WHITEPAPER

Productionizing Machine Learning Models with Containers

ALGORITHMIA

TABLE OF CONTENTS

ABOUT ALGORITHMIA:

Algorithmia empowers every developer and company to deploy, manage, and share their AI/ML model portfolio with ease. In addition to the Algorithm Marketplace, Algorithmia uses the serverless AI Layer to power our Hosting AI/ML Models and Enterprise Services.

Our platform serves as a data connector, pulling data from any cloud or on premises server. Developers can input their algorithms in any language (Python, Java, Scala, Node, Rust, Ruby, or R), and a universal API is automatically generated.

Democratizing access to algorithmic intelligence.

Sign-Up for Free or Schedule a Demo Today

Introduction

This whitepaper explains the powerful new paradigm of containerization, walks through setting up your Machine Learning models in Docker and Kubernetes, and considers why an AI Layer might be a strong fit for your organization.

The question of investing in Machine Learning has shifted from "when" to "how" – business leaders are all in on the new technology in theory, but are starting to see the complications of building a scalable Machine Learning pipeline in practice.

> *72% of business leaders have indicated that they view AI as a business advantage (PwC).*

Machine Learning models that only exist as files on your computer aren't very valuable: your predictions are only as good as their practical use. And while it has been getting significantly easier to *train and develop* accurate Machine Learning, **deploying those models** – especially at scale – is still a major challenge.

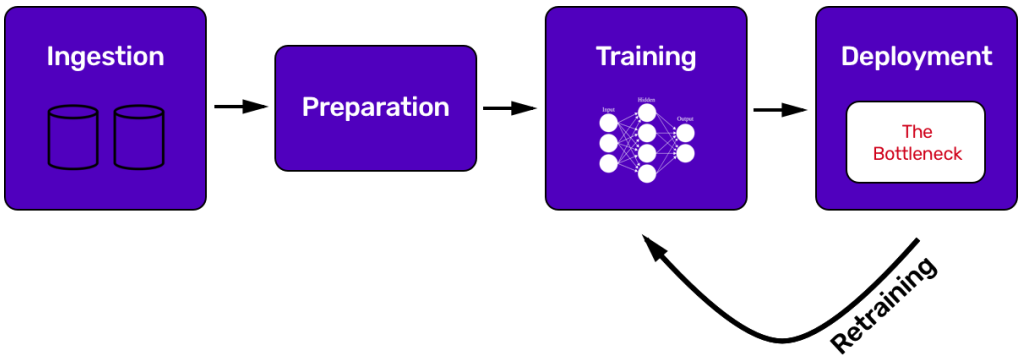

Figure 1: A Machine Learning Pipeline

In approaching the process of taking your Machine Learning into production, there are two major options: a bespoke, manual deployment, or a platform that automates the whole process. We'll dive into the complexity of a manual deployment using a containerized architecture, and see how the right platform can greatly simplify the challenge you're facing.

The Paradigm of Containerization

The cloud infrastructure that so many companies rely on today – Amazon Web Services, Microsoft Azure, Google Cloud Platform, and others – is built on idea of the virtual machine. Instead of each customer requiring their own specialized hardware, the virtual machine creates a number of instances of a "server" – so to say – on one actual physical server. That means that multiple operating systems can be running on the same hardware at once, with only an extremely remote security risk of each affecting its fellow instances on the same physical server. You can almost think of it as having your own apartment in an apartment building.

