Using Custom Code, Models and Containers in SageMaker

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Develop custom models using code in Apache MXNet and TensorFlow

Host pre-trained model artifacts on SageMaker containers

Train custom models on your own containers and host them

Use other Amazon services such as Redshift from within SageMaker

Custom Models in Apache MXNet

Apache MXNet is an effort undergoing incubation at The Apache Software Foundation (ASF), **sponsored by the** *Apache Incubator*. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.

TensorFlow

Open source deep learning framework by Google

Apache MXNet

Deep learning framework sponsored by the Apache Incubator

Apache Spark

Engine for big data processing with powerful ML libraries

Custom Code in SageMaker

TensorFlow

Open source deep learning framework by Google

Apache MXNet

Deep learning framework sponsored by the Apache Incubator

Apache Spark

Engine for big data processing with powerful ML libraries

To train and host custom code on SageMaker the code needs to follow a certain **training** and **inference interface** `train(…)`

Apache MXNet Training Code Interface

SageMaker calls this function with information on the training environment to run training

May return the model object which is passed to the save() function

`save(…)`

Apache MXNet Training Code Interface

Saves a model after training

This is an optional function, called only if you have a model to save

`model_fn(…)`

Apache MXNet Inference Code Interface

Loads a model from disk

```
transform_fn(…)
```

Apache MXNet Inference Code Interface

Transforms input data into a prediction result

The Apache MXNet inference interface for Gluon models (an imperative interface) are different

