SUSE

Analytics Edge Ecosystem Workloads - Google Summer of Code

Analytics Edge Ecosystem Workloads - Google Summer of Code

Publication Date: 2022-08-22

SUSE LLC
1800 South Novell Place
Provo, UT 84606
USA
https://documentation.suse.com

Contents

https://summerofcode.withgoogle.com/programs/2022/projects/MNFtN4so

1 Project Information

Title	Analytical Edge Ecosystem Workload - **Healthcare**
Author(s)	
Doc Repo URL	https://github.com/abhi-bhatra/ct_image_scanning/tree/UI_base
Initial Draft Date	09 SEP 2022
Target Published Date	12 SEP 2022

2 Introduction

2.1 Motivation

Cancer is world's second-leading cause of death in the world. It results in development of adnormal cells that divide uncontrollably and have the ability to infiltrate and destroy normal body tissue. Eventually, survival rates are improving and all thanks to cancer screening, treatment and prevention.

Machine Learning is one of the aspect which can be integrated with the modern science and can create wonders. Under the Google Summer of Code, we at SUSE organization have created a Machine larning based Cancer Predicition Model for early screening.

This Document will demonstrate the detailed approach undertaken to accomplish this project. The project is developed under mentorship of Bryan Gartner, Ann Davis, Brian Fromme and the SUSE organization.

2.2 Challenges

- **Biased Training Data**: Bias is a common challenge to AI models. You can read more on biasness in AI model here (https://itrexgroup.com/blog/ai-bias-definition-types-examples-debiasing-strategies/)↗. But the good news is we have checked on training datasets on preexisting biases. We have applied various computational methods that can detect and migrate bias in the data.

- **Difficulties associated with Data gathering and managing data**: Healthcare data is stored in heterogeneous and unstructured ways, and it is very challenging to generalize. But the best part is that we have mounted drive to the application and data is arranged in the drive periodically and systematically using Kubernetes Jobs and the CRON scripts.

- **Ethical concerns and considerations**: Even the researchers, scholars and pathologists sometimes can't explain how the model delivers the outcome. As a solution, we have developed explainable AI (https://itrexgroup.com/blog/explainable-ai-principles-classification-examples/)↗ where model can reveal reason behind their decision making.

2.3 Benefits

- **Reducing false positives and negatives**: Using AI in cancer detection will improve the accuracy of diagnosis, reducing false positives and negatives. Google's research says that AI-powered software cut false positives down by 6% and false negatives by 9% (https://www.cbsnews.com/news/breast-cancer-doctors-hope-mammography-tests-will-be-improved-with-new-artificial-intelligence-program/).
- **Helping Radiologists**: Research had given us proof that during an evaluation, models helped radiologists reduce false-positives rates by 37.3% (https://www.nature.com/articles/s41467-021-26023-2).
- **Eliminating Human errors**: A study has shown that AI models can perform with the accurate results and how AI algorithms can detect precancerous lesions in images and can distinguish them from other abnormalities Read about study here. (https://academic.oup.com/jnci/article/111/9/923/5272614)

3 Scope

Artificial Intelligence (AI) or machine learning seems to be bridging the gap between the acquisition of data and their meaningful interpretation into oncology. This application will try to avoid complexities and simplified the Machine Learning system for clinicians and researchers to understand how Machine Learning and its various method like Convolutional Neural Networks (CNNs) are utilized in the field of oncology.

Our application particularly deep learning methods, CNNs and variational auto-encoders, have shown outstanding results. Earlier trained physicians examine the medical images visually for the detection, characterization and monitoring of diseases. AI method will let us automatically recognize the complex patterns in imaging data, providing quantitative as well as qualitative assessments of data within a short period of time.

Using AI in imaging can tell the radiologist about the suspicious scans that need to be looked at first among the huge number of other normal imaging findings. This has a tremendous potential in reducing time, aiding early diagnosis from the time of the mammogram, reducing the need for unnecessary biopsies and the concern of a misdiagnosis. Similar techniques to those described above are being deployed for the evaluation of eye imaging, skin lesions, electrocardiograms, X-rays and cross-sectional imaging such as CT or MRI. Read more about Scope of Machine Learning in Oncology (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7592433/) ↗.

4 Audience

The target audience for Machine Learning based Cancer Prediciton System is the group of people who have been determined to be at risk for the disease. This software needs to be installed on Edge devices at the Hospital premises, where a lab technician, radiologist and an oncologist can closely work with the application.

5 Technical overview

5.1 Overview

Machine Learning is a branch of Artificial Intelligence that implements varieties of optimization and statistical techniques that allows computer to **learn** from past examples and to detect patterns from a large, nosiy or complex data.

As a part of GSoC, We have built a Machine Learning Based **Cancer Predicition System**. The fundamental goal of the system is the prediciton of Cancer Suspectibility (also known as risk assessment), in this case, we are trying to predict the likelihood of developing a cancer prior to occurence of the disease.

This Model uses CT Scanned Images in the form of DICOM (Digital Imaging and Communications in Medicine). Whole application is deployed on Microservice based architecture and is divided into four interfaces: **Lab Technician Dashboard** , **Doctor Dashboard**, **Rancher Dashboard** and **ML Pipeline Dashboard**. They are as follows:

- Lab Technician's Application is responsible for getting DICOM image as input. The person could alter the information such as Contrast, Brightness and Angle of rotation of the DICOM image. They can also read all the information associated with the DICOM image (Modality: CT Scan).
- Doctor Dashboard is more simpler dashboard designed for the doctors to examine the report send by the Lab Technician. Application will also send it's prediction over the chosen image. If doctor will not be satisfied with the response, they can send the image for the retraining with the correct label attached to it.
- Rancher Dashboard provides UI for managing application cluster.
- Kubeflow pipeline UI provides a visualization dashboard for visualizing the pipelines and the workflow logic for retraining of Machine Learning models.

