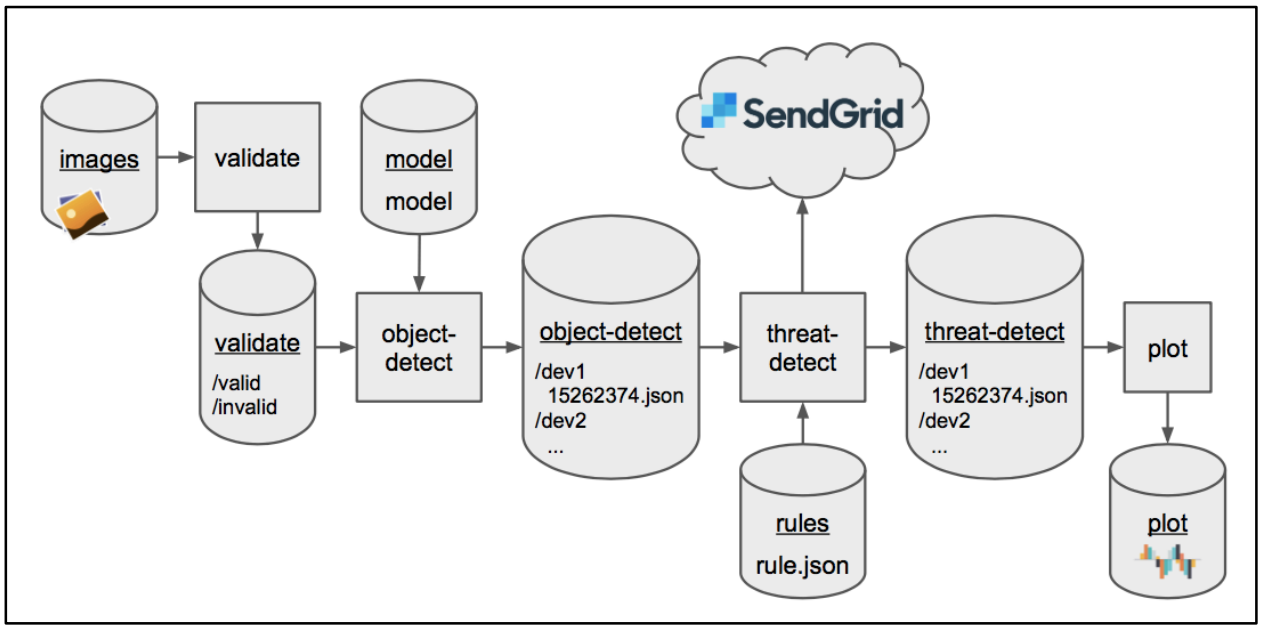
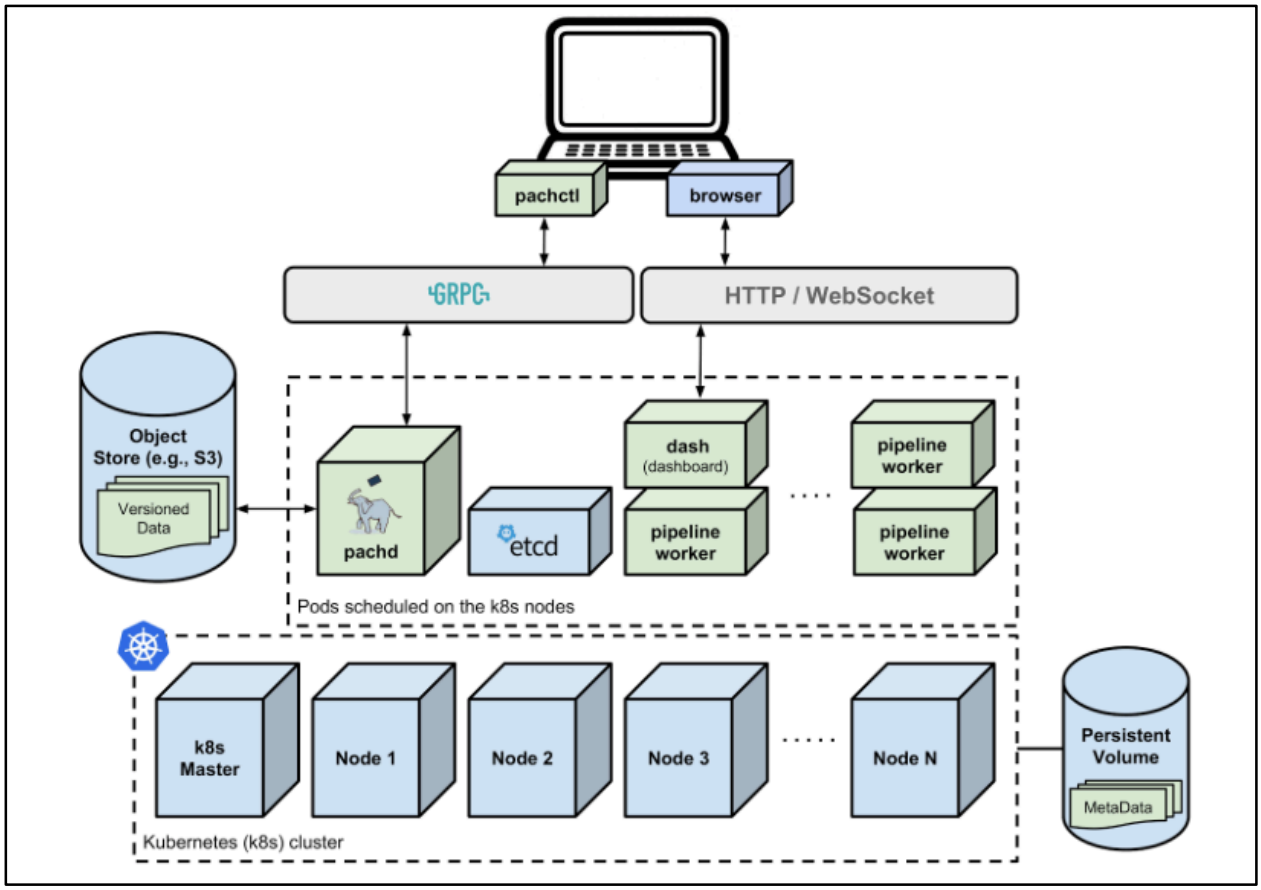
Updating, Maintenance and Scaling