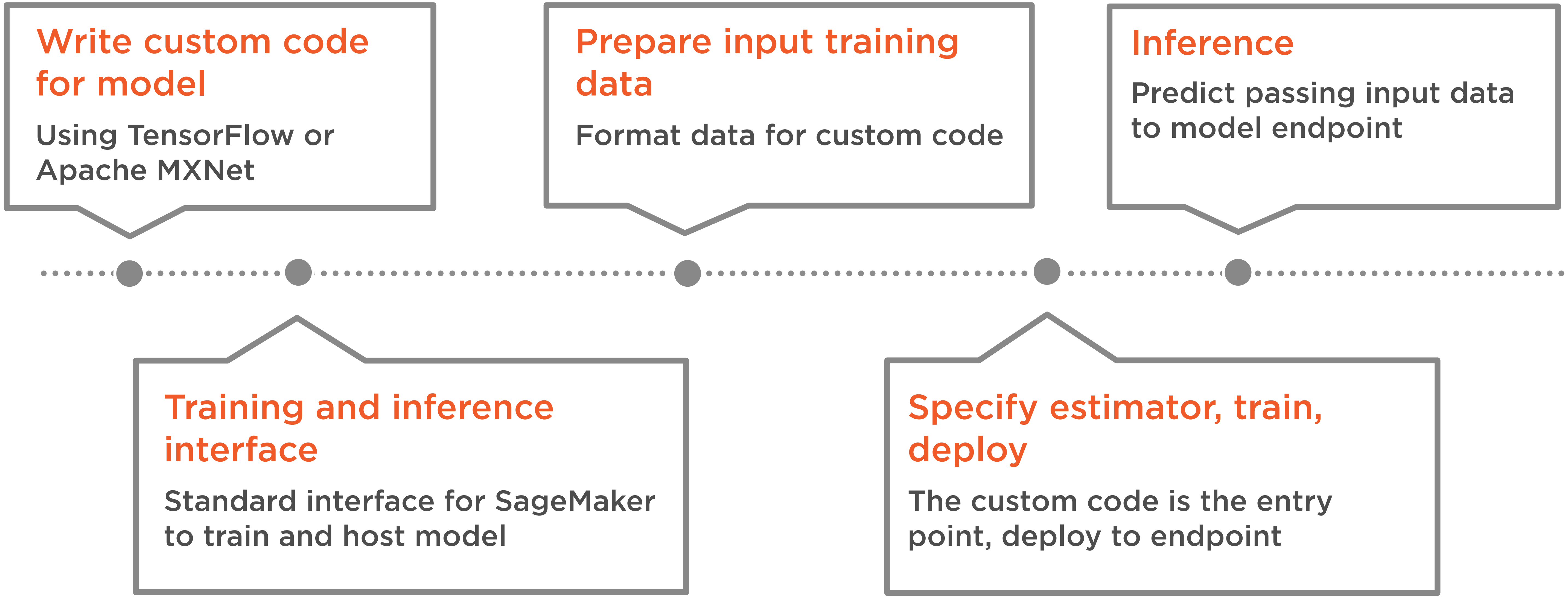
Demo

Use a neural network built in Apache MXNet for handwritten digit classification

Train on the MNIST dataset

Identify handwritten digits written on an HTML canvas

Custom Code on SageMaker



Write custom code for model

Using TensorFlow or Apache MXNet

Training and inference interface

Standard interface for SageMaker to train and host model

Prepare input training data

Format data for custom code

Specify estimator, train, deploy

The custom code is the entry point, deploy to endpoint

Inference

Predict passing input data to model endpoint

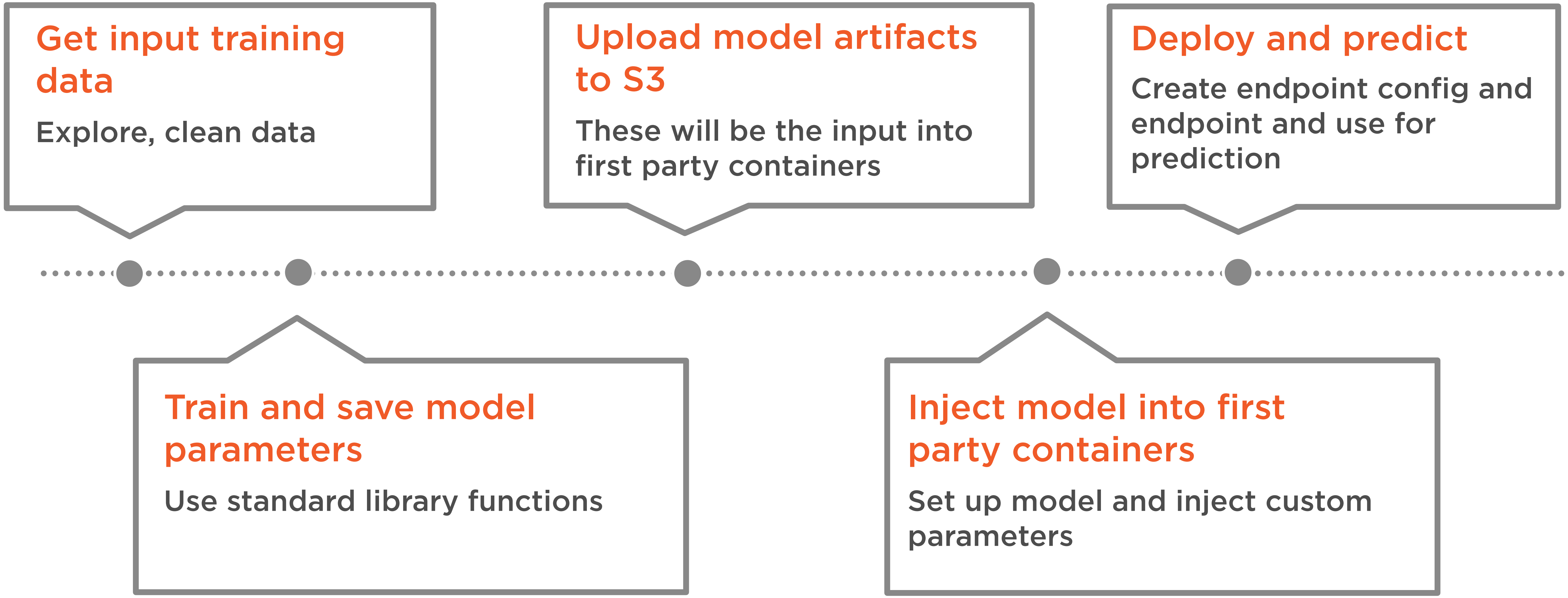
Demo

Set up the built-in algorithms to use your pre-trained model parameters

Cluster the MNIST handwritten digits using the k-means clustering algorithms in scikit-learn

