spectro cloud

Zen and the Art of ML on Kubernetes

Introduction 2

ML on Kubernetes 3

- Tooling availability 3
- Effective Resources Utilization 4
- Separation of Concerns 4

Popular ML tools with Kubernetes 5

- AI, Machine Learning and Deep Learning 5
- Machine Learning Phases & Tools 7
  - Experimental Phase 8
    - Data Preparation, Cleanup, and Visualization 9
    - ML Framework and Model Algorithm 10
    - Model Training 11
    - Validation and Tuning 13
  - Production Phase 14
    - Model Training 15
    - Model Serving 16
    - Model Performance and Tuning 16
- Higher-level Machine Learning Tools 17
  - Jupyter Notebook 17
  - Pipelines 19
  - KubeFlow 19

How Spectro Cloud Can Help 20

Introduction

With businesses and enterprises continuing to transform themselves into technology companies, applications are the new life-blood of business. Applications are defining the way most companies in many industries compete, to provide the best customer experiences and drive competitive advantage. As these companies look to further drive positive business outcomes, they're increasingly relying on insights from the massive amounts of data they've generated and collected over the years. However, with traditional algorithms finding it difficult to process and provide any meaningful value from the data, organizations are turning to data science and machine learning. These new technologies not only help analyze the data, but provide real-time business decisions and intelligence, without human intervention. From human-like voice assistants, to fraud detection or even automatic mortgage approvals - the possibilities of machine learning applications are limitless.

Increasingly conventional and machine-learning applications alike are built and deployed as containers, running on the popular orchestration platform Kubernetes. Enterprises are adopting Kubernetes to drive down operational costs, increase business agility, and provide faster time to market for their applications.

In this paper, we'll cover:

- the business and technical advantages of running AI/ML technologies and workloads on top of Kubernetes,
- some of the more popular frameworks and technologies data scientists use to power their machine learning applications,
- ways in which Spectro Cloud can help data-scientists, developers, and operators alike by providing the flexibility to run and operate Kubernetes clusters and AI/ML frameworks.

ML on Kubernetes

Kubernetes has emerged as the de-facto run-time platform to manage container-based workloads. For containerized workloads, Kubernetes provides significant capabilities and features with little or no effort required: application deployment, scaling, updates, service discovery, resiliency, and more. Application workload portability is also enabled - whether these workloads are running on-premise, in the cloud, or at the edge.

In this section, we'll identify some of the reasons why Machine Learning is a good fit to run on the Kubernetes container orchestration platform.

Tooling availability

There are a myriad of different AI/ML frameworks and tools available for data scientists and machine learning developers to use; the Machine Learning Tools section will outline some of the more popular tools used in the various phases of Machine Learning.

With the rapid adoption of Kubernetes in the enterprise, most of these frameworks are providing first-class support to containers and Kubernetes, with easy to use helm chart scripts for installation, security, upgrades, and other day-2 operations. Adding or replacing a ML framework or tool in Kubernetes really is now just an "application" install, without having to configure or replace your infrastructure.

Containers have the added benefit of providing consistent and reproducible behavior; this is especially important in ML workloads which can be version sensitive with respect to the libraries and frameworks they use (e.g: TensorFlow 1 vs TensorFlow 2). Containers also allow drivers and plugins, e.g: NVIDIA GPU drivers, to be baked into the container images; so regardless of where the workloads are deployed, the workloads are deployed with the same application, dependencies, and libraries throughout the machine learning pipeline.

Effective Resources Utilization

Kubernetes at its core is an application workload scheduler, with algorithms to optimally place and balance workloads across nodes with available capacity. Kubernetes monitors and tracks a number of different “resources”, such as CPU, memory, storage, and other hardware devices (such as GPUs/TPUs). Adding additional capacity is as easy as scaling out new worker nodes (physical or virtual). The new nodes’ additional resources are automatically added to the cluster, and can be used by the scheduler for subsequent workload placements.

This makes Kubernetes a perfect fit for machine learning workloads, which requires varying amounts of resources depending on the phase and stage of the pipeline. During the data import and preparation phases, the system needs heavy storage and potentially network resources. Training the machine learning “model” requires heavy bursts of CPU usage, and/or GPUs/TPUs/FGPAs, which can significantly speed up training.

