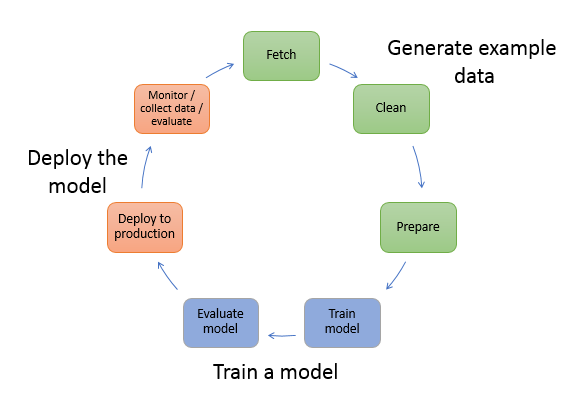
如果你喜欢哲学并且你是一个IT从业者，那么你很可能对软件哲学感兴趣，你能发现存在于软件领域的哲学之美。本文我们就从软件哲学的角度来了解一下亚马逊云科技的拳头级产品[Amazon SageMaker](https://docs.aws.amazon.com/zh_cn/sagemaker/latest/dg/whatis.html)，有两个出发点：一是SageMaker本身设计所遵循的软件哲学；二是从软件哲学的角度我们应该如何使用SageMaker提供的功能。SageMaker是一个全托管的机器学习平台（包括传统机器学习和深度学习），它覆盖了整个机器学习的生命周期，如下图所示：



我们从如下的几个方面来展开讨论：

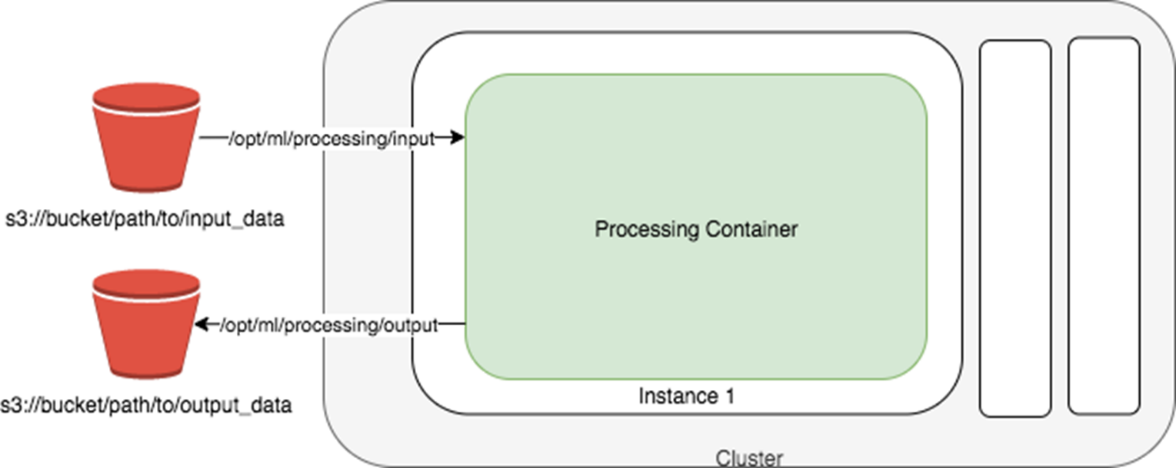
* 天下没有免费的午餐-----------------------------------------------权衡之道
* 简单之美----------------------------------------------------------大道至简
* 没有规矩不成方圆-------------------------------------------------循规蹈矩
* 没有“银弹”------------------------------------------------------对症下药
* 变化之本-----------------------------------------------------------进化本质
* 知其所以然--------------------------------------------------------心中有数
* 保持一致性--------------------------------------------------------质量可控

# 天下没有免费的午餐-----------------------------------权衡之道

软件有很多的品质（品质也叫非功能性需求）：性能（比如时间性能，空间性能，模型性能），可用性，易用性，可扩展性，兼容性，可移植性，灵活性，安全性，可维护性，成本等。一个软件没有办法满足所有的品质，因此我们在和客户交流的过程中，要真的弄清楚客户想要的是什么（没有想象中那么简单），哪个或者哪些软件的品质是客户当前最关心的。很多软件品质经常会互相制约（一个经典的例子就是安全性和时间性能，安全这个品质就是一个让人又爱又恨的东西，一般来说需要加入安全性的时候，在其他上下文不变的情况下，基本上时间性能就会变差了），所以我们需要权衡，而在权衡的时候一定要把握好“度“。