Inject the cluster centers into SageMaker’s first party containers

Custom Model on SageMaker



Get input training data

Explore, clean data

Train and save model parameters

Use standard library functions

Upload model artifacts to S3

These will be the input into first party containers

Inject model into first party containers

Set up model and inject custom parameters

Deploy and predict

Create endpoint config and endpoint and use for prediction

Demo

Setting up a Redshift cluster on AWS

Demo

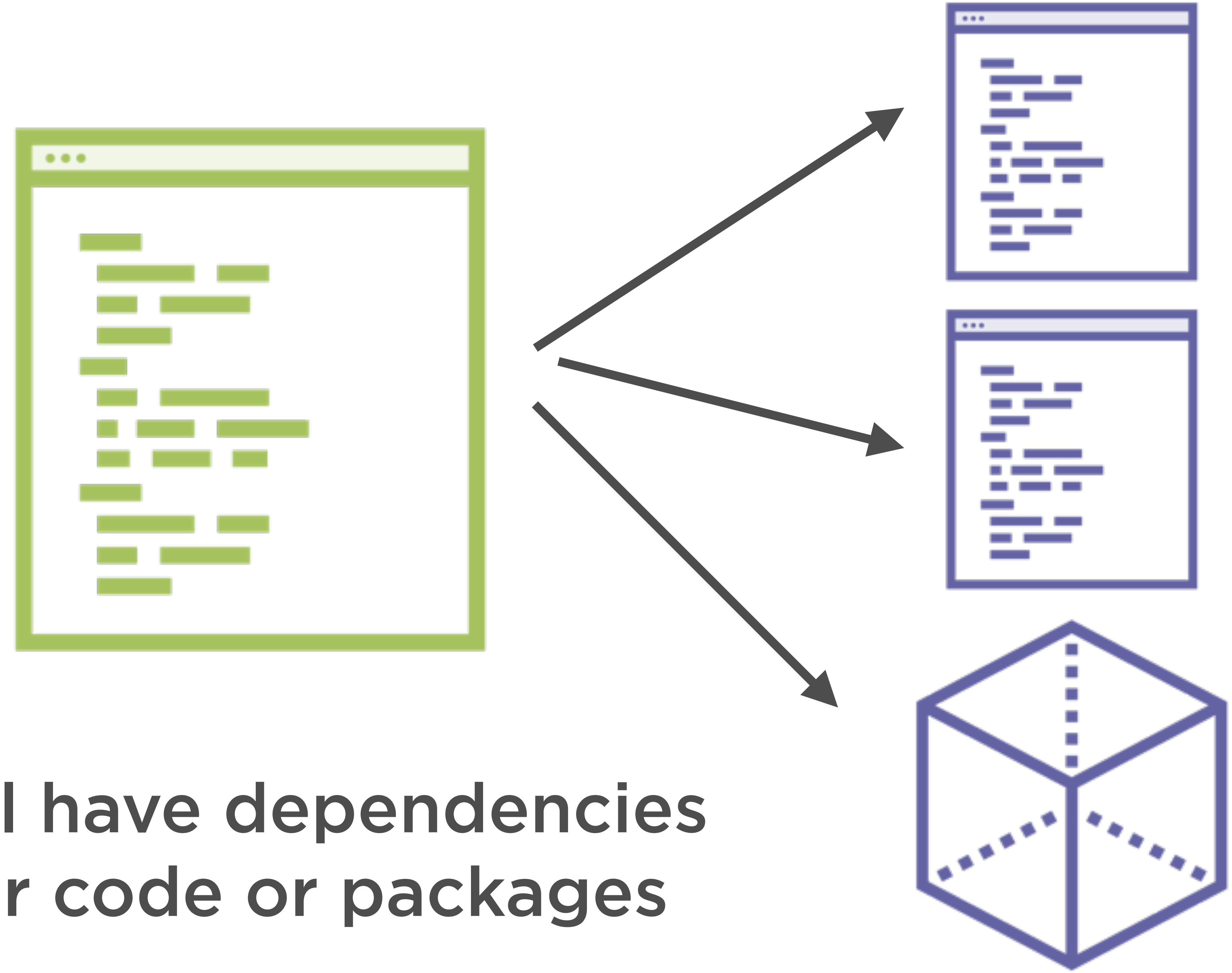
Connecting to and querying Redshift from within SageMaker’s Jupyter notebook instance

