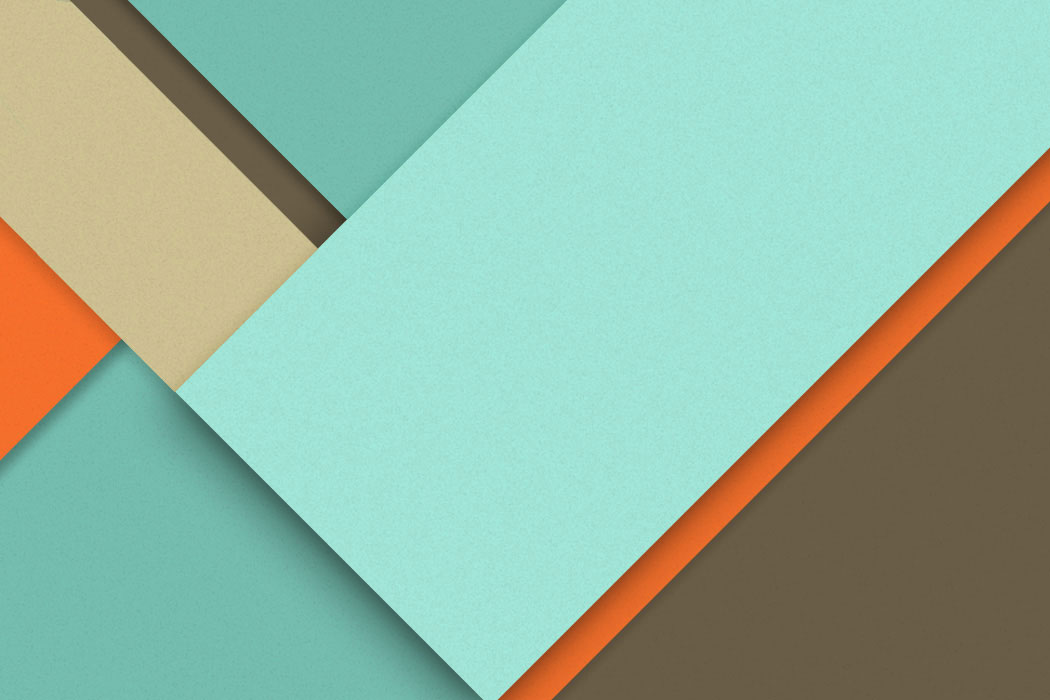
## 



**Cloud Agnostic**

**MlOps Pipeline**

31.10.2022

**─**

Lijo Zechariah James

# [Objective](#_Objective)

# [Proof of Concepts](#_Proof_of_Concepts)

## [MlOps](#_1._MlOps)

## [Cloud agnostic](#_Cloud_agnostic)

## [Docker, Containers](#_Docker,_Containers)

## [Kubernetes](#_Kubernetes)

## [Kubeflow](#_Kubeflow)

### [Git Hub Actions](#_Git_Hub_Actions)

# [Tech Stack](#_Tech_Stack)

# [System requirement and Configuration](#_System_requirement_and)

## [Microk8s](#_Microk8s )

## [Minikube](#_Minikube)

# [Getting Started](#_Implementation _1)

# [Implementation](#_Implementation )

# [Challenges](#_Challenges )

# [Outcome](#_Outcome)

# [Future scope and improvements](#_Future_scope_and)

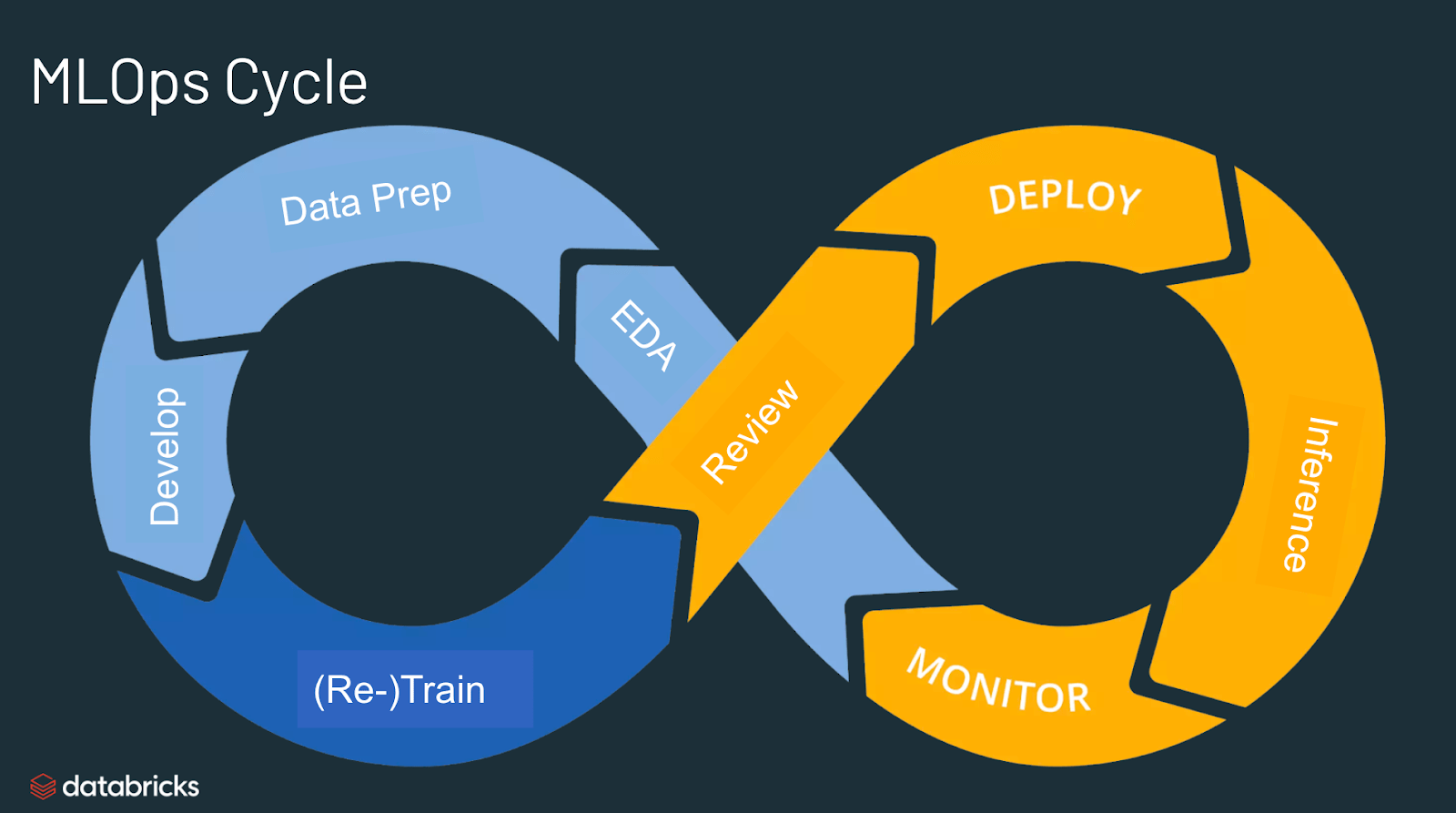
# [Bibliography](#_Bibliography)

# **Objective**

To create a cloud agnostic mlops pipeline using kubeflow. This pipeline will help to reduce the complexities of machine learning program. Since it’s automate the process of data retrieving, preprocessing, splitting etc…. Also containerize each and every step, so that it can be used for other projects.