Kubernetes will ensure that each workload gets the appropriate resources that it requests; if there is a shortage of capacity, workloads are moved to a “pending” state until either existing running workloads complete or new capacity is provisioned. Cloud environments can also be configured to automatically scale-out workers to increase capacity.

Separation of Concerns

Machine Learning tools and Kubernetes are both complex platforms with many moving parts. As with most infrastructure platforms, continued usage builds expertise with running and operating production-grade platforms. Over time, organizations develop operational runbooks to provision, upgrade, secure, monitor, and troubleshoot. With Kubernetes, most of these best practices are applied regardless of the application workloads running on the cluster, whether they be long-running batch compute jobs, general purpose web applications, or machine learning workloads.

Building familiarity with a single underlying platform, like Kubernetes, brings business agility and cost savings, as there is no longer a need for developers to write and support container automation scripts for each workload. Since almost all machine learning tools and frameworks are built using containers, and many are optimized for Kubernetes, teams leverage the same tools they're already using for logging, monitoring, and tracing.

Provisioning and maintaining Kubernetes clusters requires continuous effort, even in environments which provide managed services. However, once the cluster is operational all applications benefit from the PaaS-like capabilities Kubernetes provides - regardless of whether the cluster is running on a laptop, on an on-prem datacenter, or in the cloud. IT teams gain the flexibility to run software where the business needs it.

Kubernetes natively also provides soft multi-tenancy features, which may be useful for organizations with multiple teams, as they can now leverage a shared cluster, reaping the benefits of more efficient use of resources and ease-of-manageability from maintaining a smaller set of Kubernetes clusters. This also applies to a single team which chooses to run different sets of workloads and apps on a single cluster.

Popular ML tools with Kubernetes

Data Science, Machine Learning, and especially Deep Learning have received a great deal of attention in the last several years. There are literally 100s of different projects and frameworks coming out each year which tries to address some aspect in this nascent field. In this section, we'll give a high-level overview of the differences between AI, Machine Learning, and Deep Learning, followed by a deeper look at the processes inside machine learning and the tools/frameworks available today.

AI, Machine Learning and Deep Learning

Artificial Intelligence (AI) is all around us, from your personal Google Assistant, to the computer playing chess programs, to the traffic stop lights we *used* to pass everyday going to work. AI enables machines to learn from experience and observation; they continuously adjust their

response based on the large amounts of data and inputs they receive. In the broadest sense, Artificial Intelligence is any algorithm or program that does something intelligent.

There are three-categories of AI: Artificial Narrow Intelligence (ANI), Artificial General Intelligence (AGI), and Artificial Super Intelligence (ASI). All the algorithms and programs today exist only in the realm of ANI - also known as “weak” AI. They’re meant to only do one specific task - like driving a car down the freeway, automatically marking suspicious emails as spam, or approving your home mortgage application. This is markedly different from Artificial General Intelligence, where machines exhibit “human-like” intelligence; with the ability to solve, make judgements, learn, and be creative -- all based on applying prior learned knowledge. The last stage, Artificial Super Intelligence, are essentially machines which far-exceed human capabilities in all regards.

A subset of Artificial Narrow Intelligence, is the field of Machine Learning, where systems learn from large datasets to perform automatic approvals, object recognition, speech detection and others. Machine learning learns and recognizes patterns on its own to make educated predictions. Most of the tools and examples in the sections below are related to Machine Learning.

Deep Learning (DL) is a subset of Machine Learning; algorithms which in addition to machine learning capabilities can more accurately provide insights into image recognition, object detection, advanced classification, fraud detection, and more. Machine Learning algorithms employ an “artificial neural network”, similar to neurons in the human brain, to make more accurate predictions.

Even though DL is a subset of ML, the key difference in Deep Learning is the ability to automatically correctly identify features/characteristics which play an important role in prediction. These learned characteristics are assembled into a construct called a *model*, which can then be used for realtime predictions. For example, if we trained a classification model to identify the type of vehicle given a picture, a deep learning algorithm, with no manual intervention, would over time learn that cars have the following features: four wheels, two headlights, 6 windows (passenger, front, and back), along with other characteristics a “car” may have - compared to say, a motorcycle or truck. This is in stark contrast to a conventional

machine learning algorithm, where the model would need to be programmed to look for these specific features.

