AUGUST 2023

IT & DATA SCIENCE

Best Practices for Building Containers for Machine Learning

Table of contents

Introduction

In the world of machine learning, containers have become a crucial tool for simplifying model training and deployment workflows, managing dependencies, and ensuring reproducibility. A memorable quote from one of our 'Beers with Engineers' online events captures this sentiment: "If you manage not to use Docker when building a machine learning system, hats off, I would not know how I could dodge that" exclaimed Simon Stiebellehner, ML platform Engineering Manager. This quote emphasizes Docker's pivotal role, countering the occasional hesitation among data scientists to embrace it.

However, building containers for machine learning requires careful consideration of various factors to optimize performance, efficiency, and maintainability.

In this blog post, we will explore the best practices for building containers, some of the practices are more general and relevant for any system using containers, while others are specific for machine learning workloads.

Best practices for building containers for machine learning

Understanding Containers

Understanding Containers
A container image is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools. Container images consist of multiple layers, where each layer represents a specific modification or addition to the previous layer. This layering system allows for easy sharing and distribution of images, as only the changes need to be transmitted, reducing download times and saving storage space.

Building Docker Images with Dockerfiles
Scaling a single job to tens or hundreds of high-end, interconnected GPUs to do things like distributed model training can be problematic. This is because high-end GPUs are not as abundant as lower-end ones, and availability can often be constrained.

This is especially true when close proximity between the machines is required for attaining maximum performance. Even large cloud providers may struggle to provide this level of resources, especially when many clients are demanding the same resources simultaneously, such as during a GPU shortage.

Understand ENTRYPOINT and CMD
It's crucial to comprehend the difference between ENTRYPOINT and CMD as they dictate which command is executed, when the container is launched. The ENTRYPOINT specifies the command to be run, and CMD sets default arguments for the command. The combination of these two determines the actual command that will be executed when the container runs.

Working with Kubernetes and Docker
When working with containers in Kubernetes, being able to understand how ENTRYPOINT and CMD are utilized is vital. Kubernetes will prioritize the command and arguments (defined within the Pod specification) over the values specified in the container image. Ultimately, the command and arguments in the Kubernetes YAML file will override the values defined in the Docker image.

Best Practices

1. **Utilize Existing Images**
When building containers for machine learning, leveraging existing images as much as possible is key. Instead of starting from scratch, find well-maintained base images that suit your needs. Popular repositories like Docker Hub and Nvidia NGC Catalog offer a plethora of pre-built images for various frameworks like TensorFlow, PyTorch, and Jupyter.

2. **Optimize Tagging Strategy**
Plan and implement a well thought out tagging strategy, to keep track of container versions effectively. Consider using meaningful tags that convey essential information about the container version, such as software versions, updates, or relevant metadata. When choosing a container, avoid using the "latest" tag, as it may be continuously changing, which may lead to unexpected behavior during deployment.

3. **Choose a Building Approach**
Two common approaches for building containers are "Build Per Code Update" and "Base Image with Startup Scripts for Customization." The first involves building a new container for every code update, which can lead to image overload and larger images. Whereas the latter involves creating a base image and using startup scripts for customization, resulting in smaller, more efficient images that are easier to manage, and update.

4. **Cross-Platform Considerations**
If your container needs to run on different architectures, ensure it is cross-platform compatible. Use BuildX, a cross-platform build tool, to create containers that work seamlessly across different architectures, such as Intel and ARM.

5. **Optimize Layers for Faster Build Times**
Optimizing layers can significantly improve build times. If a layer changes frequently, place it at the end of the Dockerfile to minimize the number of layers that need rebuilding. This strategy reduces the time required to install dependencies and speeds up the build process.

6. **Prioritize Efficiency**
When building containers for machine learning, it's important to strive for efficiency. Avoid installing unnecessary packages or dependencies. Use multi-stage builds to separate the build environment from the production environment, reducing the image size. Opt for lightweight base images to keep the container as compact as possible, ensuring efficiency.

7. **Create Customized Containers**
When building containers for machine learning, customization is key. Instead of creating a one-size-fits-all container, tailor each container to specific ML use cases. For example, have separate containers for training and inference, each with their specific dependencies and configurations. This approach reduces the container's size and complexity, leading to better performance and efficiency.

8. **Use Environment Variables and Arguments**
Leverage environment variables and command-line arguments within the container to enable flexibility. This allows you to modify the container's behavior without changing the underlying image. Environment variables can control which dependencies are installed, what code is executed, or even the entire entry point of the container.

9 **Leverage Versioning**
Versioning is essential in ML containers to ensure reproducibility and consistency. Specify the exact versions of software libraries, dependencies, and models used in the container to avoid any compatibility issues. Use version control systems like Git to manage your Dockerfile and easily roll back to previous working versions if needed.

10 **Automate Building Process**
Automate the container building process to ensure consistency and efficiency. Use build scripts and continuous integration tools to automatically build and update containers when changes are made to the codebase or dependencies.

11 **Optimize Container Size**
Containers can quickly become large and unwieldy, especially when dealing with machine learning models, datasets and dependencies. It's crucial to optimize container size by removing unnecessary packages, compressing files, and using lightweight base images. Avoid including datasets or large files in the container itself; instead, fetch them from external sources during runtime.

Conclusion

Building containers for machine learning requires a thoughtful approach that prioritizes efficiency, maintainability, and performance. Leveraging existing images, optimizing tags, choosing the right build approach, considering cross-platform compatibility, and optimizing layers are all essential best practices to ensure successful containerization of machine learning applications.

By following these best practices, you can create containers that facilitate seamless deployment, enhance reproducibility, and ultimately accelerate the development and deployment of your machine learning models. Containers provide a powerful foundation for machine learning workflows, and with the right approach, you can harness their potential to build robust and scalable machine learning applications.