5.2 Components

- **Python**: Python is a high-level programming language. It emphasize on code readability with the use of significant identation. Python is being dynamically-typed and automatic garbage-collector so it is the basic language we will be using to communicate. We have created Machine Learning models on top of Keras, written in Python and Flask application which serves as API and backend is also written in Python. Learn more about Python (https://www.python.org/about/gettingstarted/) ↗.

- **Keras**: Keras is an open-source software library that provides a Python interface for designing artificial neural networks. It acts as an interface for TensorFlow library. We are using keras to design ML backend, especially convolutional neural network (https://www.tensorflow.org/tutorials/images/cnn) ↗.

- **Flask**: Flask is a micro web framework also written in Python. It is lightweight framework used to create web applications easily. Flask is used to serve as Backend and serve Machine Learning model over API. Learn more about Flask here (https://flask.palletsprojects.com/en/2.2.x/tutorial/factory/) ↗.

- **Docker**: Docker is a platform as a service that use OS-level virtualization to deliver software in packages called containers. Docker is used to ship the codes efficiently in optimized way. We are using the Python docker image to build the containers. Dockerfile seems like this (https://github.com/abhi-bhatra/ct_image_scanning/blob/UI_base/lab_tech/Dockerfile) ↗.

- **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. We have a kubernetes manifests designed to set up application over the cluster. Our application is compatible with various kubernetes distributions k3s (https://rancher.com/docs/k3s/latest/en/) ↗, RKE (https://rancher.com/products/rke) ↗, RKE2 (https://docs.rke2.io/) ↗ and other variants.

- **Rancher**: Rancher is an open source software platform that enables organizations to run containers in production. With Rancher, organizations no longer have to build a container services platform from scratch using a distinct set of open source technologies. We are using rancher to manage the kubernetes cluster (https://rancher.com/why-rancher) ↗.

- **Kubeflow**: It is used for machine learning pipelines to orchestrate complicated workflows running on Kubernetes. Kubeflow allows our project to focus on writing ML algorithms instead of managing their operations. To know more, visit the official website of Kubeflow (https://www.kubeflow.org/docs/components/pipelines/installation/localcluster-deployment/) ↗.
- **Longhorn**: Longhorn is cloud-native distributed block storage for Kubernetes that is easy to deploy and upgrade, 100 percent open source and persistent.. It is used as a CSI, used as a storageclass and mounted as a volume within the pods to share the data and information locally. Visit Here (https://longhorn.io/docs/1.3.1/deploy/install/) ↗

Every component just fits in together. Application interaction language is Python. Tensorflow and Flask both are used on top of Python. Convolutional Neural Network is used to design the Cancer prediction model, which fits in to predict the Cancer. Kubeflow is integrated as a retraining logic which allows to orchestrate workflows running on our Kubernetes cluster.

5.3 Component Architecture

5.3.1 Architecture Diagram

5.3.2 Workflow

DICOM Image is transferred from CT Scan Machine to the Lab Technician Application serving on local network on `port1`. Lab Technician's Application is responsible for getting DICOM image as input. The person could alter the information such as Contrast, Brightness and Angle of rotation of the DICOM image. They can also read all the information associated with the DICOM image (Modality: CT Scan). This application is also responsible for predicting the Body part examined by the Machine Learning Model integrated within this microservice.

After the satisfied resutls, lab technician can click on `Send report` button, this will trigger a script which transfer the data to Doctor dashboard via. Persistent Volume mounted as volume at both the applications.

Doctor Dashboard is designed for the doctors to examine the report send by the Lab Technician. It receives the report of a patient and displays it to the user, predicting whether or not person is suffering from cacner. If doctor will not be satisfied with the response, they can send the image for the retraining with the correct label attached to it.

For retraining, a script will be triggered at the backend, which runs a Kuberenetes Job to train the image again and create a newly trained Model using Kubeflow Pipelines in the backend. We can visualize the pipelines using Kubeflow UI which can be accessed through Rancher portal.

6 Prerequisites

This project leverages the Edge to Core to Cloud Computing native solutions. We are building an Analytical Edge Ecosystem Workload so it is recommended to have a basic knowledge of Python, Kubernetes, Rancher and Linux.

1. You can learn the basics of Kubernetes (https://rancher.com/learn-the-basics) ↗.
2. Go for a SUSE guide to Computing in Cloud Native World (https://more.suse.com/global-ebook-edge-computing-cloud-native-world.html) ↗
3. Having a basic knowledge of Rancher will also help in understanding the cluster and edge computing. Follow this SUSE Rancher Learning Path (https://links.imagerelay.com/cdn/3404/ql/651586f0b1df4b22b39c24a5843ed909/SUSE_rancher_learning_path.pdf) ↗.

7 Installation

7.1 Installing KVM Host

7.1.1 Pre-Installment Requirements

Install `virt-install`. `virt-install` is a command line tool that helps you create new virtual machines using libvert library. It might be a complex command with lots of switches but it is very useful when we need to automate the process of creating virtul machines. There are mutliple ways to install **libvirt** (https://libvirt.org/docs.html) and **qemu** (https://www.qemu.org/)

Installation guides for libvirt can be found here (https://libvirt.org/downloads.html) and QEMU can be installed from here (https://www.qemu.org/download/).