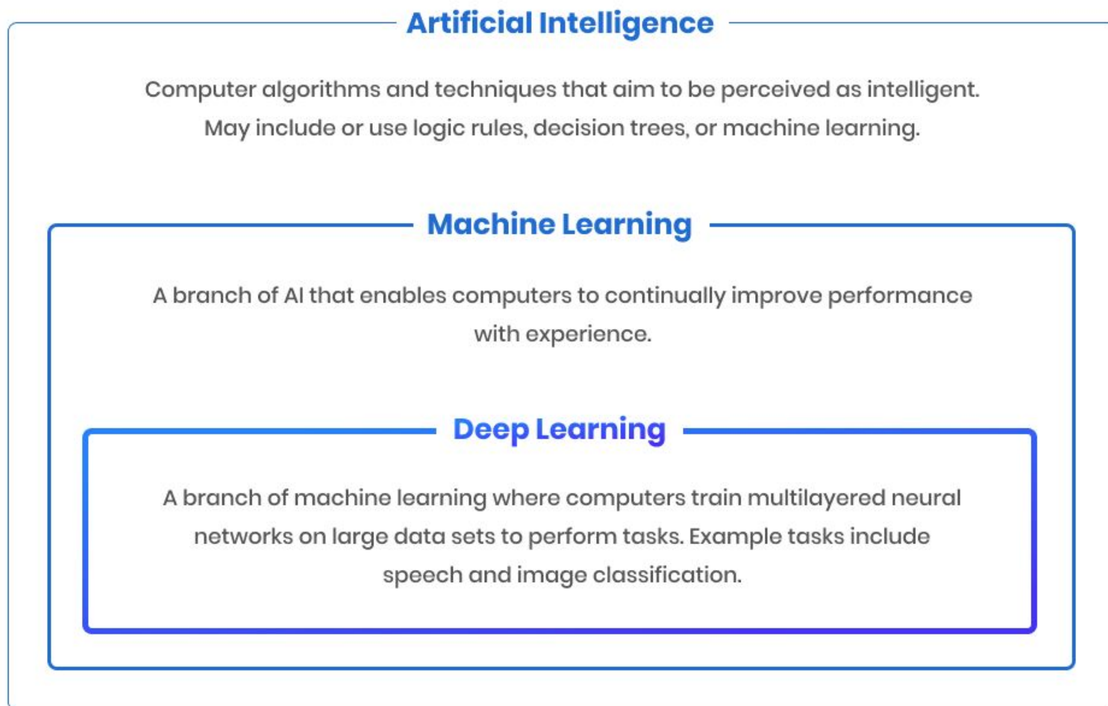

Figure 1. Branches of Artificial Intelligence

Machine Learning Phases & Tools

Machine Learning has two distinct phases: the *Experimental* phase, where the model is developed and built, and the *Production/Deployment* phase, where the model is deployed for inferencing and predictions. Each of these phases have multiple stages - where certain distinct activities are performed. During the ML workflow, each stages' output is fed as input into the next stage for processing. The whole workflow is an iterative process, and may be done as many times as necessary to obtain the desired results.

Data Science and ML technologies exist in nearly every conceivable programming language, but the most popular languages in this realm are Python and R, with Python leading by a wide margin [1]. While Python is a relatively flexible language with good support for basic operations, its runaway success can be attributed to the large number of open source projects and tools

[1] Programming Languages Most Used and Recommended by Data Scientists

available for data science and machine learning. With so many data scientists already using solely Python in their day-to-day activities, even newer frameworks and tools are built first for Python.