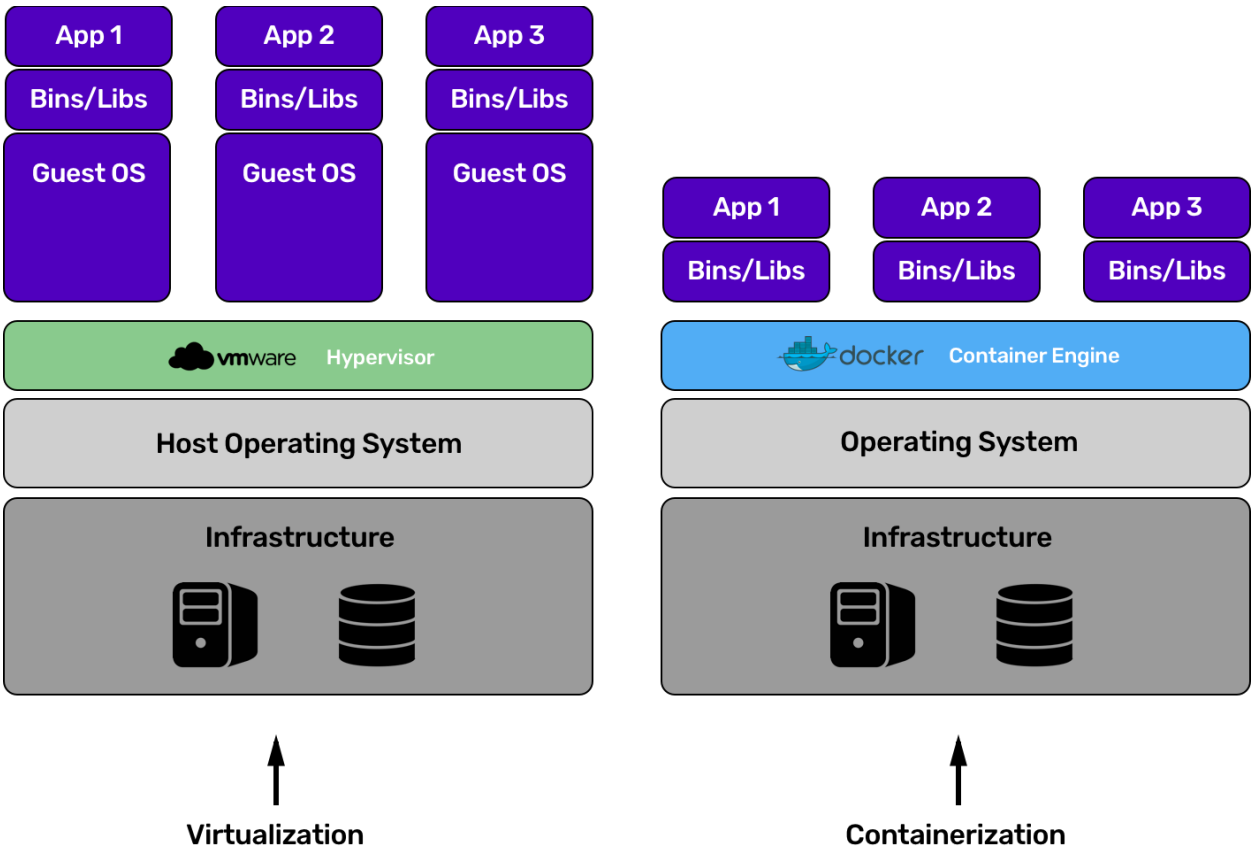

Figure 2: A virtualized architecture vs. a containerized architecture

Containers take that idea to the next level: they run multiple instances of an application on top of one operating system, of which there could be multiple operating systems on one

physical server. It's a way of further separating each instance from the lower level parts of infrastructure, and ensuring that they aren't able to negatively impact each other. It's sort of like living in an apartment with roommates, but having your own bathroom.

Containers offer a number of major benefits when compared to just using virtual machines:

- Without the need for a unique operating system, startup and shutdown are much faster than for a virtual machine.
- Given size and configuration, it's easy to share your container templates (images) with other developers.
- A supportive and helpful community has emerged around containers, with loads of open-source ideas to draw from.

There are two parts of getting your application running with containers:

- The containers themselves
- A system to orchestrate and manage all of those containers

Creating your container image and getting it running can often be easy enough, but things can get difficult when it comes time to organize and dynamically scale all of the pieces of your application.