7.1.2 Download ISO file

You can get ISO file for any OS you prefer to work with over internet. I will be using SUSE Linux Enterprise Server SLE-15 (https://www.suse.com/download/sles/)

```
cp SLE-15-SP3-Full-x86_64-GM-Media1.iso /var/lib/libvirt/images/
```

7.1.3 Run virt-install

Script should resemble to something like this:

```
os="--os-type=linux --os-variant=SLE-15"
location="--location=/var/lib/libvirt/images/SLE-15-SP3-Full-x86_64-GM-Media1.iso"
cpu="--vcpus 2"
ram="--ram 2048"
name="sle15"
disk="--disk /dev/mapper/SLE-15-SP3,size=40"
type="--virt-type qemu"
network="--network network=default"
graphics="--graphics none"
```

Run the below command:

```
virt-install $os $network $disk $location $cpu $ram $type $disk $graphics --name=$name
```

The command options are as follows:

os = "--os-type = linux --os-variant = SLE-15" — Some of these commands have main options, as well as sub options.

location = "--location = /var/lib/libvirt/images/SLE-15-SP3-Full-x86_64-GM-Media1.iso" — This is where you've copied the ISO image file containing the OS you want to install.

cpu = "--vcpus 2" — The CPU command-line option enables you to specify the number of vCPUs assigned to the VM. In this example, I'm assigning two vCPUs.

ram = "--ram 2048" — The RAM command-line option enables you to specify the amount of memory assigned to the VM.

name = "sle15" — The name command-line option enables you to assign a name to the VM.

disk = "--disk /dev/mapper/SLE-15-SP3,size = 40" — This is where the VM will be installed and the size, in gigabytes, to be allocated. This must be a disk partition and not a mount point. Type df -h to list disk partitions.

type = "--virt-type qemu" — The type command-line enables you to choose the type of VM you want to install. You can use KVM, QEMU, Xen or KQEMU. Type virsh capabilities to list all of the options. In this example, I'm using QEMU.

network = "--network network = default" — Use network = default to set up bridge networking using the default bridge device. This is the easiest method, but there are other options.

graphics = "--graphics none" — The graphics command-line option specifies that no graphical VNC or SPICE interface should be created. Use this for a kickstart installation or if you want to use a ttyS0 serial connection.

7.1.4 Edit the Network configuration

Login to the newly created KVM, and we will install minimal requirements in our new KVM. I have used OpenSUSE Linux 15:

```
zypper ref
zypper in -y open-iscsi kernel-default e2fsprogs xfsprogs
zypper in -y docker
systemctl enable --now iscsid
systemctl enable --now docker
```

For managing the network we will create network configurations as well.

```
cd /etc/sysconfig/network
```

`cp ifcfg-eth1 ifcfg-eth0`

`vi ifcfg-eth* routes`

```
// change ifcfg-eth0
STARTMODE=auto
BOOTPROTO=static
IPADDR=172.16.220.x/24


// change ifcfg-eth1
STARTMODE=auto
BOOTPROTO=dhcp
DHCLIENT_SET_DEFAULT_ROUTE=no


// create routes
default 172.16.220.1 - -
```

Restart the network service `systemctl restart network`

Validate the network settings

`ip a`

`ip r`

`hostname -f`

`systemctl status firewalld`

7.2 Install Kubernetes Cluster

Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management. We will see how to deploy various distributions of Kubernetes on the KVM. You can run any of the cluster you are comfortable to work with.

7.2.1 Pre-Installment Requirements

It is recommended to install **kubectl** in advance so as to interact with the cluster. Kubectl or Kubernetes command-line tool allows you to run commands against Kubernetes clusters. You can Install kubectl on variety of Linux platforms, macOS and Windows. Find the documentation for your preffered OS below:

- Install on Linux (https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/)
- Install on MacOS (https://kubernetes.io/docs/tasks/tools/install-kubectl-macos/)
- Install on Windows (https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/)

It is also recommeded to install docker in your system. Although Docker is not required for k3s, but RKE clusters need Docker to be installed on the system.

I am using SUSE Linux, so I can install Docker using: `zypper in -y docker`

You can find documentation to install Docker on other Operating systems here (https://www.docker.com/get-started/).

7.2.2 Installing K3S

We will be watching on How to install k3s cluster. Although it will be very easy to create a K3S cluster. As k3s is a highly available, certified Kubernetes distribution designed for production workloads with <50MB binary that reduces the dependencies and steps needed to install, run and auto-update.

The simplest form of running k3s is as follows:

`curl -sfL https://get.k3s.io | sh -`

You can find more options as environment variable that can be used to configure the Installation (https://rancher.com/docs/k3s/latest/en/installation/install-options/).

Change the path so as to access the cluster:

```
mkdir ~/.kube/
sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
chmod 644 ~/.kube/config

// ensure that the application is accesible
kubectl get nodes
```

7.2.3 Installing RKE

RKE is a fast, versatile Kubernetes installer that you can use to install Kubernetes on your Linux hosts. You can get started in a couple of quick and easy steps:

- Download the binary file from here (https://github.com/rancher/rke/#latest-release).
- Rename the rke binary to the `mv rke_linux-amd64 rke`
- Make the RKE binary that you just downloaded executable: `chmod +x rke`
- Test the installation: `rke --version`

Now there are two ways to write cluster configuration file, also called `cluster.yaml` to determine what nodes will be in the cluster and how to deploy Kubernetes, and use `rke config` command. To run the RKE cluster, use the command: `rke up`

Detailed documentation on cluster installation will be found here (https://rancher.com/docs/rke/latest/en/installation/RKE).

7.3 Installing Rancher

With Rancher, you can unify the clusters to ensure consistent operations, workload management, and enterprise-grade security. Now, move from Core to Cloud to Edge with Rancher.

There are two ways to install the Rancher.