Experimental Phase

The Experimental (or Development) phase is where the model is first conceived and developed. The data is analyzed and reviewed, an ML framework is chosen, an initial model is trained, and parameters are tuned. Generally the Experimentation phase stages include:

- Data preparation, cleanup, and visualizations
- Choosing of an ML framework library and model algorithm
- Model training and experimentation
- Iterative tuning of the model parameters and validation

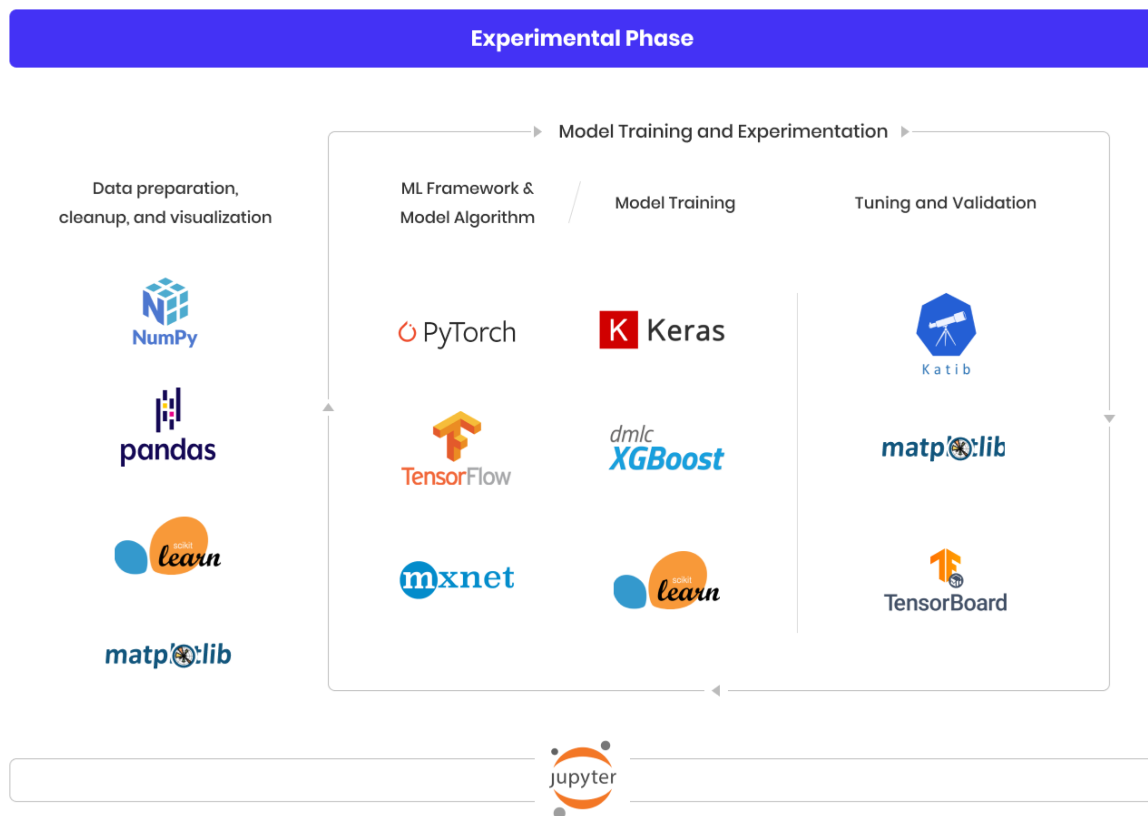

Figure 2. Experimental phase stages and and tools

Data Preparation, Cleanup, and Visualization

Building dependable machine learning models requires complete, clean data. It's during the data preparation phase where data scientists will spend time pruning and fixing incomplete data. While good machine learning algorithms can cope with some faulty or noisy inputs, in the end the trained model is only as good as the data from which it is trained on. According to a recent survey [2], data scientists spend over 40% of their time just preparing and cleaning the data!

Some of the more popular Python libraries used include:

- Numpy: Most of the ML frameworks natively support Numpy for all their data transformation. Numpy provides a multidimensional array with a wide array of commonly used functions to average, aggregate, drop, and others.
- Pandas: is similar to Numpy, provides a high-performance and easy-to-use data structure called a DataFrame. The DataFrame provides an in-memory 2d table, very similar to a spreadsheet with columns and rows. Extremely expressive language to filter and aggregate data, with also some built-in functionalities to either drop columns with missing values or impute missing values either using averages of that column OR in categorical columns, default to the most common label.
- Scikit-learn: is a massive data science library, which can be used in nearly every stage. Scikit-learn, imported as sklearn in code, has imputing and averaging functions built-in.
- Matplotlib: is the king of plotting and visualization in Python. Provides the low-level functionality, to plot charts, graphs, images, and everything else. Seaborn is a higher-level plotting library with easier syntax and great out of the box defaults.