Docker – for creating and running containers
The most popular framework for creating containers is called Docker. Started as an internal project in France at a Platform-as-a-Service company (dotCloud), Docker debuted publicly at PyCon in 2013, and has taken the open-source world by storm ever since then. The Docker Engine allows you to create container images and deploy them on top of an operating system, taking care of all of the operating-system-level virtualization. There are also a number of ancillary Docker services to help, like Docker Hub for sharing images.

Kubernetes — for orchestrating and managing containers
Once you've created your container image, orchestration is where it starts to get really technically challenging. Originally announced by Google in mid-2014, Kubernetes is one of the more established frameworks for managing and deploying containers (like Docker ones). Kubernetes allows you to handle networking, scheduling, and management of your container deployment.

Setting up Your Model in a Docker Container

Getting up and running with Docker for Machine Learning has three major steps: finding an image, building your model inside of it, and then creating a Dockerfile from your work.

1) *Finding an image*

In Docker, an image is like a template: you create many identical containers from one image. You can think of it as a floor plan for a particular apartment in a building. All of the F units in a building – 3F, 7F, 38F – will look identical, since they're built from the same image.

Your image needs be be tailored to your goals. A popular example is TensorFlow, the ubiquitous open-source Google framework used to build computational data graphs. There's already a TensorFlow image on Docker Hub for you to build your own TensorFlow projects in. To download the standard TensorFlow image, you use the `docker pull` command, followed by the URL for the download.

```
MacBook-Pro-5:Desktop gagejustins$ docker pull gcr.io/tensorflow/tensorflow
Using default tag: latest
latest: Pulling from tensorflow/tensorflow
1be7f2b886e8: Pull complete
6fbc4a21b806: Pull complete
c71a6f8e1378: Pull complete
4be3072e5a37: Pull complete
06c6d2f59700: Pull complete
20a601a42386: Pull complete
d8967df06d5c: Pull complete
adbe4dda11a0: Pull complete
eeb8b3ca49ee: Pull complete
ab96b2cecaaa: Pull complete
e61c2ef48dde: Pull complete
50042b70c2f5: Pull complete
Digest: sha256:188bcda72801c3b756d483e3110a994567f7e3d5f197860279ae68cd2a94f97c
Status: Downloaded newer image for gcr.io/tensorflow/tensorflow:latest
```

Figure 3: Downloading the TensorFlow Docker image

2) *Building your models inside the image*

Once you've downloaded the TensorFlow image, you'll need to create a container (an instance of the image) for you to write your code in. The `docker run` command builds a container from the specific TensorFlow image and runs it on the specific port (8888), which will put you in a Jupyter Notebook (where Data Scientists are most comfortable).

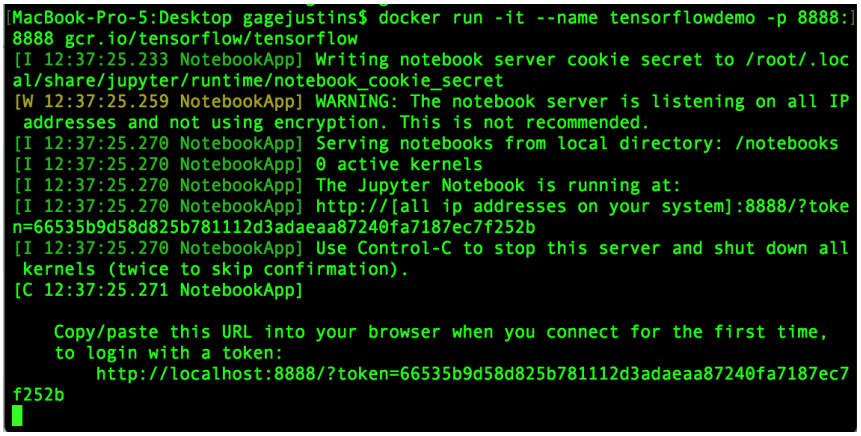

Figure 4: Creating a Docker container from the downloaded TensorFlow image

Whenever you're finished modeling, you'll want to save your model (in some frameworks, your source code as well) for insertion into your custom image.