对于SageMaker来说：

* SageMaker Processing job要求数据的输入和输出都需要在S3，基本原理图如下：



SageMaker Processing job提供了托管的单个实例或者集群来做数据预处理，特征工程以及模型评估。如果你的原始数据并没有存放在S3，这个时候你需要**权衡空间性能与可托管性**（可托管性的好处是很多运维的工作就不需要你关心了，交给了亚马逊云科技来运维），是把数据从源拷贝到S3来使用托管的Processing job服务还是就地来用自建的集群来处理；如果你的原始数据本身就存放在S3，那么直接用Processing job来使用Sklearn或者SparkML来进行数据预处理或者特征工程是你的首选。

* SageMaker的内建算法对数据输入格式的要求以及可配置的有限的超参数。SageMaker提供的内建算法（SageMaker对常见的ML任务基本都提供了一种或者多种算法）对数据输入格式的要求以及提供的超参数可能与开源界的算法的数据输入格式和提供的超参数有区别。这里你需要**权衡易用性与灵活性**：如果你只是想实现一个ML的任务并且不想关注算法的实现细节，那么可以优先尝试SageMaker的内建算法；如果你想更深入了解算法的实现细节以及更灵活的超参数设置，那么建议的选择是把你的算法或者开源的算法迁移到SageMaker中。
* SageMaker训练时的HPO自动超参数优化功能的使用。自动超参数优化的初衷是为了减轻算法工程师/数据科学家/应用科学家们手工调参的痛苦。SageMaker的HPO自动超参数优化对于内建算法和非内建算法都支持，并提供了贝叶斯搜索和随机搜索两种方式供你选择。不是所有的算法都需要走自动超参数调优，需要**权衡模型性能（就是指模型效果）与成本**。一般来说，对于深度学习模型或者海量数据集的情况下可能做自动超参数调优的时间代价和成本代价太大。因此在实际的ML项目中，客户很少对深度学习模型或者海量数据集做自动超参数调优；对于传统的机器学习模型并且在数据集不大的情况下，可以考虑用自动超参数调优来找到可能的最优解。
* SageMaker内建的inference pipeline的数据流。SageMaker Inference pipeline可以把多个容器（容器中可以跑特征处理逻辑或者跑模型serving逻辑）串接起来，它的目的是把推理时的特征处理模块和模型串接起来，或者把多个模型做成上下游串接起来。它的数据流是这样的：

SageMaker proxy/agent---🡪first container---🡪SageMaker proxy/agent---🡪second container---🡪………. SageMaker proxy/agent

也就是说，每个容器的输出由SageMaker内部组件做中转，该组件把上一个容器的输出做为新的request发送到下一个容器。通过使用Inference pipeline这个功能可以简单方便的实现模型的上下游串接或者特征处理模块和模型的串接，但从上面的数据流可以看到会带入一些延迟，这个时候你需要考虑延迟是否在可以接受的范围内并使用Inference pipeline，也就是需要**权衡易用性与时间性能**。

* SageMaker中对于Tensorflow和Pytorch两种框架都提供了多种训练方式。训练方式包括开源框架原生的方式以及SageMaker专门实现的针对这两种框架的数据并行和模型并行两种方式。SageMaker的数据并行训练方式适合每个GPU卡可以跑完整的模型训练但是数据集是海量的情况；SageMaker的模型并行训练方式适合单个GPU卡无法直接跑模型训练的情况（比如模型太大了）。也就是说，在海量数据集大规模训练或者超大模型训练的场景，使用SageMaker的这两种专有的训练方式比框架原生的训练方式会更高效，但是使用SageMaker的数据并行和模型并行的训练方式的话，对于框架的版本有要求并且需要一定的代码修改，因此需要你**权衡代码的可移植性与时间性能**。