ML Framework and Model Algorithm

For the most part, selection of a ML framework is a matter of choice, since they provide similar capabilities and performance. Some operate at a higher-level, making it easy to construct massive models with ease (like Keras), while others operate at a lower-level - opting to provide complete control and flexibility, over ease-of-use.

[2] How do Data Professionals Spend their Time on Data Science Projects?

Here are some of the more popular machine learning and deep learning frameworks:

- Scikit-learn: provides machine learning models for classifications, regressions, and clustering. Popular choice for new data scientists, however it does not provide any deep learning features.
- MXNet: flexible and deep learning framework. Language bindings for a handful of other languages such as R, Java, Scala - and popular with non-python coders.
- PyTorch: one of the newest deep learning frameworks by Facebook. Has gained a lot of recent popularity due to its simplicity and ease of use - making it great for rapid prototyping, especially for small-scale or academic projects.
- TensorFlow: the most popular open source machine learning library by Google. With support for large distributed training, or on the smaller scale on mobile devices. Extremely active community and used for most production workloads.
- Keras: is a higher level framework which comes with pre-built models and sample datasets. Fully integrated into TensorFlow.

The choice of machine learning algorithm is crucial and will significantly alter the performance and accuracy of the model. The various machine learning algorithms are out of the scope for this paper. Some of the more popular machine learning algorithms include: K-nearest neighbor and Linear Regression. Popular Deep Learning algorithms with deep neural networks include Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN).

Deep neural networks can be quite complex to setup, and generally require skilled data scientists, referred to as “network architects”, to design the layers and nodes of connectivity in the deep neural networks. For image classification problems, there are a handful of common pre-built neural networks provided by some ML Frameworks as starting points (e.g: VGG16, VGG19, ResNet50, Inception)[3], and also sample datasets (e.g: Imagenet, CIFAR-10, etc).

[3] ImageNet: VGGNet, ResNet, Inception, and Xception with Keras

Model Training

In the next stage of the ML pipeline, training and experimentation of the model begins. For image classification and object recognition ML examples, the most common implementation is supervised learning. In supervised learning, the input data contains one or more labels which correspond to the training which is done. For example, in the vehicle type classification example above, the input data would comprise a set of images of cars, trucks, motorcycles, etc - along with the actual classification of each, “car”, “truck”, “motorcycle”. This label data is crucial in supervised learning for the model to train itself to distinguish a “car” from a “motorcycle”.

During training, not all of the input data is fed into the model for training; if this was the only source of data, we’d have nothing left to validate against. To illustrate this, imagine if you trained the model with only five car images and five motorcycle images. For validation, if you were to upload an already trained car image, you would of course expect the model to accurately predict “car” (if not, you have a *terrible* model!). The true test and accuracy of a model is not with images it’s already seen, but with unseen images. During experimental training, it’s generally best practice to split your input data into three sets - training, validation, and test[4]. The test set is only used to evaluate the final model performance. The training set is used to actually fit and train the model, whereas the validation set is used to fine-tune the model and parameters. Some advanced techniques such as K-Fold Validation expand on this - and allow you to train and validate with both the train and validation sets - ensuring model performance is consistent.

The aspect of how a neural network trains itself is out of scope of this paper, a good reference for deep neural networks implementation is: [What is Deep Learning and How Does it Work?](). At a high-level, the neural network is trained with the training set data over multiple passes, called an epoch. During each pass, or epoch, parameters on each of the layers and nodes are slightly modified to correctly identify or predict the training set data. During this fitting process, the overall optimization algorithm attempts to minimize loss and maximize accuracy[5].

---

[4] Train, Validation and Test Sets
[5] Accuracy and Loss - AI Wiki

The model training is done with the same frameworks discussed in the previous ML Framework and Model Algorithm section. Each of these frameworks will have a “fit” method passing in the training set. Depending on complexity of the model and size of the input data, this can take anywhere from minutes to weeks.