# **Proof of Concepts**

## 1. MlOps



### **1.1 What is the use of MLOps?**

MLOps stands for Machine Learning Operations. MLOps is a core function of Machine Learning engineering, focused on streamlining the process of taking machine learning models to production, and then maintaining and monitoring them. MLOps is a collaborative function, often comprising data scientists, devops engineers, and IT.

### **1.2 Why do we need MLOps?**

Productionizing machine learning is difficult. The machine learning lifecycle consists of many complex components such as data ingest, data prep, model training, model tuning, model deployment, model monitoring, explainability, and much more. It also requires collaboration and hand-offs across teams, from Data Engineering to Data Science to ML Engineering. Naturally, it requires stringent operational rigor to keep all these processes synchronous and working in tandem. MLOps encompasses the experimentation, iteration, and continuous improvement of the machine learning lifecycle.

Like DevOps, MLOps helps to improve the quality of production models, while incorporating business and regulatory requirements and model governance. A few of the problems that MLOps solves are:

* **Inefficient workflows**.
* **Failing to comply with regulations**
* **Bottlenecks**

### **1.3 What are the benefits of MLOps?**

**1.3.1 Increases productivity**

MLOps increases the productivity of data scientists and machine learning engineers.

Some ways in which it happens is by

* Creating automated pipelines

There are many repetitive tasks in ML modeling. MLOps stand for automating the entire workflow of the ML model. This saves time and avoids human-induced errors. We can avoid wasting time on repetitive tasks and make value-added tasks.

* Standardizing ML workflows for efficient collaboration

This reduces compatibility problems and quickens the construction and deployment of modeling processes.

**1.3.2 Less Expenditure**

* If we have one model, there is no need to hire extra people to develop new versions.
* Since there is no need to purchase additional hardware and software tools for delivering model versions, a significant portion of the operational costs can be avoided.
* It makes it possible for you to identify and minimize errors methodically. Reduced model management errors will also result in lower expenses.
* Automation reduces the need for manual management of machine learning models. Employee time will be freed up as a result and may be put to better use.

**1.3.3 Reproducibility**

* Automating ML workflow provides reproducibility and repeatability regarding how the machine learning model is deployed. This helps in becoming more productive by reducing the time to deploy models.

**1.3.4 Reliability**

* MLOps makes ML pipelines more reliable, human error will be less, and we can get real insights into the data.
* For reliable scaling, MLOps streamlines model management procedures.

**1.3.5 Connectivity**

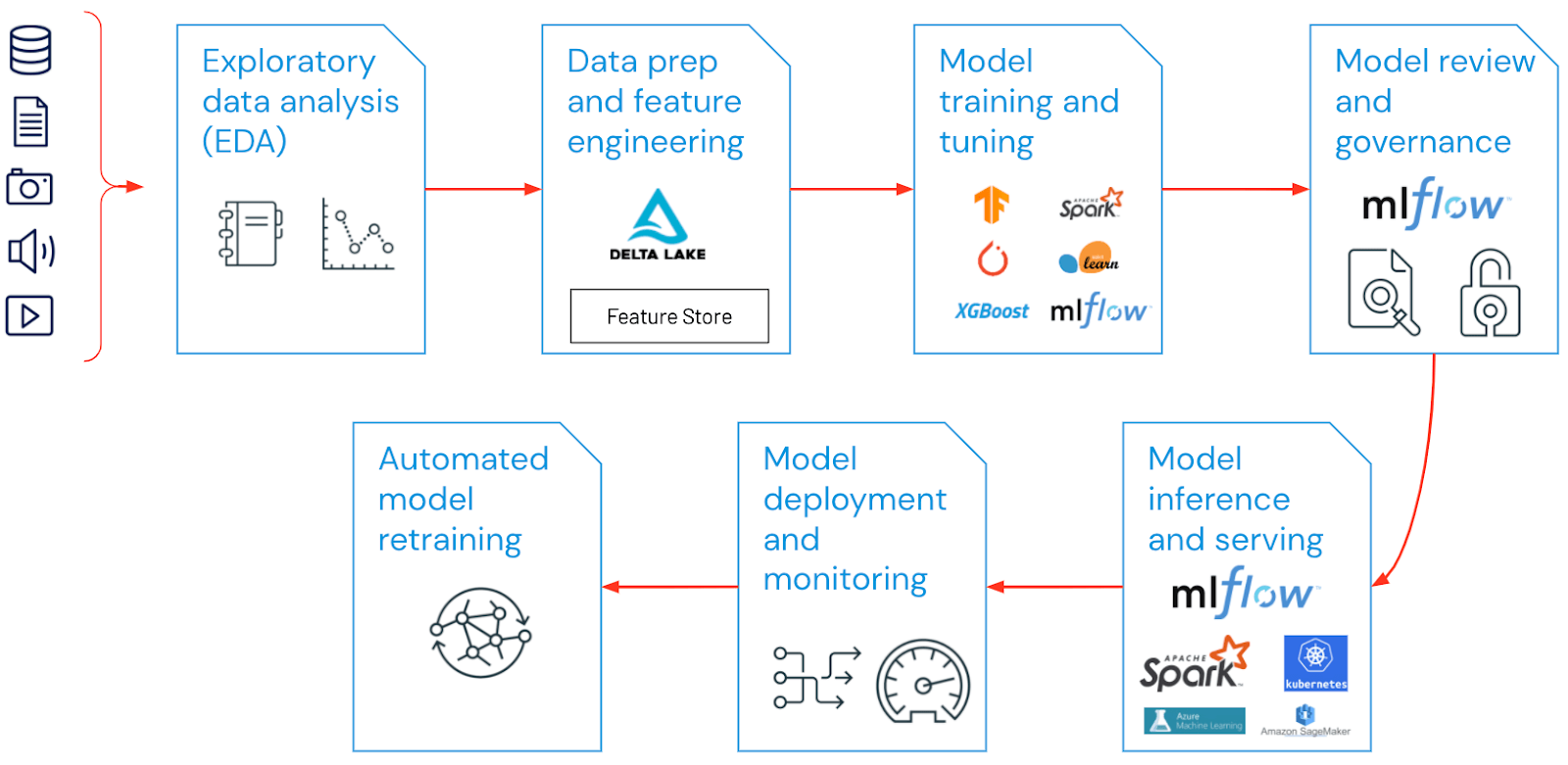
* MLOps enable distributed teams of employees to collaborate on a common delivery model. It reduces deployment time and costs. Also, data collection and training of new models can be done independently by a department or team. Collaborating on a common distribution model allows all departments to share and process data more quickly and efficiently.

**1.3.6 Monitorability**

Through MLOps, we get insights about model performance by

* Retraining the model continuously to make sure that it gives the most accurate output.
* Sending signals if there is any model drift (if the model degrades below a threshold).
* This ensures that the process runs within regulations and that the program returns high-quality information.