# 简单之美-------------------------------------------------大道至简

“简单”可能的含义有很多，比如精简，简朴，可读性好等。“简单”的度量标准可能每个人的理解都不一样，但是一个通用的原则是，“您”在设计软件的时候尽量多想着：“软件需要别人来用，还需要别人来迭代和维护”，您一定要高抬贵手。“简单”的对立面就是“复杂”，业界的共识是通过降低复杂度来得到高质量长生命期的软件，而如何降低复杂度是每个软件设计人员以及开发人员无时无刻需要关注的事情。

在SageMaker中的体现：

* SageMaker是基于container的设计，到目前为止没有选择Kubernetes。在当前业界大兴Kubernetes的情况下，SageMaker并没有随大流。Kubernetes的功能很强大但是很复杂，对于SageMaker来说，很多Kubernetes的功能用不上，那么为了减少软件依赖以及降低复杂度，SageMaker选择了更轻量的设计（**杀鸡真的没有必要用牛刀**）。
* SageMaker high level API（high level API指的是SageMaker Python SDK，这个API的使用习惯类似常见的ML框架比如Sklearn）设计很简洁，类层次也很清晰（分层就是一种降低复杂度的方法），很多feature通过简单的参数设置就能搞定。比如通过简单的设置distribution参数就把底层复杂的分布式环境部署隐藏掉了（信息隐藏也是降低复杂度的一种方法），让API调用者的关注点更集中在训练脚本本身；比如简单的设置模型的S3保存位置，SageMaker就会帮助你在训练结束或者训练中断时把对应目录下的模型压缩打包并上传到S3指定路径；比如通过设置git\_config参数，你就可以直接用github中的代码在SageMake中来训练，而不需要你关心代码的搬迁过程。
* SageMaker提供了多种算法选择：内建算法，BYOS（基于预置的机器学习框架来自定义算法），BYOC（自定义算法设计并自己来打包容器镜像）和第三方应用市场（在AWS Marketplace中挑选第三方算法包，直接在Amazon SageMaker中使用）。而BYOS和BYOC是SageMaker中实际用的最多的两种选择。那如何选择BYOS和BYOC？总的来说，**优先看BYOS是否能满足需求**。BYOS相对于BYOC要容易，需要迁移到SageMaker的工作量也少。而选择BYOC，常见的是如下的情景：

|  |  |
| --- | --- |
| 情景1 | SageMaker中的内置框架的python版本不是你需要的版本 |
| 情景2 | 你需要一个完全不同于SageMaker的那些内置框架比如paddlepaddle |
| 情景3 | 有些客户习惯使用基于docker image的容器跑ML，那么BYOC可能对他们来说比较容易过渡。 |
| 情景4 | 有些客户代码分为两部分：底层基础平台级别代码，上层用户定制代码。底层代码打包为docker image并push到ECR以BYOC方式跑通。上层用户指定docker image为上面打包好的image，然后跑自己的定制代码。这样做的好处是，**代码管理分离**，不会发生纯BYOS方式上层用户误修改底层代码的问题。 |
| 情景5 | BYOC一次性安装了相关的软件包；如果用到的软件包并不在SageMaker内置的容器镜像中，BYOS每次都有安装软件包的过程。 |

除了上面这些情景，尽量优先考虑BYOS的方式，它使用方式简单，学习曲线也短。

* SageMaker 提供了两个变量sagemaker\_program和sagemaker\_submit\_directory来帮助你轻松的完成BYOC的调试。前者告知SageMaker把这个参数的值作为user entry point（就是用户提供的需要SageMaker调用的脚本），后者则是这个entry\_point对应的代码以及它所依赖的代码打包后的S3路径。通过设置这两个参数，在调试代码的时候只是需要把修改后的代码重新打包上传就可以，而不是每次都build docker file，简单方便而且很节省时间。

# 没有规矩不成方圆---------------------------------------循规蹈矩

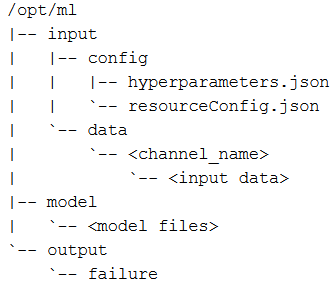
拥有丰富经验的你可能听过或者践行过契约式编程，而契约式编程简单说就是，你需要按照对方的一些约定来coding。一般来说，只要是提供给别人使用的软件/工具，或多或少都会有一些约定。SageMaker从**尽量减少代码侵入性和最小代码迁移工作量**的思路出发，提供了很多约定。