Implemented System:



The images for threat detection will be stored in the Persistent volume. The images will go through 'validate' process which will check images based on resolution and other parameters, and create file names. The 'validate' 'image' in the Docker container will check for validity of image. It will store the valid and invalid images separately. The TensorFlow model developed for object detection for different clients will be stored in 'model' repository. The Python 'image' will consist of codes that are executable using Python libraries in the 'TensorFlow' image in Docker container. The 'object detect' will process images and detect 'objects' in the image. The 'objects' detected in the image will be classified as threat or friendly based on rules in the file 'rule.json'. For the purpose of demonstration, we will consider only one rule. Different clients can have different threat perception and this will help determine if the 'object' in image is a threat or not. 'Threat-detect' 'image' in the Docker container will contain the package necessary to run for threat detection. The threat will be described in form of a json file with the date and time the threat was detected along with its location. This information will go to the user using an email service such as SendGrid. The threat once detected will be stored in the separate repository for regulatory purpose. This storage will happen in a 'static' 'object' storage platform like S3. For internal customer analytics, we will plot the number of threats against user or location. All the computation will happen on EC2 instance. Kubernetes will manage the EC2 instances (VMs). We will create a Pachyderm 'image' stored in Docker container called as 'pachd', that will automate the entire process of running different 'images'. It will under the hood manage Kubernetes and optimize the utilization of EC2 instance.

Spin It Up:



Any containerized application typically consists of multiple containers. In such cases, a multi-container application can be deployed on multiple hosts. Users may need an external tool to manage such multi-container, multi-host deployments. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications.

Operating Kubernetes for production applications presents a number of challenges. Firstly, there is a need to manage the scaling and availability of Kubernetes masters and persistence layer by ensuring that appropriate instance types are chosen. The health of nodes needs to be monitored so that unhealthy nodes are replaced. Also, there is a need to patch and upgrade masters and worker nodes to ensure that we are running the latest version of Kubernetes. This all requires expertise and a lot of manual work. This is where paid services such as GKE and Amazon EKS are important.

Kops, short for Kubernetes Operations, is a set of tools for installing, operating, and deleting Kubernetes clusters in the cloud. A rolling upgrade of an older version of Kubernetes to a new version can also be performed. It also manages the cluster add-ons. After the cluster is created, the usual Kubectl Command Line Interface can be used to manage resources in the cluster.