7.3.1 Method 1

The most easy way is to Install Rancher as a Docker container and import the existing cluster on the Rancher portal.

a. Install Rancher as a Docker Image and run it: `sudo docker run --privileged -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher`

b. Now you will be able to access the Rancher dashboard by accesing the URL `https://localhost` or if you are using VM, then access it on your VM's IP: `https://VM:IP`

c. Follow the instructions shown on the Dashboard to login to your Rancher portal. Username will be **admin**, and you need to generate a password from Bootstrap password (Instructions will guide you for the same)

d. Import the existing cluster by clicking on `Import Cluster` button.

e. Select the Provider where your cluster is up and Running. We are going to use the cluster, so we use **Generic**

f. Give cluster a name and a description (optional) and Click on next

g. Some commands will appear, run those commands in your local cluster you set up earlier to import those clusters to Rancher. Command should look like this: `kubectl apply -f https://<server-ip>/v3/import/42ql8klfghhgv7z-plr2mwtqm4gvpn6t766g4gmjnzzsfztzbq64wmb_c-m-8rdkjd4k.yaml`

h. Or if certificate errors arise, you can use the second command, looks like this: `curl – insecure -sfL https://172.16.220.83:4431/v3/import/42ql8klfghhgv7z-plr2mwtqm4gvpn6t766g4gmjnzzsfztzbq64wmb_c-m-8rdkjd4k.yaml | kubectl apply -f -`

i. After running those command return to Homepage and you can see the clusters are registered on Rancher.

You can read the detailed overview on How to Install and Deploy Workload on the cluster imported in Rancher (https://medium.com/@abhinavsharma332/deploying-wordpress-over-rancher-cb9539b1d7da).

7.3.2 Method 2

Second method is to install Rancher using the Manifests directly into your cluster.

a. Add the Rancher Helm chart: `helm repo add rancher-CHART_REPO https://releases.rancher.com/server-charts/CHART_REPO` (Find the Stable version here (https://docs.ranchermanager.rancher.io/v2.5/reference-guides/installation-references/helm-chart-options#helm-chart-repositories))

b. Create a namespace: `kubectl create namespace cattle-system`

c. Choose the SSL configuration: The Rancher management server is designed to be secure by default and requires SSL/TLS configuration. There are three recommended options for the source of the certificate used for TLS termination at the Rancher server: Rancher-generated TLS certificate, Let's Encrypt and Bring your own certificate (https://docs.ranchermanager.rancher.io/v2.5/pages-for-subheaders/install-upgrade-on-a-kubernetes-cluster#3-choose-your-ssl-configuration).

d. Install Cert Manager:

```
# If you have installed the CRDs manually instead of with the `--set installCRDs=true`
 option added to your Helm install command, you should upgrade your CRD resources before
 upgrading the Helm chart:
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.1/cert-
manager.crds.yaml

# Add the Jetstack Helm repository
helm repo add jetstack https://charts.jetstack.io

# Update your local Helm chart repository cache
helm repo update

# Install the cert-manager Helm chart
helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --create-namespace \
  --version v1.5.1
```

a. Verify the Installation: `kubectl get pods --namespace cert-manager`

b. Install Rancher:

```
helm install rancher rancher-<CHART_REPO>/rancher \
  --namespace cattle-system \
  --set hostname=rancher.my.org \
  --set replicas=3

  # --version 2.3.6 can be used
```

a. Wait for Rancher to be rolled out: `kubectl -n cattle-system rollout status deploy/rancher`

To know more installation of Rancher, visit the official Rancher Installation Guide (https://docs.ranchermanager.rancher.io/pages-for-subheaders/install-upgrade-on-a-kubernetes-cluster).

7.4 Installing Longhorn

Longhorn is an official CNCF project, when combined with Rancher, Longhorn makes the deployment of highly available persistent block storage in your Kubernetes environment easy, fast and reliable.

There are 3 ways to installing Longhorn to Clusters:

a. Using the Apps and Marketplace in Rancher UI (https://longhorn.io/docs/1.3.1/deploy/install/install-with-rancher/)

b. Using the kubectl manifests files (https://longhorn.io/docs/1.3.1/deploy/install/install-with-kubectl/)

   - `kubectl apply -f https://raw.githubusercontent.com/longhorn/longhorn/v1.2.4/deploy/longhorn.yaml`

c. Using the Helm (https://longhorn.io/docs/1.3.1/deploy/install/install-with-helm/):

   - Add Longhorn Helm repository: `helm repo add longhorn https://charts.longhorn.io`
   - `helm repo update`
   - Install the helm chart: `helm install longhorn/longhorn -name longhorn -namespace longhorn-system`
   - Access the UI by going to change: **LoadBalancer** to **ClusterIP**

7.5 Installing Kubeflow

The Kubeflow project is designed for making deployments of machine learning (ML) workflows on Kubernetes. It provides 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.

We can install various components of Kubeflow such as:

1. Kubeflow Central Dashboard: Central dashboard provides quick access to the Kubeflow components deployed in your cluster.
2. Kubeflow Notebooks: Kubeflow includes services to create and manage interactive Jupyter notebooks.
3. Kubeflow Pipelines: Kubeflow Pipelines is a platform for building and deploying portable, scalable machine learning (ML) workflows based on Docker containers.

Learn more about the Kubeflow components and there installation (https://www.kubeflow.org/docs/components/)↗.

7.5.1 Deploying Kubeflow Pipelines

We will look at how to deploy Kubeflow Pipelines standalone on our local clusters.

Now, to deploy Kubeflow Pipelines run the following commands:

```
export PIPELINE_VERSION=1.8.3

kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize/cluster-scoped-
resources?ref=$PIPELINE_VERSION"

kubectl wait --for condition=established --timeout=60s crd/applications.app.k8s.io

kubectl apply -k "github.com/kubeflow/pipelines/manifests/kustomize/env/platform-
agnostic-pns?ref=$PIPELINE_VERSION"
```

It will take 15–20 mins to deploy the Kubeflow Pipelines on your cluster. You can check the status using `kubectl get all -n kubeflow`