On performance, when training complex models (such as any deep neural network), the computations run significantly faster on GPUs, TPUs, and other specialized hardware - sometimes by a factor of 10x! Modern GPUs contain 1000s of cores and can easily handle the large number of parallel computations performed during ML training.

Validation and Tuning

As part of training the model, data scientists are continuously validating the model against the validation set to ensure the trained model is producing accurate predictions. Graphs help visualize the progress and performance of the model, perhaps overfitting or underfitting the data, both very frequent problems[6].

This is an iterative process, where after each validation the hyperparameters for machine learning models are modified and a new training and validation cycle begins. For small models, this can be done manually, however there are tools like Katib which automate much of the tuning process: they create multiple parallel trials where various parameters such as learning rate, number of nodes, layers, etc are modified, the model is trained with the new parameters, and the resulting validation accuracy is used to determine whether the experiment should stop or continue.

Once the final model with parameters has been selected, data scientists frequently evaluate the model once more against the previously unseen “test dataset”. The performance is a likely indicator of the model will perform in production.

Common tools used in the Validation and Tuning stages:

- Katib: is a tool for automated experiments to fine-tune hyperparameters (number of nodes, layers, etc).

[6] How to use Learning Curves to Diagnose Machine Learning Model Performance

- TensorBoard: is a tool to plot training and validation losses along with the hyperparameters for each model. Makes it easy to visualize and spot problems.
- Matplotlib: Python graphic library to plot charts, graphs, and images in Jupyter notebooks.

Production Phase

During the Production phase, the model is deployed for use in an actual application for inference and predictions. The following stages are observed:

- Model Training (optional)
- Model real-time serving or scheduled batch predictions
- Monitor model's performance and tuning