### **1.4 What are the components of MLOps?**



* **Exploratory data analysis (EDA)** - Iteratively explore, share, and prep data for the machine learning lifecycle by creating reproducible, editable, and shareable datasets, tables, and visualizations.
* **Data Prep and Feature Engineering**- Iteratively transform, aggregate, and de-duplicate data to create refined features. Most importantly, make the features visible and shareable across data teams, leveraging a feature store.
* **Model training and tuning** - Use popular open source libraries such as scikit-learn and hyperopt to train and improve model performance. As a simpler alternative, use automated machine learning tools such as AutoML to automatically perform trial runs and create reviewable and deployable code.
* **Model review and governance**- Track model lineage, model versions, and manage model artifacts and transitions through their lifecycle. Discover, share, and collaborate across ML models with the help of an open source MLOps platform such as MLflow.
* **Model inference and serving** - Manage the frequency of model refresh, inference request times and similar production-specifics in testing and QA. Use CI/CD tools such as repos and orchestrators (borrowing devops principles) to automate the pre-production pipeline.
* **Model deployment and monitoring** - Automate permissions and cluster creation to productionize registered models. Enable REST API model endpoints.
* **Automated model retraining** - Create alerts and automation to take corrective action In case of model drift due to differences in training and inference data.

### **1.5 What are the best practices for MLOps?**

**1.5.1. Communication and collaboration between roles — “ML products are a team effort”**

Production of a successful machine learning lifecycle is a lot like racing in formula one. From the outside, it appears that the driver is the only one responsible for getting the car around the track, but in reality, there are upwards of 80 team members behind the scenes. This is similar to developing an enterprise-level ML product. The data scientist sits in the driver’s seat, directing how the model will be built every step of the way. However, this assumes data scientists have expertise in every step of development which is commonly not the case. The driver is not going to get out and perform maintenance on the car, they need a team of engineers, mechanics, and strategists to be successful.

**1.5.2. Establish Business Objectives — “Have a clear goal in mind”**

Every business has key performance indicators (KPIs). These are measurable values that reflects how well a company is achieving its objectives. The first step in the machine learning lifecycle is taking a business question, and determining how it can be solved with ML, “What do you want from the data?”. Machine Learning model evaluation metrics are measured by accuracy, precision, and recall, how can the predictions be translated to real-world metrics that can be easily understood for project members outside of the data team?

**1.5.3. Obtaining the Ground Truth — “Validate the dataset”**

Arguably the most important step in developing any machine learning model is the process of verifying the labels of the dataset. Ground truthing is imperative to train the model to result in predictions that accurately reflect the real world.

Legitimizing the source of the data and labeling it correctly can be an arduous process, and even maybe the most time-consuming of all. That is why it is important to recognize the amount of time and resources early on in the development process because depending on the size of the dataset could be a hindrance to model performance. For example, if the model is trained to detect objects in a picture, obtaining the ground truth will involve labeling each observation in the dataset with a bounding box, which has to be done manually.

**1.5.4. Choosing the Right Model — “Experiment and Reproduce”**

Different challenges require different tools, and in this case algorithms. Algorithms can range from very simple like linear regression to advanced deep learning neural networks and everything in between. Each model has its advantages and disadvantages and will pose certain considerations affecting MLOps. The best way to narrow down what algorithm from an MLOps perspective will depend on two things: what kind of data are they performing on, and how well they fit with the CI/CD pipeline. Having these two principles in mind will ultimately reduce dependencies and make life easier when it comes to the deployment phase.

**1.5.5. Determine the Type of Deployment — “Continuous Integration and Delivery”**

There are typically two methods to consider when deploying a model once it reaches the production stage. Either embedding the model into an application, or staging it as a Software-as-a-Service, or in this case “model-as-a-service”. Each has its advantages and disadvantages, costs, and restraints.

But it is important to have this in mind before setting off on the development of a machine learning pipeline because certain software frameworks will only support specific packages. The production environment needs to be cohesive with the model of choice.

MLOps addresses the challenges that arise once the model is ready to enter production. MLOps borrows the continuous integration and continuous delivery (CI/CD) principle commonly used in DevOps. However, the difference between them is that with ML, the data is continuously being updated, as well as the models. While traditional software only requires code to have CI/CD.

**1.5.6. Containerization — “Works every time, all the time”**

Open-source software packages are often very rigid with their dependencies. In turn, this forces software to rely on the exact package versions and modules. Keeping with the theme of streamlining and standardization in MLOps, containerization serves as a way to automate the machine learning environments from development into production. Technologies like Docker serve as a way to practice containerization.

Tools like Docker provides an isolated environment for the model and accompanying applications to run in production. Ensuring the environment for the model and its accompanying application will always have the required packages available.

### **1.6 What is the difference between MLOps and DevOps?**

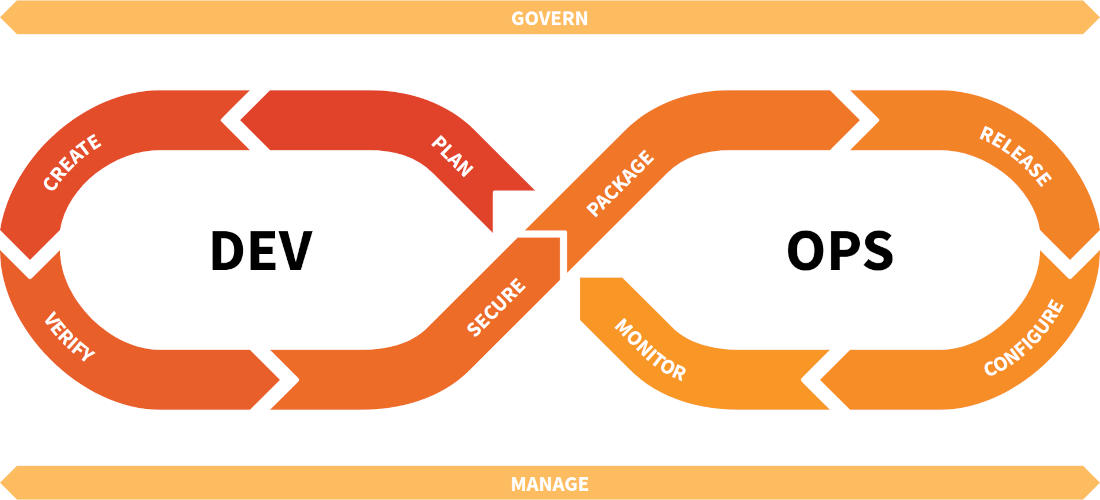
**1.6.1 DevOps**

DevOps is a practice where people work in a team to build and deliver software at the best possible speed. DevOps enable software developers(devs) and operations(Ops) teams to fasten up the delivery of Software through collaboration, and in an iterative manner. DevOps methodology helps improve communication between your developers and ops working on projects. It best serves the following purposes:

* you can launch new features faster
* increases the customer’s satisfaction and of developers too at the same time.
* feedback loops help better communication

Key principles of DevOps:

* Automation
* Iteration
* Self-service
* Continuous improvement
* Continuous testing
* Collaboration