Once all the services will start, you can see all pods status 1/1 Running. Your output will be somewhat similar to this:

```
NAME                                                READY   STATUS
 RESTARTS   AGE
pod/workflow-controller-5667759dd7-fbgrp            1/1     Running            0
    2d3h
pod/ml-pipeline-scheduledworkflow-7f8bc78db9-qpx4f  1/1     Running            0
    2d3h
pod/ml-pipeline-viewer-crd-8497d9695c-tqmdg         1/1     Running            0
    2d3h
```

```
pod/ml-pipeline-ui-69bc756bd7-nmzm6                  1/1     Running           0
    2d3h
pod/metadata-envoy-deployment-6df8bdd989-lc77p       1/1     Running           0
    2d3h
pod/minio-5b65df66c9-qt6lk                           1/1     Running           0
    2d3h
pod/ml-pipeline-persistenceagent-585c4b58d6-mcmtx    1/1     Running           1
    2d3h
pod/ml-pipeline-7cc4f8fdf7-b2vjp                     1/1     Running           2
    2d3h
pod/cache-server-6cddbbc849-bnd6n                    1/1     Running           1
    2d3h
```

Now you can access the Kubeflow Pipeline UI using port-forwarding: `kubectl port-forward -n kubeflow svc/ml-pipeline-ui 8080:80`

We can access the portal using `http://localhost:8080` or we can also access on our cluster IP using `http://VM_IP:8080`

7.6 Application Setup

7.6.1 Flask Interface

Complete Cancer Prediction System is built on top of Flask. It has two separate applications for the doctor and the radiologist. Directory Structure of the application is as follows:

```
/application
-- doctor_app/
   -- app.py
   -- Dockerfile
   -- classification-model.h5
   -- prediction-model.h5
   -- requirements.txt
   -- static/
      -- styles/
         -- css/
         -- js/
   -- template/
      -- base.html
      -- gallery.html
      -- predict.html
      -- retrain.html
      -- upload.html

-- lab_tech/
   -- app.py
   -- Dockerfile
   -- classification-model.h5
   -- adjust.py
   -- requirements.txt
   -- static/
      -- styles/
         -- css/
         -- js/
   -- template/
      -- base.html
      -- predict.html
      -- send.html
      -- upload.html
```

To work with the above application locally:

1. Clone the GitHub Repository: `git clone https://github.com/abhi-bhatra/ct_image_scanning`
2. Browse to application directory: `cd application/`
3. Let's run the **Lab Technician Application**, Use `cd lab-tech`:
    - Application is built on top of Python, so we will install the requirements: `python -m pip install requirements.txt`
    - Set the Debug on, if you want to live Debug the Flask Application: `export DEBUG=1`
    - Run the application on Port 5001: `flask run -p 5001`
    - Application workflow
        - **app.py**: This is the core of Flask application. All the Machine Learning Prediction codes resides in this file
        - **Dockerfile**: docker image of the Flask Application.

        - **templates**: In this directory, complete application frontend resides.
        - **static**: This directory serves all the static content to the application, like CSS, JS or static Images.

4. Now, Let's run the **Doctor Dashboard**, Use `cd doctor-app`:
    - Install the requirements: `python -m pip install requirements.txt`
    - Set the Debug on, if you want to live Debug the Flask Application: `export DEBUG=1`
    - Run the application on Port 5002: `flask run -p 5002`
    - Application Workflow
        - **app.py**: This is the core of Flask application. It displays the repors send by the Lab Technicians Application in a palette. Prediciton API and Routes have been defined in this file.
        - **Dockerfile**: Docker configuration of the Flask Application. It is pretty much similar to the Lab Technician application, but with minor dependencies added.
        - **templates**: In this directory, complete application frontend resides.
        - **static**: This directory serves all the static content to the application, like CSS, JS or static Images.
        - **classification-model.h5**: This is the trained Machine Learning model output file, which is imported in the `app.py` and classifies whether the DICOM belongs to chest or any other part of the body.
        - **prediction-model.h5**: This is the trained Machine Learning model output file, imported in `app.py` and predict whether the patient is suffering from cancer or not with the probabilty of prediction.

7.6.2 Dataset

The dataset is designed to allow for different methods to be tested for examining the trends in CT image data associated with using contrast and patient age. The basic idea is to identify image textures, statistical patterns and features correlating strongly with these traits and possibly build simple tools for automatically classifying these images when they have been misclassified (or finding outliers which could be suspicious cases, bad measurements, or poorly calibrated machines).

Dataset is being imported from **"Kaggle CT Medical Images ~ CT images from cancer imaging archive with contrast and patient age" (https://www.kaggle.com/datasets/kmader/siim-medical-images)** ↗

Dataset is managed within the application, with the following Directory Structure:

```
/dataset
-- archive/
   -- dicom_dir/
   .
   ID_0001_AGE_0069_CONTRAST_1_CT.dcm
   .
   -- tiff_images/
   .
   ID_0000_AGE_0060_CONTRAST_1_CT.tif
   .
   -- full_archive.npz
   -- overview.csv

-- dataset-classification
   -- Chest-CT/
   -- NonChest-CT/

-- dataset-prediction/
   -- train/
      -- cancer/
      -- non-cancer/
   -- test/
      -- cancer/
      -- non-cancer/
   -- validation/
      -- cancer/
      -- non-cancer/
```

1. **archive**: This folder comprises of raw dataset downloaded from Kaggle. We use python notebooks to process the data for further used in Machine Learning model.
2. **dataset-classification**: This is a separate dataset which separates all the DICOM Images as Chest and Non Chest. Currentyl, our model support Cancer classification on Chest DICOM Images. So, we need to filter our the Non Chest DICOM Images.
3. **dataset-prediciton**: This is the final dataset used in Machine Learning model. All the ras images are processed into Train, Test and Validation sets. The labels are attached to the DICOM, so images can be classified as Cancer and Non-Cancer Images.

7.6.3 Data Cleaning and Visualization

It is equally important to have the right data that fits in with your model to get better, more accurate and more optimized results. We have a raw data downloaded from Public Datasets, now in order to process this data we have created some Python Scripts and Notebooks.

Data Visualization

Data visualization is the representation of data through use of common graphics, such as charts, plots, infographics, and even animations. These visual displays of information communicate complex data relationships and data-driven insights in a way that is easy to understand.