3) Creating a Dockerfile

A Dockerfile is a set of instructions for Docker to follow. For deployment, you'll tell Docker to download an image (TensorFlow in our case), copy over your application code into the image, and deploy it as an instance in a container. You'll need to navigate a bunch of complexity around your model's dependencies, and add those to the Dockerfile as well. A sample might look like this:

```
FROM tensorflow
LABEL maintainer "Algorithmia <dev@algorithmia.com>"
RUN mkdir /model
WORKDIR /model
COPY . /model
RUN pip install requirements.txt
```

As you get deeper into deploying with Docker, you'll find other elements you'll need to deal with. If you want a database integrated with your model in any way, you'll need to use Docker Compose and create those images separately. Maintaining security credentials is also a challenge, since you likely shouldn't be using them as plaintext on the command line.

Orchestrating Your Containers with Kubernetes

If you think of Docker as getting you to the point where you can build individual apartment units, you can understand that there's more to do: you need to build a whole building. That means making sure your apartments work well together, ensuring that you have the manpower to build a ton of apartments over time, and dealing with the plumbing and electricity behind the scenes.

This is where Kubernetes comes in: it's a tool to instantiate, schedule, and scale up or down your containers. You also need an orchestration system like Kubernetes to link containers to each other (plumbing) and expose them to outside interfaces. It's worth noting that Kubernetes is one of multiple orchestration solutions, and Docker even has its own called Swarm.