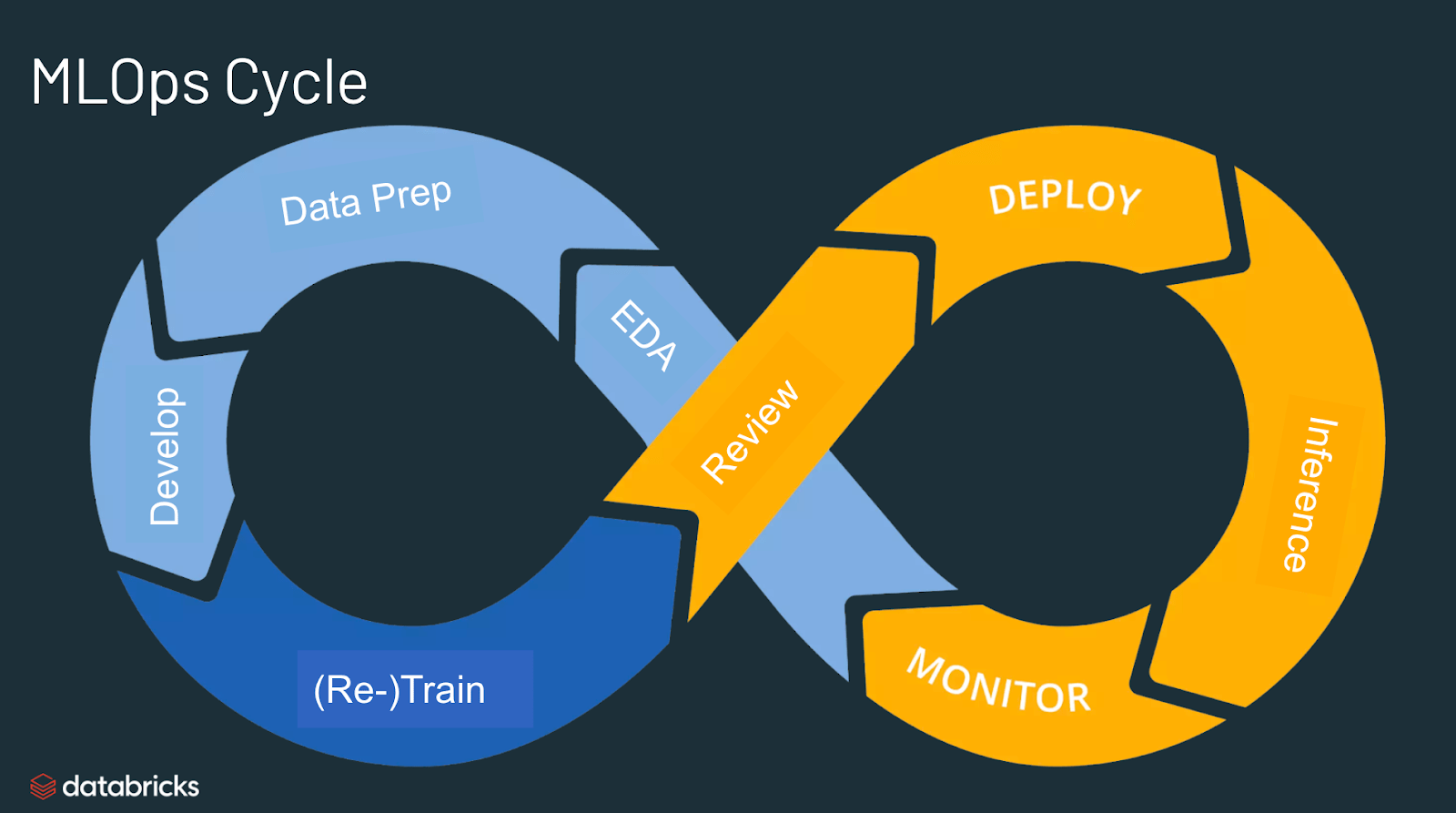
From this figure above we can understand the whole DevOps process. Organizing tasks and schedules and other stuff starts with this very step called plan. Planning starts according to the user stories made in every sprint if you are using agile methodology. Then starts development or coding part of the software. Testing is done of the application developed so far for any bugs. Once code passes this stage of testing (or continuous integration) it is sent for deployment. In the next step, Ops maintain infrastructure and truncates any vulnerabilities or security issues from the software. The last stage is to monitor the application developed for fixing the hiccups to ensure a smooth end-user experience.

**1.6.2 Mlops**

In DevOps, we saw that it was for streamlining software development and then deploying and monitoring them. In MLOps we focus on Machine Learning Operations. So, the guys who are involved in this methodology are data scientists, IT, and DevOps Engineers. It is a useful approach for creating best-in-class machine learning solutions for the end-user. For developing machine learning solutions the standard lifecycle goes like this:

* Requirement gathering
* Exploratory data analysis
* Feature engineering
* Feature selection
* Model creation
* Model hyperparameter tuning
* Model deployment
* Retraining, if needed

So from this whole pipeline, it is understood that developing models is just a very small part of the whole process. Many other configurations, steps, processes, or tools are to be integrated into the system. For this streamlining, we have this machine learning development methodology MLOps.



MLOps also provide the same benefits as in the DevOps. It increases scalability, efficiency and reduces risk to a greater extent of the whole process of developing a machine learning solution.

## **Cloud agnostic**

**“A business is cloud-agnostic when the company IT systems are not locked into a single cloud vendor or do not rely on one cloud provider's proprietary services,”** explains Bailey. “**Typically, services are spread between multiple cloud vendors to preserve and ensure the uptime of critical applications.”**

### **Benefits of being cloud agnostic**

**Flexibility**

A cloud agnostic strategy allows you to switch cloud providers with minimal headache if pricing, performance or offerings change. It also means you can take a multi-cloud approach which sees workloads split between providers.

**Reliability**

Spreading systems and workloads across more than one cloud platform avoids the risk of redundancy and downtime if one encounters problems.

**Avoids Vendor Lock-in**

A ‘pro’ that overlaps with the previous two. Cloud agnostic organisations have risk management in place which makes them generally more robust to changes in the business IT landscape.

### **Challenges to Cloud Agnosticism**

**Front-Loaded Costs & Complexity**

While going cloud agnostic can see huge savings on time, costs and stress down the line, it does involve more front-loaded work. Building an agnostic cloud strategy from the ground up, either for particular workloads or across an organisation, can be expected to increase initial development costs and complexity.

**Vendor Lock-Out**

A strict cloud agnostic approach could mean an organisation limits itself to using services available across the public cloud providers its multi-cloud strategy uses. If one of AWS, Microsoft Azure, Alibaba and Google Cloud Platform, or another larger provider, introduces a handy new feature, a strictly cloud agnostic strategy may mean there is a reluctance to use it because it can’t be replicated on another platform.

### **Strategies and Tools for Going Cloud Agnostic**

There are increasing choices when it comes to a cloud agnostic development tech stack but here are some of the most popular ingredients K&C’s engineers use to bake apps as comfortable on one cloud platform as another:

**Automate-First Strategy**

Because a cloud agnotistic strategy inherently increases the workload shouldered by DevOps, an automate-first approach is key. Infrastructure should be spinnable with little to no manual effort and testing and integration managed through a scheduled CI pipeline requiring approvals.

**Microservices**

Microservices architecture structures an application as a collection of services that are broken into small units, each serving a specific business function and interacting with each other. Applications, especially if intended as cloud agnostic, are easier to both build and maintain when broken down into more manageable microservice units. Microservices are:

* Highly maintainable and testable
* Loosely coupled
* Independently deployable
* Organized around business capabilities
* Owned by a small team