Let us understand the data, we are using:

Import the python modules that we are going to use

```
import numpy as np
import pandas as pd
from skimage.io import imread
import seaborn as sns
import matplotlib.pyplot as plt
from glob import glob
import pydicom as dicom
import os
```

Specify the path of `archive` *directory*

```
PATH="archive/"
data_df = pd.read_csv(os.path.join(PATH,"overview.csv"))
print("CT Medical images -  rows:",data_df.shape[0]," columns:", data_df.shape[1])
data_df.head()
```

Process the dataset

```
def process_data(path):
    data = pd.DataFrame([{'path': filepath} for filepath in glob(PATH+path)])
    data['file'] = data['path'].map(os.path.basename)
    data['ID'] = data['file'].map(lambda x: str(x.split('_')[1]))
    data['Age'] = data['file'].map(lambda x: int(x.split('_')[3]))
    data['Contrast'] = data['file'].map(lambda x: bool(int(x.split('_')[5])))
    data['Modality'] = data['file'].map(lambda x: str(x.split('_')[6].split('.')[-2]))
    return data

tiff_data = pd.DataFrame([{'path': filepath} for filepath in glob(PATH+'tiff_images/
*.tif')])
tiff_data = process_data('tiff_images/*.tif')
dicom_data = process_data('dicom_dir/*.dcm')
```

Let us now count the observations in each category, they show the mean of a quantitative variable among observations in each category

```
def countplot_comparison(feature):
    fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize = (16, 4))
    s1 = sns.countplot(data_df[feature], ax=ax1)
    s1.set_title("Overview data")
    s2 = sns.countplot(tiff_data[feature], ax=ax2)
    s2.set_title("Tiff files data")
    s3 = sns.countplot(dicom_data[feature], ax=ax3)
    s3.set_title("Dicom files data")
    plt.show()

countplot_comparison('Contrast')
```

Examine the DICOM Images

```
def show_images(data, dim=16, imtype='TIFF'):
    img_data = list(data[:dim].T.to_dict().values())
    f, ax = plt.subplots(4,4, figsize=(16,20))
    for i,data_row in enumerate(img_data):
        if(imtype=='TIFF'):
            data_row_img = imread(data_row['path'])
        elif(imtype=='DICOM'):
            data_row_img = dicom.read_file(data_row['path'])
        if(imtype=='TIFF'):
            ax[i//4, i%4].matshow(data_row_img,cmap='gray')
        elif(imtype=='DICOM'):
            ax[i//4, i%4].imshow(data_row_img.pixel_array, cmap=plt.cm.bone)
        ax[i//4, i%4].axis('off')
        ax[i//4, i%4].set_title('Modality: {Modality} Age: {Age}\nSlice: {ID} Contrast:
 {Contrast}'.format(**data_row))
    plt.show()
```

```
show_images(tiff_data,16,'TIFF')
```

Let's just extract the voxel data and combine the slices

```
def extract_voxel_data(list_of_dicom_files):
    datasets = [dicom.read_file(f) for f in list_of_dicom_files]
    try:
        voxel_ndarray, ijk_to_xyz = dicom_numpy.combine_slices(datasets)
    except dicom_numpy.DicomImportException as e:
        raise
    return voxel_ndarray

show_images(dicom_data,16,'DICOM')
```

We can also read the metadata attached to the DICOM image (metadata: shows the background information related to the image like modality, patient's age, patient's sex, etc.)

```
dicom_file_path = list(dicom_data[:1].T.to_dict().values())[0]['path']
dicom_file_dataset = dicom.read_file(dicom_file_path)
dicom_file_dataset
```

Data Cleaning Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled.

Let us now do some manipulations over the data before putting it in the Machine Learning model:

Mention the path of the dataset and import the modules

```
import pydicom
import numpy as np
from PIL import Image
import os
import re

PATH="archive/"
```

Read the DICOM and the Metadata

```
def read_dicom(img_path):
    ds = dicom.dcmread(img_path)
    parameters=[]
    for i in ds:
        parameters.append(str(i))
    new_para=[]
    for i in parameters:
        new_para.append(i[13:])
    dict_item = {re.sub(' +', ' ', i[:35]):re.sub(' +', ' ', i[36:]) for i in new_para}
    return dict_item

// Test on Sample Image
new_ls=read_dicom('archive\dicom_dir\ID_0001_AGE_0069_CONTRAST_1_CT.dcm')
for key, value in new_ls.items():
    print(key, value)
```

Output should be something like this

```
// Trimmed output

Group Length  UL: 524296
Specific Character Set  CS: 'ISO_IR 100'
Image Type  CS: ['ORIGINAL', 'PRIMARY', 'AXIAL']
SOP Class UID  UI: CT Image Storage
SOP Instance UID  UI: 1.3.6.1.4.1.14519.5.2.1.7777.9002.184912220734460823585918206046
Study Date  DA: '19820630'
Series Date  DA: '19820630'
Acquisition Date  DA: '19820630'
Content Date  DA: '19820630'
Study Time  TM: '134257.000000'
Series Time  TM: '135135.242000'
Acquisition Time  TM: '135311.581000'
Content Time  TM: '135259.355000'
Data Set Type  US: 0
Data Set Subtype  LO: 'IMA SPI'
Accession Number  SH: '2819497684894126'
Modality  CS: 'CT'
Manufacturer  LO: 'SIEMENS'
Referring Physician's Name  PN: ''
Station Name  SH: ''
Manufacturer's Model Name  LO: 'SOMATOM PLUS 4'
Private Creator  UN: b'\x14\x00\x00\x00'
Private tag data  LO: 'SIEMENS MED'
Patient's Name  PN: 'TCGA-17-Z011'
Patient ID  LO: 'TCGA-17-Z011'
Patient's Birth Date  DA: ''
Patient's Sex  CS: 'M'
Patient's Age  AS: '069Y'
.
.
```

create a function for conversion

```
def dicom_conversion(dicom_dir):
    for filename in os.listdir(dicom_dir):
        if filename.endswith(".dcm"):
            ds = pydicom.dcmread(dicom_dir + '\\' + filename)
            new_image = ds.pixel_array.astype(float)
            scaled_image = (np.maximum(new_image, 0) / new_image.max()) * 255.0
            scaled_image = np.uint8(scaled_image)
            final_image = Image.fromarray(scaled_image)
            final_image.save('dataset-prediction\\' + filename[:-4] + '.png')
            print(filename)

dicom_to_jpeg(os.path.join("archive", "dicom_dir/"))
```

More details on Data Cleaning and Visualization can be found on Jupyter Notebook in the official GitHub Repository.