在SageMaker中的体现：

* 训练时，数据Channel相关的约定：

|  |  |
| --- | --- |
|  | 介绍 |
| Channel命名 | 使用SagemMaker API设置数据channel的时候，channel名字你可以随意选取比如名字取为“train”，然后通过SageMaker设置的环境变量SM\_CHANNEL\_{channel\_name}（其中的{channel\_name}换成你设置channel的名字，“train”对应的就是SM\_CHANNEL\_TRAIN）就可以获得channel中数据的本地路径“/opt/ml/input/data/train”。SageMaker会把数据集拷贝到这个约定好的路径，那么你的程序只有遵守约定才能读取到需要的数据。 |
| Channel顺序 | 环境变量中多个channel的名字的顺序与调用SageMaker estimator fit API时写入的顺序是不同的。比如对于在fit API时设置{'training':train\_s3, 'training-2':train2\_s3, 'evaluation': validate\_s3}这样的三个channel，环境变量SM\_CHANNELS被设置为['evaluation', 'training', 'training-2']，也就是说最后一个channel 'evaluation'出现在环境变量SM\_CHANNELS中的第一个，其他channel则是按照原来顺序排列，在训练脚本中读取数据的时候一定要注意这个细节，否则会出问题。 |

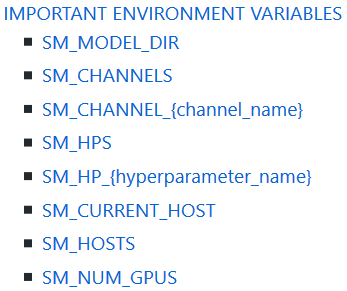
* 训练容器本地路径相关的约定，如下图所示：



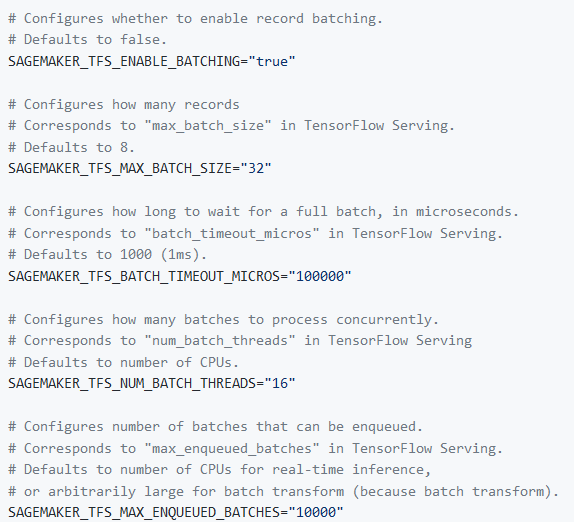
我们重点关注下表中的四种路径（除了下面这些路径，训练过程中放置在其他路径下的文件，在训练结束后都会被丢弃）：

|  |  |
| --- | --- |
| 路径 | 介绍 |
| /opt/ml/model | 这里存放最终的模型文件。  被SageMaker上传时机：训练结束（这个路径下的所有文件会被压缩打包后上传）。 |
| /opt/ml/output/data | 这个一般存放的是训练过程中的一些和模型文件以及checkpoint没有关系的文件（比如验证集评估结果文件）。  被SageMaker上传时机：训练结束（这个路径下的所有文件会被压缩打包后上传）。 |
| 配置的checkpoint local path路径 | 这个是为了让SageMaker来帮助你自动上传checkpoint或者在开始训练任务前把checkpoint从S3下载到这个路径。  被SageMaker上传时机：训练中近实时上传（但是这些文件并不压缩和打包）。 |
| 配置的tensorbaord log local path路径 | 这个是为了让SageMaker来帮助你自动上传tensorboard的log到指定的S3路径。  被SageMaker上传时机：训练中近实时上传。 |