### **Should You Go Cloud Agnostic?**

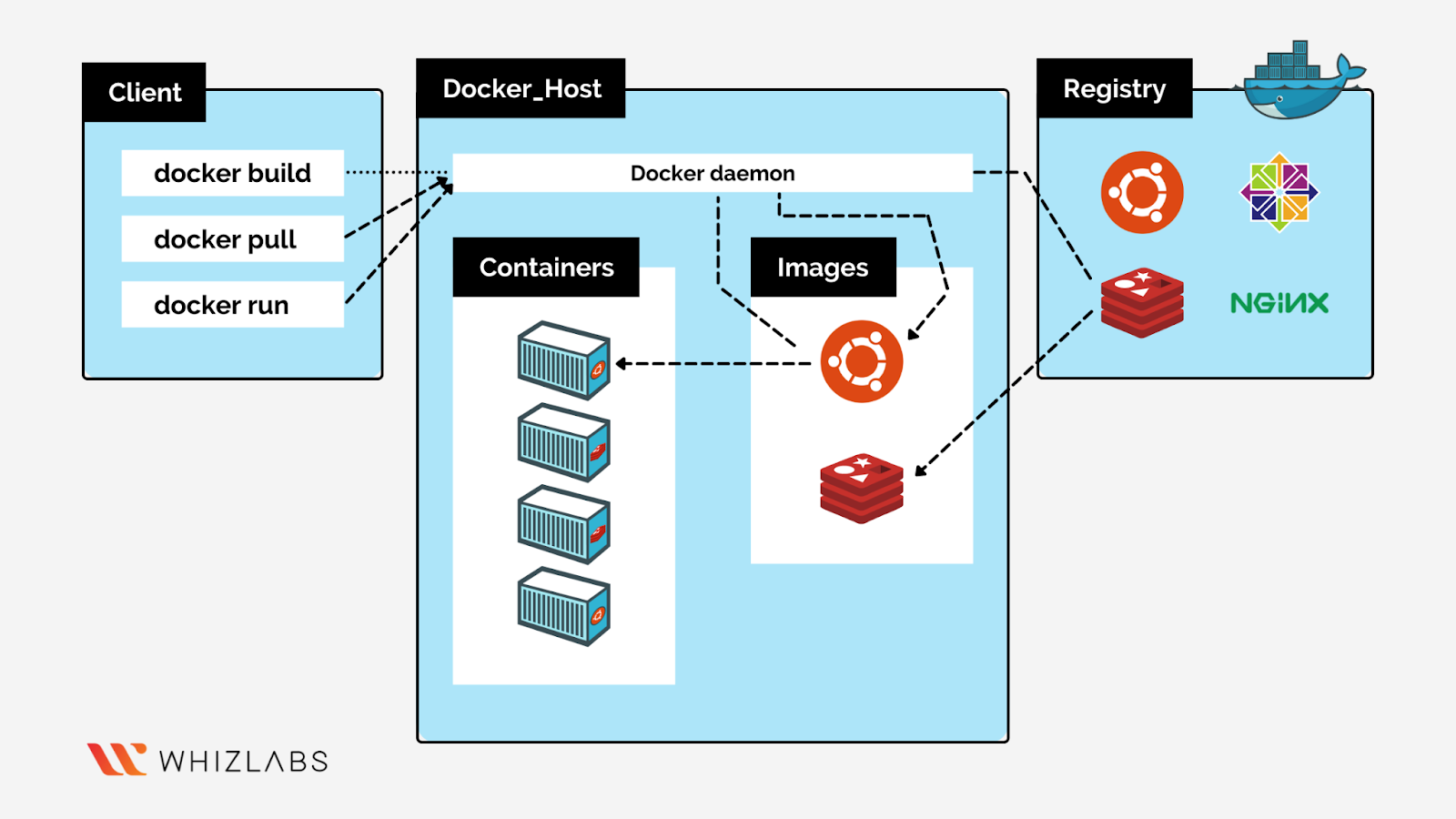
Cloud agnostic solutions are ideal for businesses that want to avoid having cloud vendor lock-in and need to reduce their risk of downtime. This is a concern for organizations as some cloud providers may increase prices; having the flexibility to make a switch when necessary is key.

However, understanding the costs of your cloud environment in context is a necessity. It’s not just about the total investment, but about understanding the value for each business unit.

## Docker, Containers

### **Docker architecture**

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers



### **The Docker daemon**

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

### **The Docker client**

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

### **Docker Desktop**

Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see Docker Desktop.

### **Docker registries**

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

### **Docker objects**

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

### **Images**

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

### **Containers**

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

## Kubernetes

### **What is Kubernetes?**

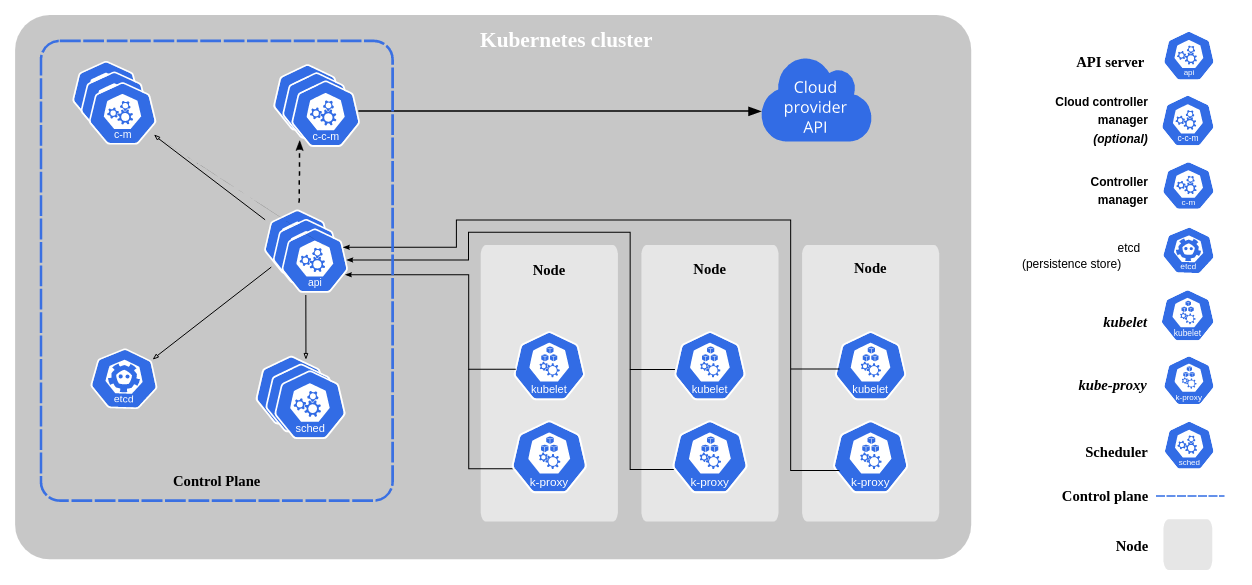
Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation results from counting the eight letters between the "K" and the "s".

### **How Kubernetes works**

When you deploy Kubernetes, you get a cluster.

A Kubernetes cluster consists of a set of worker machines, called [nodes](https://kubernetes.io/docs/concepts/architecture/nodes/), that run containerized applications. Every cluster has at least one worker node.



**Control Plane Components**

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied).

Control plane components can be run on any machine in the cluster. However, for simplicity, set up scripts typically start all control plane components on the same machine, and do not run user containers on this machine.

**kube-apiserver**

The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

The main implementation of a Kubernetes API server is kube-apiserver. kube-apiserver is designed to scale horizontally—that is, it scales by deploying more instances. You can run several instances of kube-apiserver and balance traffic between those instances.

**etcd**

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data. If your Kubernetes cluster uses etcd as its backing store, make sure you have a back up plan for those data. You can find in-depth information about etcd in the official documentation.

**kube-scheduler**

Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.

Factors taken into account for scheduling decisions include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.

**kube-controller-manager**

Control plane component that runs controller processes.

Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

Some types of these controllers are:

* Node controller: Responsible for noticing and responding when nodes go down.
* Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
* EndpointSlice controller: Populates EndpointSlice objects (to provide a link between Services and Pods).
* ServiceAccount controller: Create default ServiceAccounts for new namespaces.