7.6.4 Machine Learning Approach

In the project, Machine Learning is one of the major component used for predicition. Further classifies Machine Learning into Deep Learning. We are using CNN. **Convolutional Neural Network (CNN)** is a deep learning method and has achieved better results in detecting and segmenting specific objects in images in the last decade than conventional models such as regression, support vector machines or artificial neural networks.

In our Cancer Prediction System, we have created two Machine Learning Models:

Body Part Classification Model Classification models are a subset of supervised machine learning. A classification model reads some input and generates an output that classifies the input into some category. In our case, model is taking CT-Scan and X-Ray images as input, and images are labelled. The model is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data.

Import model and include path

```
import os
from os.path import exists

import tensorflow as tf

img_shape = (512,512,3)
BATCH_SIZE = 1
IMG_SIZE = (512, 512)

PATH="dataset-classification/"
train_dir=os.path.join(PATH, 'train')
```

```
validation_dir=os.path.join(PATH, 'validation')
```

Load your data in the tf. data. Dataset format

```
train_dataset = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)

validation_dataset = tf.keras.utils.image_dataset_from_directory(
    validation_dir,
    shuffle=True,
    batch_size=BATCH_SIZE,
    image_size=IMG_SIZE
)
```

Build the model

```
base_model = tf.keras.applications.VGG19(input_shape=img_shape, include_top=False,
 weights='imagenet')
base_model.trainable = False
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()(base_model.output)
prediction_layer = tf.keras.layers.Dense(units=1, activation='sigmoid')
(global_average_layer)

model = tf.keras.models.Model(inputs=base_model.input, outputs=prediction_layer)
model.summary()
```

Create Optimizer and loss = "Binary crossentropy"

```
opt = tf.keras.optimizers.RMSprop(learning_rate=0.0001)
model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
```

Fit the dataset on model

```
history = model.fit( train_dataset, batch_size=100, epochs=50 )

// Save the model
model.save('image-model.h5')
```

Evaluate and Predict

```
model.evaluate(validation_dataset)

model.predict(validation_dataset)
```

Cancer Prediction Model In this model, we use machine learning in cancer diagnosis and detection. We are using Artificial neural networks (ANNs) for detecting and classifying tumors CRT images. Let us now see the implementation of the model:

Import the necessary Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from glob import glob
import re
from skimage.io import imread
import keras
```

Ingest the Dataset in the Notebook

```
BASE_IMG_PATH='archive/'
path= os.path.join(BASE_IMG_PATH,'overview.csv')
overview = pd.read_csv(path, index_col=0)
```

Contrast value will be the target value, so we will be transforming the target Parameter in 0s and 1s

```
overview['Contrast'] = overview['Contrast'].map(lambda x: 1 if x else 0)
g = sns.FacetGrid(overview, col="Contrast", size=8)
g = g.map(sns.distplot, "Age")
g = sns.FacetGrid(overview, hue="Contrast",size=6, legend_out=True)
g = g.map(sns.distplot, "Age").add_legend()
```

Reading the sample DICOM image

```
j_imread = lambda x: np.expand_dims(imread(x)[::2,::2],0)
test_image = j_imread(all_images_list[0])
plt.imshow(test_image[0])
```

Test the contrast and compile the Image

```
check_contrast = re.compile(r'/tiff_images\
\ID_([\d]+)_AGE_[\d]+_CONTRAST_([\d]+)_CT.tif')
label = []
id_list = []
for image in all_images_list:
    id_list.append(check_contrast.findall(image)[0][0])
    label.append(check_contrast.findall(image)[0][1])
label_list = pd.DataFrame(label,id_list)
images = np.stack([jimread(i) for i in all_images_list],0)
```

Split the train and test Dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(images, label_list, test_size=0.1,
 random_state=0)
n_train, depth, width, height = X_train.shape
n_test,_,_,_ = X_test.shape

input_train = X_train.reshape((n_train, width,height,depth))
input_train.shape
input_train.astype('float32')
input_train = input_train / np.max(input_train)
input_test = X_test.reshape(n_test, *input_shape)
input_test.astype('float32')
input_test = input_test / np.max(input_test)
output_train = keras.utils.to_categorical(y_train, 2)
output_test = keras.utils.to_categorical(y_test, 2)
output_train[6]
output_train[8]
output_test[5]
input_train[5]
```

Train the Machine Learning Model

```
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.optimizers import Adam
from keras.layers import Conv2D, MaxPooling2D
batch_size = 20
epochs = 100
model2 = Sequential()
model2.add(Conv2D(50, (5, 5), activation='relu', input_shape=input_shape))
model2.add(MaxPooling2D(pool_size=(3, 3)))
model2.add(Conv2D(30, (4, 4), activation='relu', input_shape=input_shape))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Flatten())
model2.add(Dense(2, activation='softmax'))
model2.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
history = model2.fit(input_train, output_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(input_test, output_test))
model2.save('model_dicom_cancer.h5')
```

Let’s test the Model

```
import pydicom as dicom
import matplotlib.pylab as plt
from skimage.transform import resize
image_path = '<IMG_PATH>.dcm'
ds = dicom.dcmread(image_path)
test1 = ds.pixel_array
IMG_PX_SIZE = 256
resized1 = resize(test1, (IMG_PX_SIZE, IMG_PX_SIZE, 1), anti_aliasing=True)
resized1.shape
pred1 = model2.predict(resized1.reshape(1,256, 256, 1))
round_prediction1 = np.round(pred1[0])
prediction = str('%.2f' % (pred1[0][1]*100) + '%')
round_prediction1 = np.round(pred1[0])
prob = str('%.2f' % (pred1[0][1]*100) + '%')
if pred1[0][1]*100 < 90:
    print('Normal Patient')
else:
    print(prob,"The patient has Cancer disease")
plt.imshow(test1, cmap="bone")
```

Finally, the model is imported in the Flask applications we discussed above.