* SageMaker给容器提供了很多方便使用的环境变量，包括SageMaker相关的和内建框架相关的。比如SageMaker相关的一部分环境变量如下：



SageMaker内建的TF serving框架的service side batch相关的环境变量如下：



* SageMaker内建算法对输入数据格式的要求。SageMaker内建算法对输入数据的格式要求可能和开源算法对数据格式的要求不同，如果不注意这个，在调试模型的时候可能会出现比较奇怪的结果。比如SageMaker 目标检测算法对BBOX的格式要求如下：对于json格式的输入标注文件， 要求的坐标格式是 [top, left, width, height]，如果你的数据集的标注是PASCAL VOC格式的（xmin，ymin，xmax，ymax）或者是COCO格式的（x，y，width，height），都需要做不同的转换；对于recordIO格式的输入文件， 要求坐标格式是相对坐标，[xmin/width，ymin/height，xmax/width，ymax/height]。
* Spot实例与SageMaker checkpoint机制的配合。为了节省成本，使用spot实例进行训练是首选。为了让spot实例被回收对你的训练任务造成的影响最小化，SageMaker通过两个参数checkpoint\_local\_path和checkpoint\_s3\_uri来助你一臂之力（当然你不使用spot实例，也仍然可以利用SageMaker的checkpoint机制）。这样训练job被spot回收中断以后并自动重新开始训练后，就不用从头开始训练了，而是从最新的checkpoint开始接着训练（SageMaker提供了checkpoint上传和下载的机制，你需要修改你的代码来配合，也就是你需要从约定的checkpoint local路径来初始化你的模型参数，否则是空谈），从而在节省成本的同时节省训练时间。

# 没有“银弹”--------------------------------------------对症下药

可能我们自己脑海中或者遇到过别人问我们如下的问题：对于XX任务当前哪个模型效果最好？使用AutoML是不是就不需要我们做特征工程和样本工程了？自动超参调优是不是就彻底解放我们的手动超参调优了？如果真的是这样的话，那就太美好了。在软件界，“没有银弹”这句话流行很久了，对于人工智能领域也是同样道理，都需要case by case来分析每一个目标任务。

现在Bert以及bert-like的模型比如GPT3，T5等很火，恨不得只要是NLP的任务都用这样的模型。我觉得还是应该至少要考虑具体的目标任务是什么，目标任务的建模难度这两个因素来进行模型选型。如果在语料充足的情况下，做一个简单的文本分类任务，这个时候可能用一个简单的模型比如fasttext来就够用了，而不用bert模型做fine tuning这么复杂；如果是针对用户评论做细粒度情感分析任务，这个任务就很复杂了，用简单模型就可能不合适了，这个时候用比如T5这样的复杂模型才合适。**盲目的追新和追热点，受伤的是你的项目**（可能你能从中受益）。

对于SageMaker来说：

* SageMaker有内建算法，BYOS，BYOC和Marketplace算法可供你选择，总有一款适合你。很有意思的一个现象是，SageMaker在刚发布的时候bulid了17种内建算法，很多年过后一直也没有在增加新的内建算法。我猜测SageMaker的开发团队会认为，即使不断的增加一些内建算法，也没有办法及时对主流的一些算法进行跟进。正是因为针对任何一种细分场景，没有包治百病的“算法”，SageMaker就不在内建算法上花费更多的时间和精力，它提供更灵活的BYOS和BYOC让客户把开源的算法方便的迁移过来，或者通过Marketplace让买家和卖家都能尝到使用算法的甜头。
* SageMaker提供了Autopilot和Auto model tuning（即自动超参数调优）这样两种AutoML机制。AutoML一直是一个很热门的研究方向，也是我们人类很期待的一个能大量落地的方向（谁不喜欢简单省事就能完成一项任务呢？）。如果每个目标任务都可以用AutoML来解决的很好，那么大部分人都可以腾出时间来攻克别的技术难题了。虽然Autopilot可以直接对结构化数据来建模，它也能自动做一些特征处理，但是它并不是银弹，不是什么数据集直接丢给它就能出一个不错的效果的；要使用Autopilot，自己提前做一些特征工程可能效果会更好（比如特征缩放，特征生成，特征交叉，甚至不同的缺失值处理方法，异常值处理等）。

而对于自动超参数调优，如果使用默认的超参数搜索空间，时间成本和金钱成本太大，那么还是需要人工首先限定每个需要搜索的超参数的区间的左右端点，同样这里没有“银弹”，左右端点的确定要么根据已有的经验，要么就是通过实验来大致选取。**一定不要无条件的使用Autopilot或者自动超参数调优来解决你的问题**，三思而后行！