**cloud-controller-manager**

A Kubernetes control plane component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

The following controllers can have cloud provider dependencies:

* Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
* Route controller: For setting up routes in the underlying cloud infrastructure
* Service controller: For creating, updating and deleting cloud provider load balancers

**Node Components**

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

**kubelet**

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

**kube-proxy**

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

**Container runtime**

The container runtime is the software that is responsible for running containers.

Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).

**Addons**

Addons use Kubernetes resources (DaemonSet, Deployment, etc) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace.

Selected addons are described below; for an extended list of available addons, please see Addons.

**DNS**

While the other addons are not strictly required, all Kubernetes clusters should have cluster DNS, as many examples rely on it. Cluster DNS is a DNS server, in addition to the other DNS server(s) in your environment, which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.

**Web UI (Dashboard)**

Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.

**Container Resource Monitoring**

Container Resource Monitoring records generic time-series metrics about containers in a central database, and provides a UI for browsing that data.

**Cluster-level Logging**

A cluster-level logging mechanism is responsible for saving container logs to a central log store with search/browsing interface.

### **Why use Kubernetes?**

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behavior was handled by a system?.

Kubernetes provides you with:

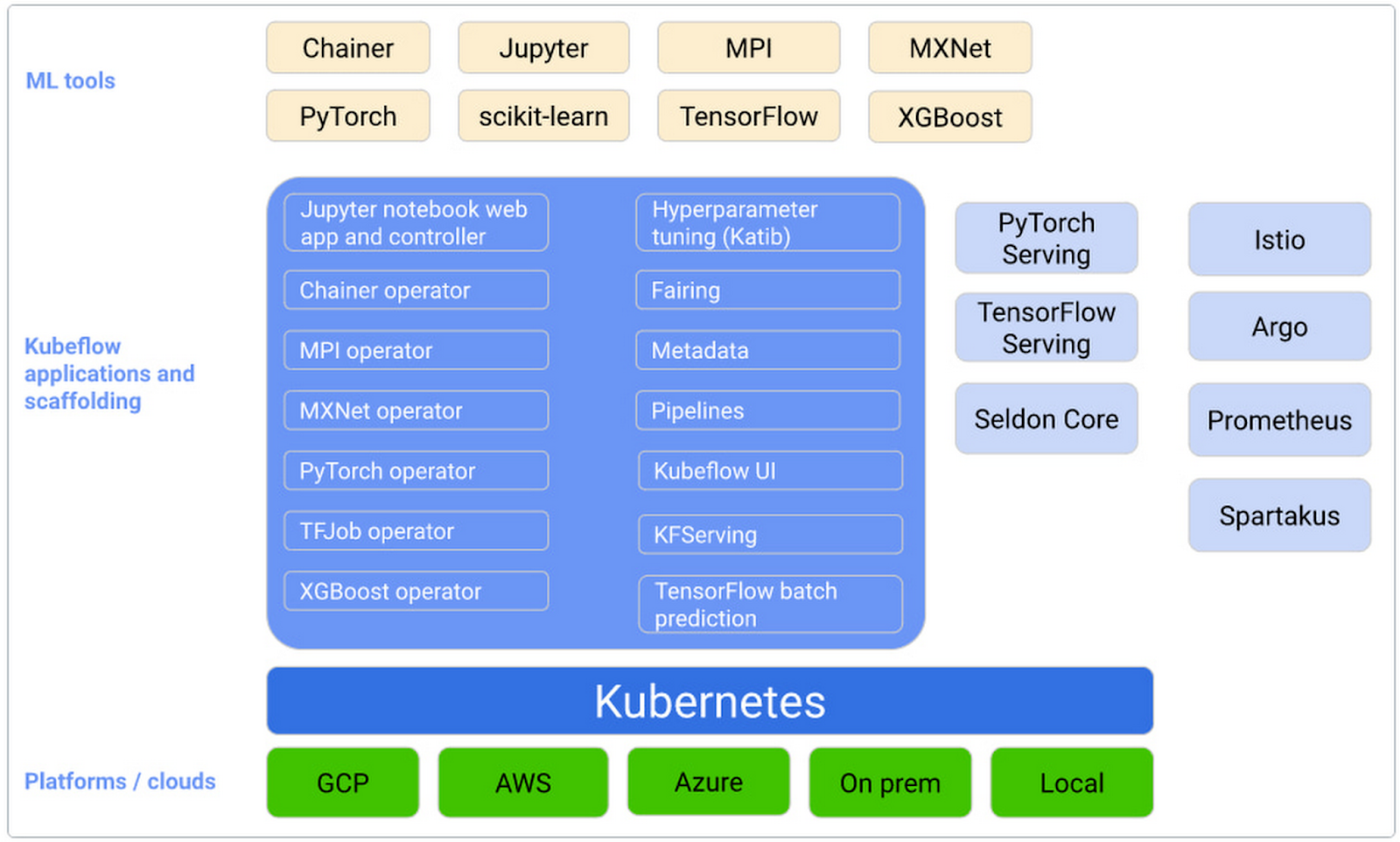
* **Service discovery and load balancing** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.
* **Storage orchestration** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.
* **Automated rollouts and rollbacks** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.
* **Automatic bin packing** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.
* **Self-healing** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
* **Secret and configuration management** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

## Kubeflow

### **What is Kubeflow?**

The Kubeflow project is dedicated to making deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable. Our goal is not to recreate other services, but to provide a straightforward way to deploy best-of-breed open-source systems for ML to diverse infrastructures. Anywhere you are running Kubernetes, you should be able to run Kubeflow

