same time on the same server. This is mostly due to the fact that containers virtualize the OS instead of virtualizing hardware; so spinning up a new container does not require booting up the OS. This allows us to spin up a new container to execute different stages of our image-processing pipeline.

Pachyderm

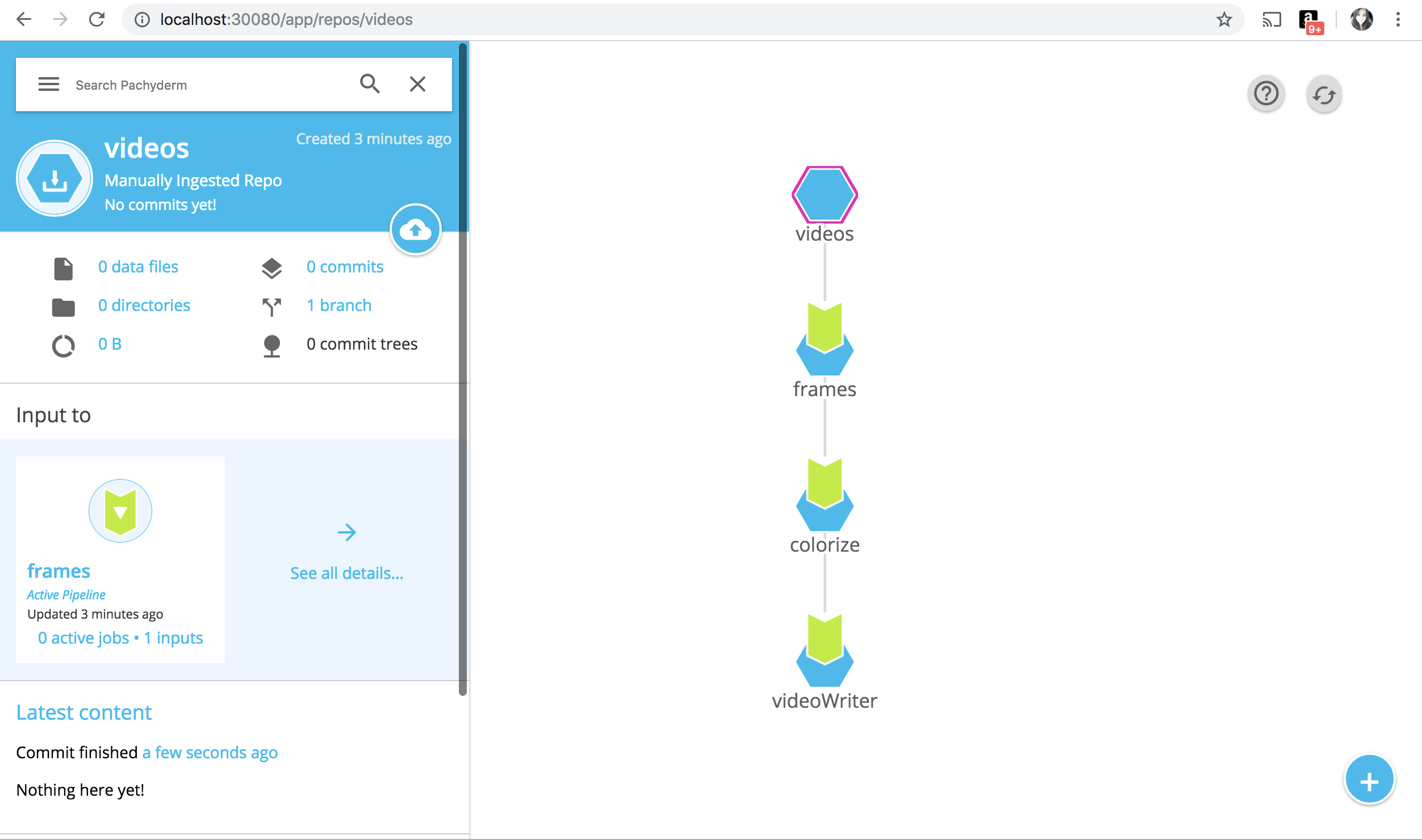
The final component is Pachyderm. Pachyderm adds distributed data storage to the stack and a processing layer that aids in mounting the Pachyderm File System to a container and synchronization between container accesses. In other words, Pachyderm handles concurrency between reads and writes from multiple running containers accessing the same repository, so the application code is simple, only needing to read from PFS and write out to the PFS. Below is an overview of the entire development stack.



# Implementation

Overview

The processing takes place in three stages. First videos run through a pipeline that extracts all the frames using OpenCV. Second, the frames are run through a pipeline that uses Zhang’s Image Colorization, a CNN for colorizing images using object detection and classification []. Finally, the color images are fed into a pipeline that uses OpenCV to compile the frames back into an mp4 video format. The image below shows Pachyderm’s dashboard and the current state of pipelines and data repositories.



Pipelines

Pachyderm pipelines describe which Docker image to pull from the Docker store, application execution commands to be run in the container, and glob descriptions for reading data from the PFS. For instance,line 6 below is simply telling pachyderm how to run my application in the container terminal. Line 7 tells Pachyderm to spin up the Docker image tagged: datasutures/caffe\_colorization:v9. On Lines 10 through 14, *frames* is the repository to mount to the container/s, and “*/\*”* is the glob pattern that describes how the datum are distributed across containers. Here each listing in the repo is treated like a single datum, so they can each be distributed and processed in parallel. This advantageous given colorizing frames is a long process. More reading on glob patterns can be found in Pachyderm’s documentation.



Pipeline scripts for *frames* and *videoWriter* are modeled in the same way with their own input repositories, Docker images and command line inputs. Pachyderm also provides a number of pipeline specifications, including memory and CPU usage configurations.

Dockerfile

A Dockerfile “is a text document that contains all the commands to create an image,” including system libraries and dependencies for the executable code. User scripts for processing data (.py, etc.) are added to the image using the Dockerfile *ADD* command, and when the image is built, the script is copied to the image. This means anytime this image is spun up, it contains all the dependencies and the application code needed to run the application process.

For the Video Colorization System, one Docker image was used in both the frame extraction stage and the video write-back stage, as both stages required an image configured with Python and OpenCV. A second Docker image was configured with Caffe, Python, OpenCV and Zhang’s Colorful Image Colorization computer vision algorithm and used for the image colorization stage. Since Pachyderm mounts its file system to the container, the root directory of the image will have the directory */pfs* and */out* when the container is running. User code can reference these directories like they would any directory in the image, so python image processing scripts read from */pfs*, process the data (extract frames, colorize frames, or write frames), and write the results to */out*.

# Results

# References

[https://linuxhelp4u.blogspot.com/2018/04/minikube-how-to-set-up-local-kubernetes.html //](https://linuxhelp4u.blogspot.com/2018/04/minikube-how-to-set-up-local-kubernetes.html%20//) image\_minikube\_architecture

<https://medium.com/pachyderm-data/lets-build-a-modern-hadoop-4fc160f8d74f>

image\_pachyderm\_minikube