# 变化之本------------------------------------------------进化本质

为什么要考虑“变化”？设计之初就应该考虑到将来的可能变化，也就是说系统框架要设计的比较有弹性（就像AWS的很多服务那样弹），对于将来的需求的改动不会付出很高代价。在软件设计中，经常会谈到“面向变化编程”，即永远不要假设需求不变，现实中需求大大小小经常变。**变化是创新的必经之路，永恒不变的东西只有变化**。

在SageMaker中的体现：

* SageMaker早期的版本提供了SageMaker-container包供你使用来创建SageMaker兼容的容器和自定义的框架。后期的版本，为了让基于SageMaker-container包的容器镜像尽量更小更内聚，SageMaker把这个SageMaker-container包拆分为sagemaker-training toolkit（专为训练的容器）和sagemaker inference toolkit（专为推理/serving的容器）两个包来瘦身。
* 随着不断的进化，SageMaker现在是一个完全自洽的全生命周期的机器学习平台。在早期的时候，SageMaker只有三大核心功能：训练，推理（离线推理和线上推理），notebook实例。为了能把ML生命周期中的数据预处理，特征工程，模型评估这些功能也纳入，SageMaker后续推出了Processing job来做这些事情。而随着很多客户对于多种机器学习任务的高质量标注需求的上升，SageMaker推出了带有人工标注和机器自动标注的Ground Truth功能（这里又体现了客户至上的企业文化）。而随着SageMaker studio（它是用于机器学习的集成式开发环境 IDE，可让你构建、训练、调试、部署和监控机器学习模型）的推出，以及MLOps的更多功能的加入，现在的SageMaker变成了“超人”（短时间能增加如此多的功能并且还保持健壮，正是因为SageMaker的设计基因就是面向变化的）。

# 知其所以然------------------------------------------做到心中有数

我们可能知其然，但是所以然知道了吗？可能有些感兴趣的东西我们会去了解其深层的原因，但是软件的问题我们去研究了吗？软件是枯燥的，很多时候我们都是作为谋生的手段来应付之，因此不知所以然也就很正常了。但是如果您是要真正的学习东西，或者更好的服务于客户，最好还是”再深一点“。

对于SageMaker来说：

* SageMaker相关的代码比较分散，为了满足好奇心可以去阅读源码。比如有SageMaker平台相关的开源代码包sagemaker-container，sagemaker-training，sagemaker-inference；与内建框架相关的开源的代码比如SageMaker tensorflow training，SageMaker tensorflow serving；SageMaker Python SDK的开源实现代码。通过阅读这些代码，你会对SageMaker如何工作有更深刻的理解。
* 当训练文件的数量比较多的时候，SageMaker pipe mode和file mode哪种方式训练 更快呢？拿Tensorflow的tfrecorddataset API来举例，在其他的dataset API基本一样的前提下，file mode更快（这个在多个实际客户项目中测试过）。主要的区别就是pipemodedataset API和tfrecorddataset API。tfrecorddataset API可以设置num\_parallel\_reads来并行读取多个文件的数据，还可以设置buffer\_size来优化数据读取。Pipemodedataset API则没有类似上面的参数来加速数据的读取。也就是说pipe mode更适合读取文件数量不多，但是每个文件都很大的场景。
* SageMaker Endpoint for TFS vs for Mxnet/Pytorch的内建serving框架复杂性对比。SageMaker Endpoint for TFS的介绍如下：

|  |  |
| --- | --- |
|  | 介绍 |
| 架构拓扑 | Nginx----->guincorn(optional)----->TFS |
| guincorn启用条件 | 取决于是否设置了定制的 inference.py用于数据预处理和后处理，或者是否启用了MultiModel |
| Ping健康检查的传递路径 | 不管有没有启动guincorn进程，SageMaker发给Nginx的ping健康检查，Nginx都会直接发给TFS进程。 |
| 架构如此复杂的原因 | TFS不支持/ping REST API；TFS不支持钩子函数（钩子函数用来作输入的预处理和输出的后处理） |