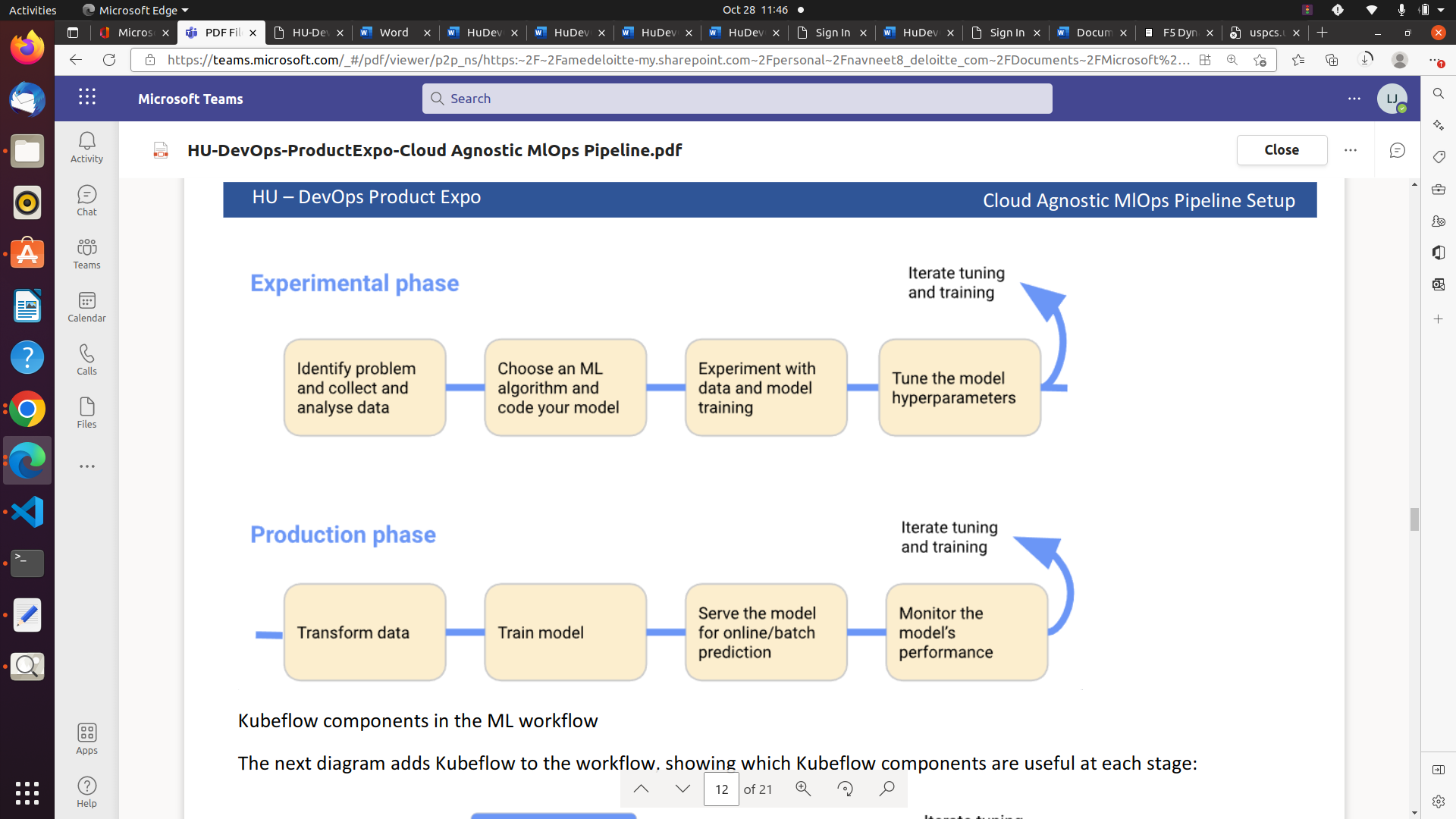
### **Architecture**



### **Introducing the ML workflow**

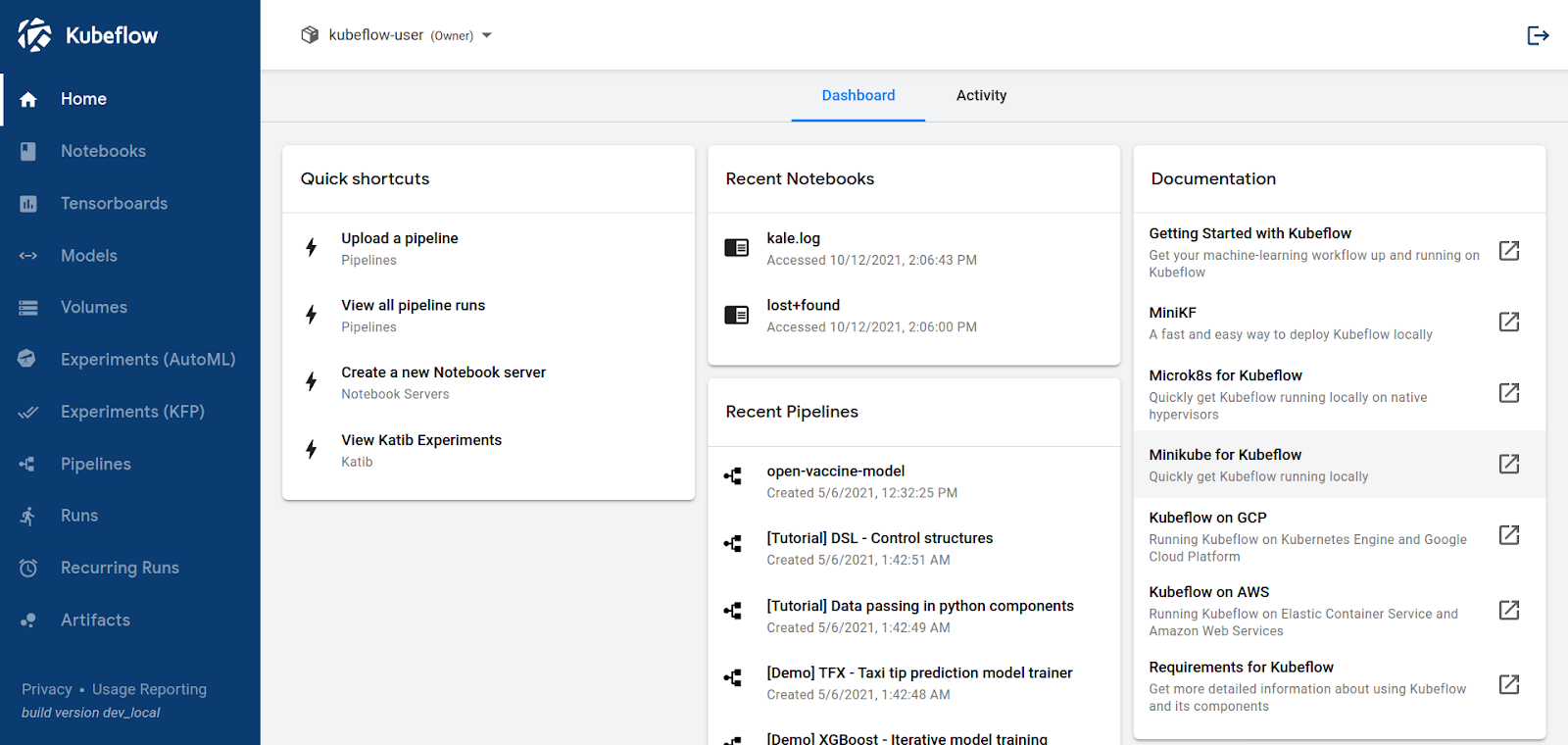
When you develop and deploy an ML system, the ML workflow typically consists of several stages. Developing an ML system is an iterative process. You need to evaluate the output of various stages of the ML workflow, and apply changes to the model and parameters when necessary to ensure the model keeps producing the results you need.

For the sake of simplicity, the following diagram shows the workflow stages in sequence. The arrow at the end of the workflow points back into the flow to indicate the iterative nature of the process:



* In the experimental phase, you develop your model based on initial assumptions, and test and update the model iteratively to produce the results you’re looking for:
* In the production phase, you deploy a system that performs the following processes:

### **Kubeflow interface**



## Git Hub Actions

To automate the process of building a docker image everytime we push anything into the repo. Really helpful in the containerizing task.

# **Tech Stack**

## Kubeflow

## To build and run the pipeline

## Kubernetes

## To host the kubeflow

## Docker and Dockerhub

* To containerize and store the images of different steps in pipeline. Really useful for future use

## Python

* To create ML code and convert it into yaml file. Can be used without converting it into yaml. If you have Kale in your system.

## Github and Github actions

* To host the code of docker image for each step in pipeline
* To automate the process building and pushing the image

## Google drive

* To store the ML data’s.

## Cloud provider

* To create an instance to run Kubernetes.
* Used AWS but any cloud provider will do the trick.