Bring Your Own Container

Custom Algorithms on SageMaker

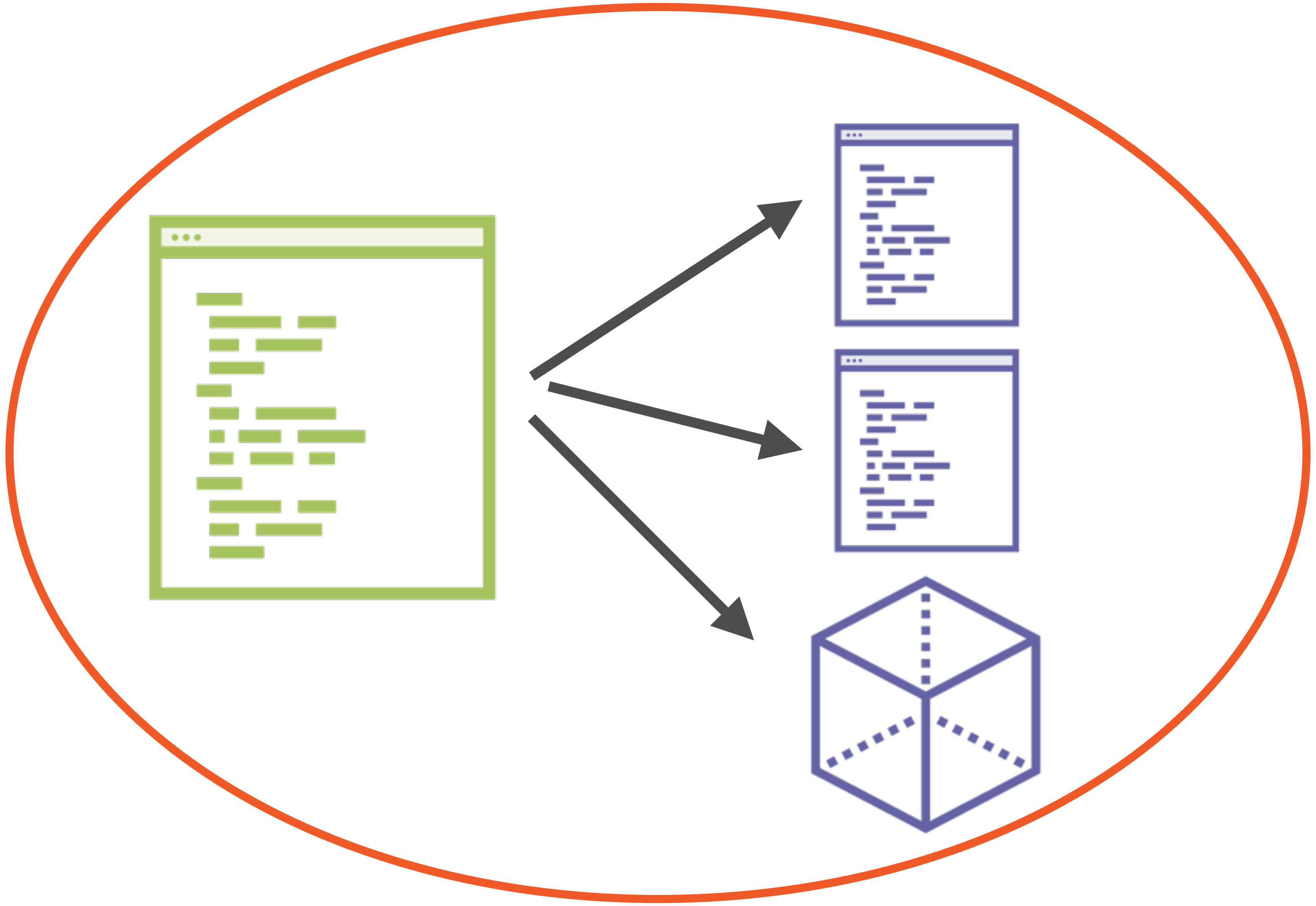
Custom code to train and host our model