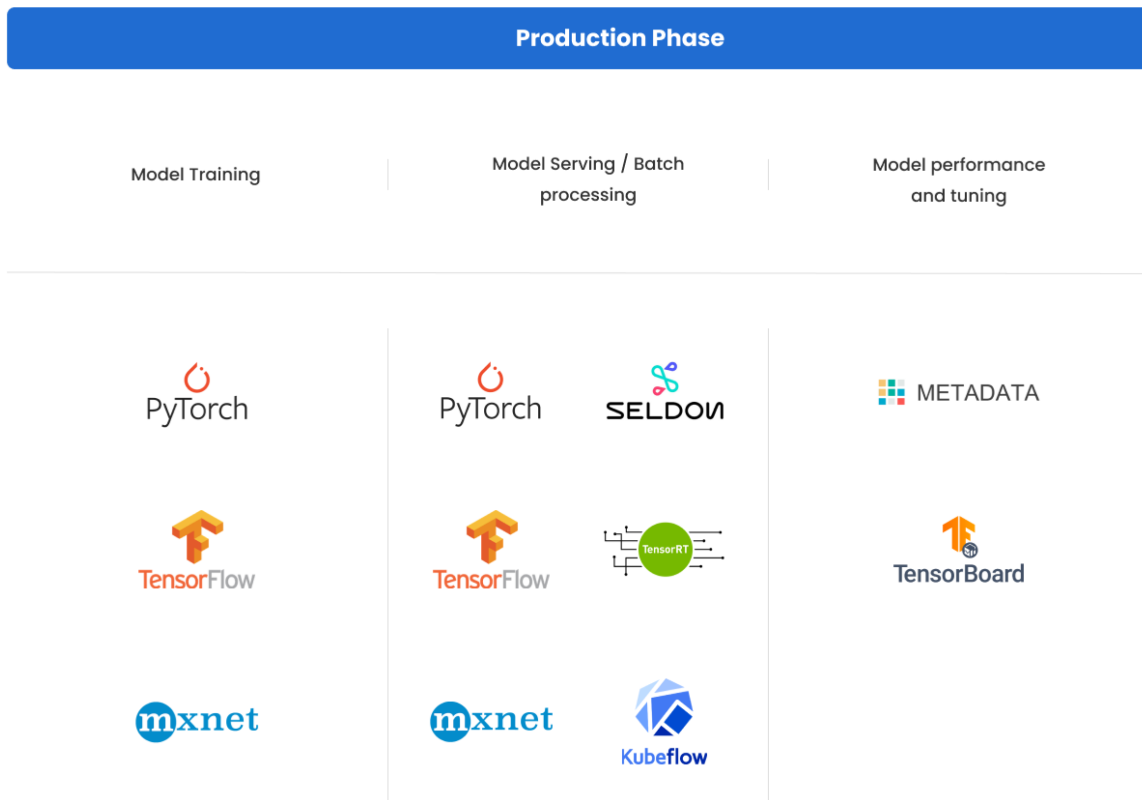

Figure 3. Production phase stages and tools

Model Training

Model training is generally not repeated between the Experimental and Production phases. If the model was properly trained and tuned during the Experimental phase, the trained model can be used as-is with serving tools discussed in the next section. Some teams choose to re-train the model in production with all the input data (training set and validation set), especially if the input set is small.

Over time, as the model starts to drift (see Model Performance and Tuning), the model is retrained with new input data. If there is a large enough variation in the input data, it's recommended to retrain the model from scratch, otherwise batches of new input data can be trained on-top of the existing model. Online learning is a newer variation which trains the models continuously for each observation observed; ensuring that the model stays up-to-date and adapts to new patterns in the data.

Model Serving

For each of the ML Frameworks, like TensorFlow and PyTorch, there are corresponding "serving" libraries/tools. These serving tools take an exported model and the corresponding weights/configuration, and deploy it with REST/gRPC APIs. A client application can then make the corresponding REST or gRPC call to request a prediction; the response will contain the likely prediction and explanations by the model.

Some of the popular serving tools are:

- TensorFlow Serving: is high-performance serving engine for TensorFlow models
- NVIDIA TensorRT: provides a highly-optimized serving engine for use with NVIDIA GPU cards.
- KFServing: is a sub-project of KubeFlow, is a popular tool to deploy your models to production. It provides an abstraction layer over many different ML framework's models (TensorFlow Serving, PyTorch). With the abstraction, entire ML frameworks can be swapped without affecting clients and users.
- Seldon: similar to KFServing, provides REST/gRPC API endpoints for models built from ML Frameworks.

Model Performance and Tuning

Most machine learning models begin degrading quite quickly after deployment[7]. It's not that the prediction for the same requests change over time, but rather the opposite. Machine Learning is useful where inputs and outputs are changing week-to-week, even day-to-day. Imagine building a Machine Learning algorithm to provide approve-and-deny signals for house mortgage applications. If the model was trained during an unhealthy and faltering stock market/economy, the *general* supervised learning from actual loans approvals and rejections could've been to approve 95% of applications with a credit score of 750+. Now, since the model was deployed, if the stock market starts rebounding and economy shows signs of recovery, human loan officers may loosen up the loan approval to applications with a credit score of 700+ - but unless the deployed model isn't updated with new training, the robo-agent is still harshly rejecting clients who would have otherwise been approved!

More so than general-purpose applications, machine learning models need continuous monitoring and fine-tuning, to ensure model accuracy and performance.

Previously mentioned TensorBoard can be used to graph the performance model over time. It's advisable to regularly evaluate the model with newer test data sets, and to retrain the model when performance drops below acceptable metrics. KubeFlow Metadata is a new project which tracks the metadata of artifacts created by the machine learning process. The repository will track deployed models (including their learning rate, layers, and node parameters), input data sets, and the evaluation results of the models. Over time these tools give insights into which key factors drive positive outcomes on the models and their results.

[7] Why is Retraining so Important

Higher-level Machine Learning Tools

As described in the previous sections, there is an arsenal of tools available for data scientists to use as they're building and running their machine learning workloads. It helps to break down these tools into various categories:

- ML/Data Science libraries (mentioned previously): these are used to help analyze data, plotting, and validation
- Deep Learning Frameworks (mentioned previously): specific neural networks and models used for training and inferencing
- IDE or interactive tools: interactive web applications which let data scientists collaborate and share documents that contain live code, visualizations, and narrative text
- ML Automation: repeatable ML pipelines which can be shared anywhere

The ML/Data Science Libraries and the DL Frameworks are programming libraries your application depends on. Almost all of them are installed and maintained using the programming language's package manager, e.g: pip3 for Python.