# **System requirement and Configuration**

## Microk8s

* Operating system: Ubuntu 20.04
* Kubernetes version: 1.20
* system specification→
  + Ram:32 GB
  + cpu: 8

## Minikube

* Operating system: Ubuntu 18.04
* Kubernetes version: 1.21.24
* system specification→
  + Ram:32 GB
  + cpu: 8

# **Getting Started**

* Download and run the python program available in this [repo](https://github.com/lijozech-deloitte/hu-devops_projectmonth)
* Upload the created YAML file into kubeflow and create a pipeline using it.
* Create runs of this pipeline

# **Implementation**

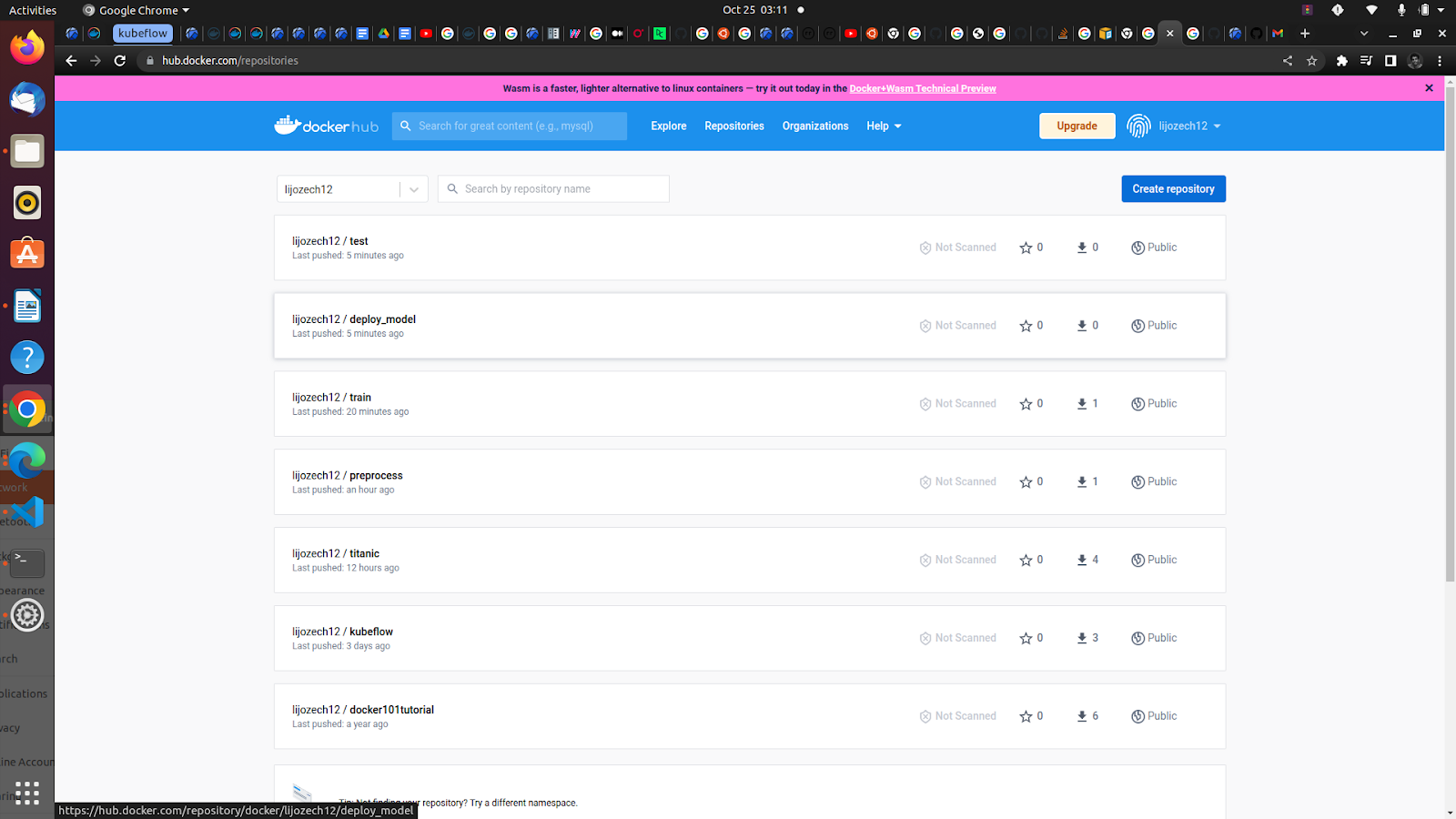
Succesfully run a four step Boston mlops pipeline. It does the following.

* Retrieve data from Python libraries
* Preprocess the data
* Split the data into two parts.
  + Training data(77% of data)
  + Testing data(33% of data)
* Train the model using training data
* Test the mode using testing data
* Deploy the model

## 

### **Things I learned and implemented**

* Containerize each steps of pipeline
* How to automate the process of building and pushing images to docker hub.
* Making components from images
* Passing data between Containers



# **Challenges**

## Installations problems

This issue really severe. I lost more than 1 week to correct this issue. I tried methods.

* Microk8s. The issue is with load balancer and version of microK8s. It worked for the first time. After that I was unable to do it again using same methods
* Mini Kube. Conflicts between K8s version and Kubeflow version. Sometimes running of minikube cluster
* AWS EKS. Not a cloud agnostic approach. But tried to use when both of the first approach didn’t work. Due to the permission issue with deloitte cloud unable to do. Since it needs lot resource to run.
* Personal GCP account. Running the kubeflow pipeline will cost a lot. Kubeflow need lot of memory for that

## Lack of proper Documentations, Courses and Video Tutorials

Since the Kubeflow is a new technology. The Documentations and other study materials are very less. But thank to some blogs that helped me to understand lot of things. Also I overcome some of the issues by trial and error methods. But I always run those experiments on test pipelines. Only if it works I apply those changes to production pipeline

## S3 bucket configuration issues & Storage of data

I faced issue when I was unable to use authorize my AWS account in order to access data stored in S3 bucket. What I done for solution is importing wget Library program and using it for downloading data i stored in google drive/ GitHub. It worked fine.

## Python Dictionary issues

I faced this issue during when working with advanced Pipeline. The needed python dictionaries are not loading into my pipeline. But it worked for another simple pipeline that I created.

## Memory run out issue

Since the Kubeflow pipeline and ML programs in general consumes lot of memory and resources. I faced memory runout issue when I tried to run the more advanced mlops pipeline. The resources are full by the previously created pipelines and functions.

## Kale is not able to install issue

Kale is really helpful to building mlops pipeline. It reduces the pipeline building tasks by significantly. But here due to some of the issues with Kubeflow. I was unable to install kale. This made pipeline building process a task. Since I need to build custom functions and make components out of it. Also I need to create, download and again upload yaml file to run the pipeline. Using kale it would happened with a single touch of button.

## Less support for the cloud agnostic pipeline

It’s harder to build cloud agnostic pipeline using Kubeflow compared to other cloud specific methods. Since K8s cluster managed by cloud providers( ex. EKS ) is more versatile and easier to build and run pipeline, access data stored in the same cloud etc...

# **Outcome**

# I created a cloud agnostic mlops pipeline. This pipeline contains Preprocessing, Training, Testing and Deploying steps. These steps are containerized and stored in docker hub. Every time their change in code this images got updated automatically with the help of GitHub actions. This pipeline can be run in any Kubernetes cluster despite where that cluster is hosted on.

# **Future scope and improvements**

* Building more complex pipeline
* Automating pipeline building steps

# **Bibliography**

<https://minikube.sigs.k8s.io/docs/start/>

<https://docs.docker.com/engine/swarm/secrets/>

<https://charmed-kubeflow.io/docs/install>

<https://microk8s.io/docs/getting-started>

<https://www.kubeflow.org/docs/components/pipelines/sdk/sdk-overview/>

# 