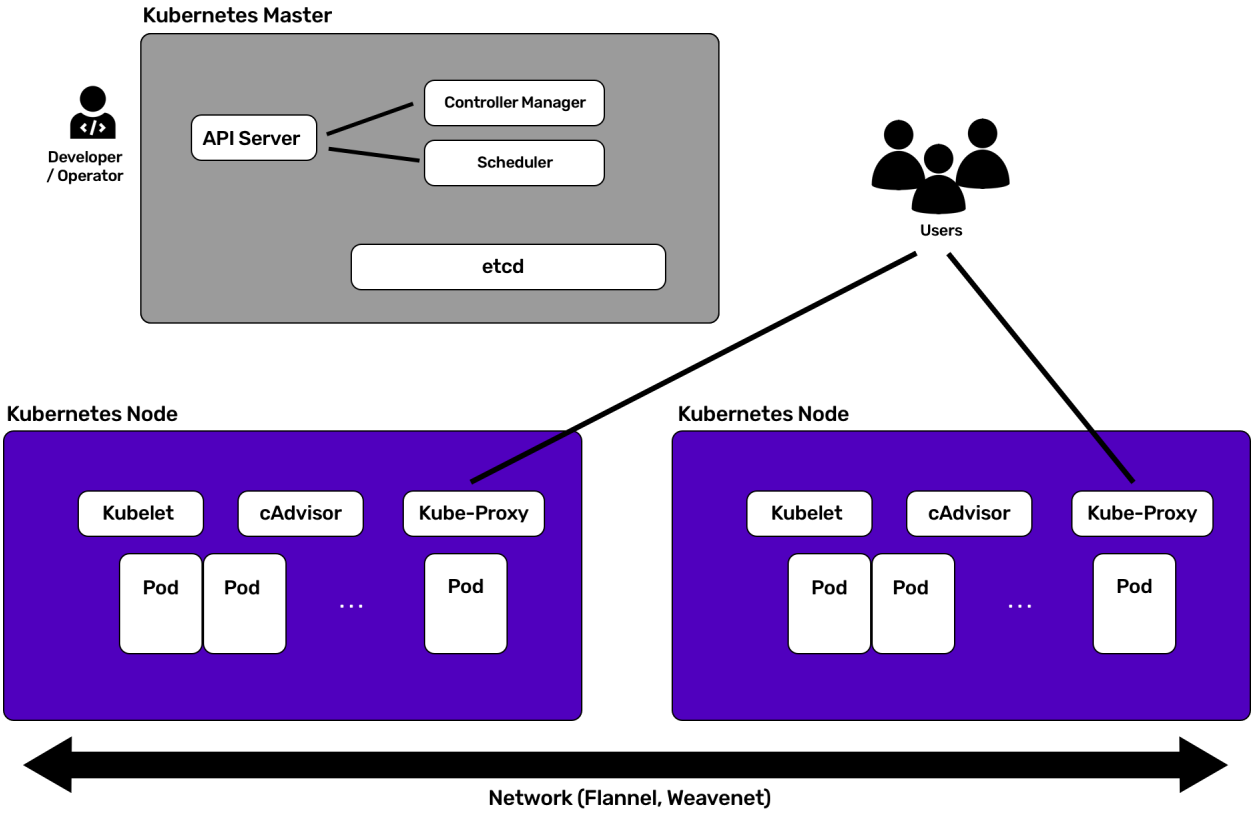

Figure 5: How Kubernetes works: master and nodes

Getting your Docker images ready to go wasn't the most technically challenging task, but Kubernetes gets into some real complicated territory. The framework has a long list of components that you need to master to create a production grade deployment.

Kubernetes uses a Master-Node architecture, where one server runs the show, while creating and running other nodes. A node is just another server, and a cluster will have many of them (and one master, with a backup or two). Part of the value Kubernetes adds is that you don't need to tell it which nodes to run which containers on: it takes care of that automatically.

The master makes decisions about your cluster, like scheduling and responding to cluster events. You'll need to become familiar with Kubernetes specific elements like etcd, kube-apiserver, kube-scheduler, and cloud-controller-manager. Nodes contain components like the container runtime (Docker), kubelet, and kube-proxy. There's also the concept of a Pod, which is a group of containers that can interact as if they're on the same machine.

There's a lot here, and it's not hard to see why deployment is a full time discipline and vertical in of itself. Running an application that works at scale with Kubernetes, especially when the application relies on Machine Learning, is very much a team effort and a significant undertaking. Even managed Kubernetes services – like AWS's EKS or Azure's AKS – mostly help with availability, not abstraction.

The AI Layer: Automate Your Deployment and Avoid the Complexity of Manual Orchestration

Even if you have a team with the right expertise, deploying your Machine Learning manually in a containerized architecture is tedious and challenging; especially when significant scale is a constraint. When you consider the need for contextual functionality, like model versioning, interaction among multiple Data Science languages, and algorithm management, the deployment process can become a months-long affair. Using a Platform-as-a-Service like Algorithmia can turn that months-long process into one that takes minutes.

Algorithmia deploys your Machine Learning models as Microservices with API endpoints, along with clients available in all major programming languages. You can write your models in whatever language you want, call them in whatever language you want, and be sure that they'll all deploy together seamlessly.

For Management	For Data Scientists	For DevOps and Engineers
Final step on the road to ROI for Machine Learning	Use the language(s) you want	Massively parallel computing
Feel safe with robust security and management features	Save time by pipelining	Only pay for what you use
Keep Data Science focused on their core competencies	No more DevOps required	CPUs and GPUs optimized for Machine Learning
Pay as you go model for elastic scaling	White-glove support	Serverless architecture that scales to meet your needs

> *"As someone that has spent years designing and deploying Machine Learning systems, I'm impressed by Algorithmia's serverless microservice architecture – it's a great solution for organizations that want to deploy AI at any scale."* – Anna Patterson, VP of Engineering, Artificial Intelligence at Google

With Algorithmia, you get all of the benefits of a scalable cluster run on Kubernetes and Docker (in fact, that's the backend that we use) with none of the complexity of running it yourself. It's as simple as a git push for your Data Scientists.

We believe that every application will have an AI Layer – a platform that stands between the data backend (data and models) and the end user application. The AI Layer helps cross the river and makes it easier to create feedback loops between data and use case.