We can install Kubernetes on EC2 either manually by following a series of steps. Another option is to use service such as Google Kubernetes Engine or Amazon EKS (Amazon Elastic Container Service for Kubernetes) for the advantages they provide as above. Another way to spin it up without a service is using kops as discussed above.

We will use a R4.X large EC2 instance on AWS with a 30.5 GiB memory. This will provide us with sufficient memory and computational requirements. On all Virtual Machines, (EC2 instances) there is Docker and Kubernetes installed. Master Kubernetes will be installed on one of the Virtual Machines (VM) which will play an important role in managing the Cluster. Somewhere on one VM, there is Pachd Pod – which contains Pachd container. The Pachd pods manage how pods would validate, perform object detection, detect threat etc. This is to ensure that right data is processed at right time. Pachyderm will work with Kubernetes under the hood to manage the resources in an optimum way.

Docker image made on laptops are either pushed as images on Docker repository or Docker hub. Docker Hub repositories allows us to share images with co-workers, customers, or the Docker community at large. If we are building images internally, either on our own Docker daemon, or using Continuous integration services, we can push them to a Docker Hub repository.

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like hard-drives, but have a lifecycle independent of any individual pod that uses the PV. This API object captures the details of the implementation of the storage. Persistent Volumes is like hard-drive that works with 'eacd' to manage what has been run, what is next on EC2 machine etc.
In class, we tried to check if 'Validate' works and an Invalid image doesn't run. For eg blah.jpg will not run.

Updating and Maintenance:
In case when we are required to build a new container, we must first version the old container. By not immediately removing the original container, we have the ability to revert back to the known working container if the upgraded container doesn't have the right data, or fails a sanity test. Before stopping that container and running a new code, we must first test out our new container. Testing will allow comparison between 2 outputs.

Once the test is successful, we now have the opportunity to make sure that the data we expect to be in the new container is there and run a sanity check. The docker volumes will stick around so long as any container is using them, so we can delete the original container safely. Once the original container is removed, the new container can assume the namesake of the original to make everything as it was to begin.

Implementation:

Docker build –t dwhitena.mgmt–thread–detect:v2

Giving a unique name will prevent confusion between different containers for Kubernetes

Docker push dwhitena.mgmt–thread–detect:v2

This will push image on Docker container

Pachctl update–pipeline –f threat–detect.json

This will get the pipeline spin up new container

Kubectl logs po/pipeline–object–detect–v1…..

OR

Pachctl list–job

This will give the logs from running pods

Data Provenance provides a historical record of the data and its origins. The provenance of data which is generated by complex transformations such as workflows is of considerable value to scientists. From it, one can ascertain the quality of the data based on its ancestral data and derivations, track back sources of errors, allow automated update of data, and provide attribution of data sources. So, if something say is weird about plot I obtained about threats detected at different locations, I can go and check what was wrong at that time with code or input. For any change made to code, there is id associated with that change. Similarly, id is generated whenever new output is generated. State of object detect, validate, threat-detect is available using pachctl command.

Pachctl inspect–commit plot (name of plot)

Things to Improve in Future:

1. In future, we would like to automate the image obtaining process. The objective in that case is to sample the video for image - Group the image, process batch of image together. This is expected to improve the efficiency and performance of the system.
2. Images of different resolution can be pre-processed, this will help to run images from different clients with fewer invalid images.
3. If we have a large number of images to process say 1000 images, to work faster, we can spin up multiple instances of 'image' of say 'object-detect'. This will solve the problem – through parallelism. A part of EC2 instance can be asked to do parallelism, so we can have say 5 'threat-detect' 'images' that are running in parallel.
4. An improvement to existing system of alerting through email would be to Message/ Call the client to immediately alert about the intruder. We can also develop an app for the client and alert him/ her through an in-app notification.
5. Face recognition to recognize and monitor the movement in the premise of building, this will help us go beyond the current application in which we can decide an intruder in midst of legitimate entrants.

Appendix:

1. https://kubernetes.io/docs/concepts/storage/persistent-volumes/
2. https://docs.docker.com/docker-hub/repos/
3. https://aws.amazon.com/ec2/pricing/on-demand/
4. https://kubernetes.io/docs/reference/kubectl/overview/
5. https://aws.amazon.com/blogs/compute/kubernetes-clusters-aws-kops/