In this section, we'll cover some of the higher level machine learning tools in use:

Jupyter Notebook

Python provides best-in-class tooling to run data science and machine learning workloads (not surprising, since over 80% of data scientists only use Python). One of these include Jupyter Notebook, an interactive web application that allows users to create and share documents that contain live code, equations, visualizations and narrative text.

Most of the experimental phase will be within a Jupyter notebook - data preparation, training, and validation are triggered from within a Jupyter notebook. The notebook itself is a fully encapsulated file, which can be shared with team members and committed to a code repository.

```
In [73]: train_path = extract_path + '/training_set'
         test_path = extract_path + '/test_set'

         fig, axs = plt.subplots(2,3, figsize=(10,7))
         axs = axs.flatten()

         # fig.suptitle("Cats and Dogs")
         cats = ["/cats/cat.%s.jpg" % n for n in [1,7,100]]
         dogs = ["/dogs/dog.%s.jpg" % n for n in [2,8,99]]

         for img, ax in zip(cats + dogs, axs):
             img = load_img("%s/%s" % (train_path, img), target_size=(224, 224))
             ax.imshow(img)
             ax.axis('off')

         plt.tight_layout()
```

Figure 4. Jupyter notebook with code block and a matplotlib output!

The Jupyter Notebook server which hosts the Jupyter UI and provides the python runtime must be installed and configured separately. Also, even though there are published Jupyter Notebook servers with GPU support, the aspect of installing GPU drivers, device plugins, and other configurations is left as an operation for the cluster operators to perform.

Pipelines

The Jupyter notebook files are great during the experimentation phase as you're frequently tuning and changing code blocks and seeing the results quickly.

However, during the Production phase it's best to have the entire ML process automated. To that end, KubeFlow Pipelines allows users to model their entire ML workflow: data preparation,

cleanup, training, uploading artifacts, tagging metadata, serving, etc. Each of the steps is modeled as an executable container image or a python function. Then a pipeline workflow descriptor describes the order in which these steps are executed and their dependencies.

These pipelines can be triggered via a CICD pipeline, CLI, or from the Pipelines UI. The UI will also show the progress of each running pipeline, along with the input and output of each step.

Best of all, the entire pipeline can be versioned in a code repository; so at any point in the future, you can deploy the pipeline again and feel confident in being able to reproduce the same results.

KubeFlow

Installing, maintaining, and configuring all the interactive tools and ML automation tools is a significant effort. KubeFlow[8] was invented as a single ML platform to deploy and maintain the ML tools: Jupyter Notebook, Katib, TensorBoard, Metadata, Pipelines, ML Framework Serving components, KFServing, Fairing, Minio, and many more.

During the installation phase of KubeFlow, users designate the ML tools to install - and it does the rest, including installing the tools' dependencies (cert-manager, Istio, Knative, Argo, etc). There are a few caveats and gotchas, especially if you're deploying into a cluster with pre-existing Cert-Manager and Istio and/or if you're running alternative container runtimes like ContainerD. However, as a whole, KubeFlow really simplifies the installation and management of all the tools above.

KubeFlow is also introducing multi-tenancy features to allow multiple teams to use the platform without overstepping on each other.

[8] Kubeflow Overview

How Spectro Cloud Can Help

The world of AI/ML is hustling and bustling. Making Kubernetes infrastructure work well underneath these deployments can also be an undertaking, one that not all groups want to spend a great deal of time concentrating on. From managing day 2 operations to GPU driver installations and management and secrets and volume management for your Jupyter notebooks, many organizations conclude that they want to offload these infrastructure focused concerns, so as to concentrate more on their data science workloads.

As enterprises transform and modernize their technology, Kubernetes will continue to grow as an essential part of these transformations, enabling speed in development. However, Kubernetes itself takes significant effort to make production ready; as an operator, you have to deal with continuous security, Kubernetes upgrade lifecycle, integrations with existing infrastructure services, as well as operating an end-to-end solution. These challenges are difficult as-is with a single cluster, but multiply when managing multiple Kubernetes clusters.

Spectro Cloud brings the ease of a managed services approach to the deployment and management of an organization’s unique Kubernetes infrastructure stacks. These custom stacks can include GPU drivers and operating systems you require for your workload, as well as installation and management of KubeFlow and related out of box integrations. You can focus on data science, rather than infrastructure.