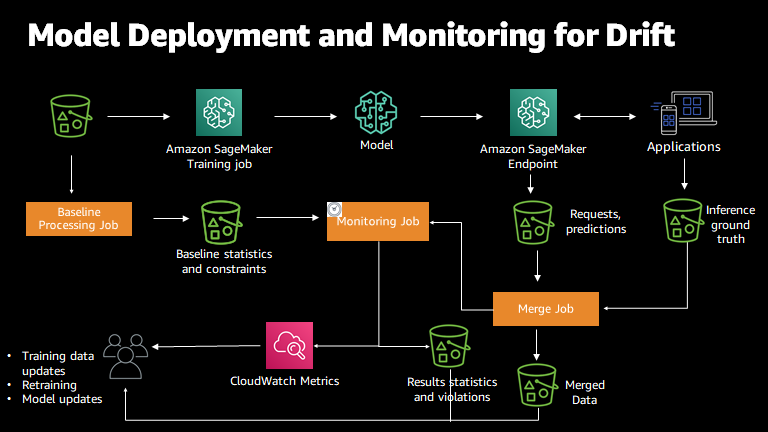
SageMaker Endpoint for Mxnet serving的介绍如下（SageMaker Endpoint for Pytorch serving是类似的）：只使用一个组件mxnet model server，它的特点是，直接支持钩子函数；支持处理/ping REST API；缺省会使用所有的GPU来做推理，单个TFS进程只使用一个GPU来推理。

# 保持一致性----------------------------------------------质量可控

一致性是降低系统复杂度有利的手段。如果一个系统保持一致，意味着类似的事情用类似的方法去做，降低了认知负荷。在ML机器学习领域，我们经常会谈到一致性，比如效果线上线下一致性（如果模型离线效果好，模型上线以后表现不好，这就是发生了效果线上线下不一致），特征的线上线下一致性（特征的线上线下不一致是效果线上线下不一致的一个常见原因；特征的线上线下不一致指的是线下训练时样本中的特征的特征值可能会发生变化，并不和该样本在线上生成时的特征值完全一样。关于特征的线上线下一致性更详细的讨论请参考我的另一个[文章](https://github.com/yuhuiaws/ML-study/blob/main/%E6%8E%A8%E8%8D%90%E7%B3%BB%E7%BB%9F%E4%B8%93%E9%A2%98/%E6%8E%92%E5%BA%8F%E6%A8%A1%E5%9E%8B%E7%9A%84%E6%A0%B7%E6%9C%AC%E5%B7%A5%E7%A8%8Bby_%E6%A2%81%E5%AE%87%E8%BE%89.docx)）。**保持一致性是模型质量可控的一个重要因素**。

对于SageMaker来说：

* 使用SageMaker Processing job对训练集做的一些特征工程比如某个特征Z-score标准化，那么为了让预测时与训练时有一致的特征工程，需要如何处理呢？在对训练集做了某个特征的Z-score标准化以后，用到的该特征的均值和方差这些metadata需要保存起来（SparkML和Sklearn都有相应的API把这些metadata以文件的形式保存起来）；然后利用SageMaker Inference pipeline在第一个容器中把之前保存的metadata文件加载进来对原始特征进行Z-score标准化处理之后再送入第二个容器即模型推理容器。
* **如果在模型serving的时候，能及时知道每个特征的分布是否和训练时的数据集尤其是验证集的分布是否基本一致，对于模型才更可控**。而SageMaker的model monitor的一个重要功能就是监控特征的统计漂移，如果模型在生产过程中接收到的数据的统计性质偏离了训练所依据的基准数据的性质，则模型将开始失去其预测的准确性。Model Monitor 使用规则检测数据漂移，并配合AWS其他服务在发生数据漂移时向您发出警报。下图说明了此流程的工作方式：



# 总结

本文从软件哲学角度来介绍了SageMaker的一些设计思想以及如何使用SageMaker的一些功能。总体来讲，不管你是否考虑采用SageMaker作为你的机器学习平台，至少它的这些实现的思路以及设计哲学都是可以用来参考的。SageMaker作为全球使用量最大的机器学习平台，是值得你花时间来好好研究和探索以及实践的。关于SageMaker更多详细和更多深入的内容请参考我的[github](https://github.com/yuhuiaws/ML-study/tree/main/AWS-Sagemaker%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0%E5%B9%B3%E5%8F%B0)。感谢大家的耐心阅读。