7.6.5 Cluster setup

We have a set of nodes that run our containerized applications. We had packages an app with its dependencies and some necessary services. To start working with this model, we will follow these steps:

Namespace Setup

```
apiVersion: v1
kind: Namespace
metadata:
  name: cancerns
```

Dataset Setup

- Build a Persistent Volume and a PVC on Longhorn as a storageclass

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-pvc
  namespace: cancerns
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: longhorn
```

- Build separate PV for dataset

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ds-pvc
  namespace: cancerns
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: longhorn
```

- Create a deployment script and attach Persistent Volume and run a script to download dataset

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: datasetvm
  namespace: cancerns
spec:
  replicas: 1
  selector:
    matchLabels:
      app: datasetvm
  template:
    metadata:
      labels:
        app: datasetvm
    spec:
      containers:
      - name: datasetvm
        image: "ubuntu:latest"
        imagePullPolicy: Always
        volumeMounts:
        - name: dataset
          mountPath: /dataset
        env:
        - name: DATASET
          value: "https://rancherdataset.blob.core.windows.net/ct-images/dataset.zip"
        command: ["/bin/sh","-c"]
        args: ["apt-get update; apt-get install unzip wget -y; wget $DATASET -O /dataset/
dataset.zip; unzip /dataset/dataset.zip -d /dataset/dataset; ls -l /dataset/dataset"]
      volumes:
      - name: dataset
        persistentVolumeClaim:
          claimName: ds-pvc
```

Doctor App Setup

Doctor's application had Deployment and Service manifests. - kustomization.yaml

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
- service.yaml
```

- Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: doctor-app
  namespace: cancerns
spec:
  replicas: 1
  selector:
    matchLabels:
      app: docapi
  template:
    metadata:
      labels:
        app: docapi
    spec:
      containers:
        - name: doccontainer
          image: abhinav332/doctor-app:v5
          imagePullPolicy: Always
          volumeMounts:
          - name: dst
            mountPath: /dst
          - name: dataset
            mountPath: /dataset
          ports:
            - containerPort: 5002
              protocol: TCP
      volumes:
      - name: dst
        persistentVolumeClaim:
          claimName: data-pvc
      - name: dataset
        persistentVolumeClaim:
          claimName: ds-pvc
```

- Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: doctor-svc
  namespace: cancerns
spec:
  ports:
  - port: 5002
    protocol: TCP
    targetPort: 5002
  selector:
    app: docapi
  type: LoadBalancer
```

Run the application using: `kubectl apply -k doctor-app/`

Lab Technician Application

Lab Technician Application have Deployment, Service and Kustomization manifests. Let's take a look over the manifests:

- kustomization.yaml

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- deployment.yaml
- service.yaml
```

- Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: labtech-app
  namespace: cancerns
spec:
  replicas: 1
  selector:
    matchLabels:
      app: labapi
  template:
    metadata:
      labels:
        app: labapi
    spec:
      containers:
        - name: labcontainer
          image: abhinav332/lab-app:v4
          imagePullPolicy: Always
          volumeMounts:
          - name: dst
            mountPath: /dst
          ports:
            - containerPort: 5001
              protocol: TCP
      volumes:
      - name: dst
        persistentVolumeClaim:
          claimName: data-pvc
```

- Service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: labtech-svc
  namespace: cancerns
spec:
  ports:
  - port: 5001
    protocol: TCP
    targetPort: 5001
  selector:
    app: labapi
  type: LoadBalancer
```

Run the application using: `kubectl apply -k lab-tech/`

8 Demonstration

8.1 Workflow

This application is a combination of various tools and technologies embedded together on a same platform. Here, I will demonstrate you DIY guide to the application:

- Clone the official GitHub repository:
  - `git clone https://github.com/abhi-bhatra/ct_image_scanning.git`
- Ensure to change the branch:
  - `git checkout UI_base`
- Set-Up Application on k8s cluster

```
# Change the directory to the Kubernetes manifest
cd k8s/

# Set up a new namespace for the application
kubectl apply -f namespace.yaml

# Change the directory to dataset/ folder to install and create a volume which contains a
 dataset
cd dataset/ && kubectl apply -k dataset/ && cd ..

# Navigate to lab-tech/ dir, it will install the lab technician UI
kubectl apply -k lab-tech/

# Navigate to doctor-app/ dir, it will install the Doctor's Dashboard UI
kubectl apply -k doctor-app/

# Navigate to the Kubeflow to install the kubeflow, you will be needing to run this bash
 script to install the kflow pipeline
cd kubeflow/ && bash kflowsetup.sh
kubectl apply -f kubeflow-istio.yaml
```

8.2 Application Walkthrough

9 Summary

Healthcare Space needs to be very cautious when using AI for Cancer detection. Aritficial Intelligence is indeed a powerful tool which saves patient's life and physician's time. But it can be devastating if not trained and deployed correctly. Some key checkpoints we have undertaken:

- Make sure that our AI system for cancer detection does not contain any rooted bias. If that happen, we eliminates bias from Training Data and retrain
- We have invested a lot of time to search for data. In future, this data is not only used for Cancer Detection, but also benefits the predictive analytics in complete Healthcare space.
- We have worked on building self-learning and explainable AI to fight cancer if black-box concept is a challenge and integrating MLOps ideology.