Custom Algorithms on SageMaker



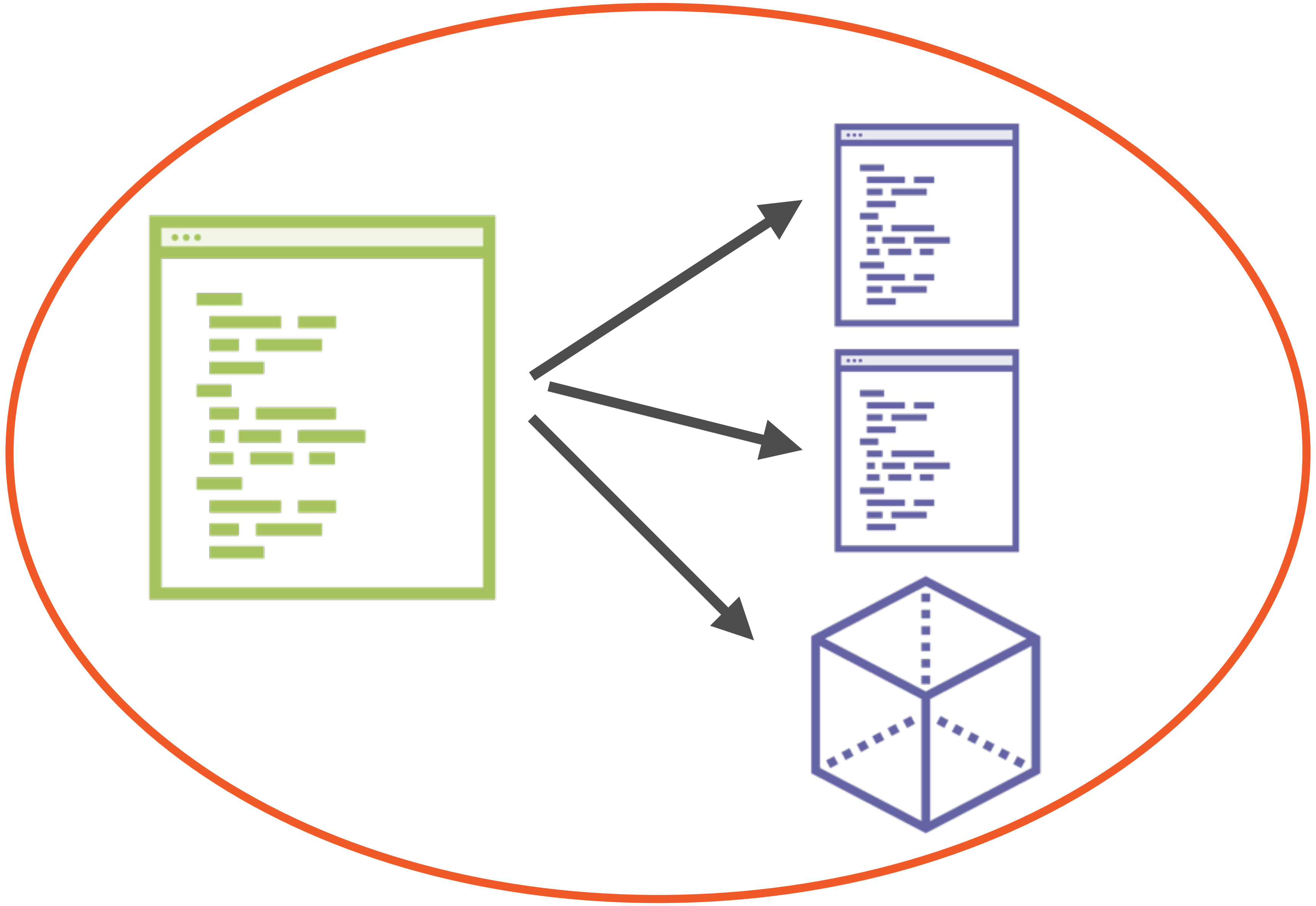
Code will have dependencies on other code or packages

Docker Containers



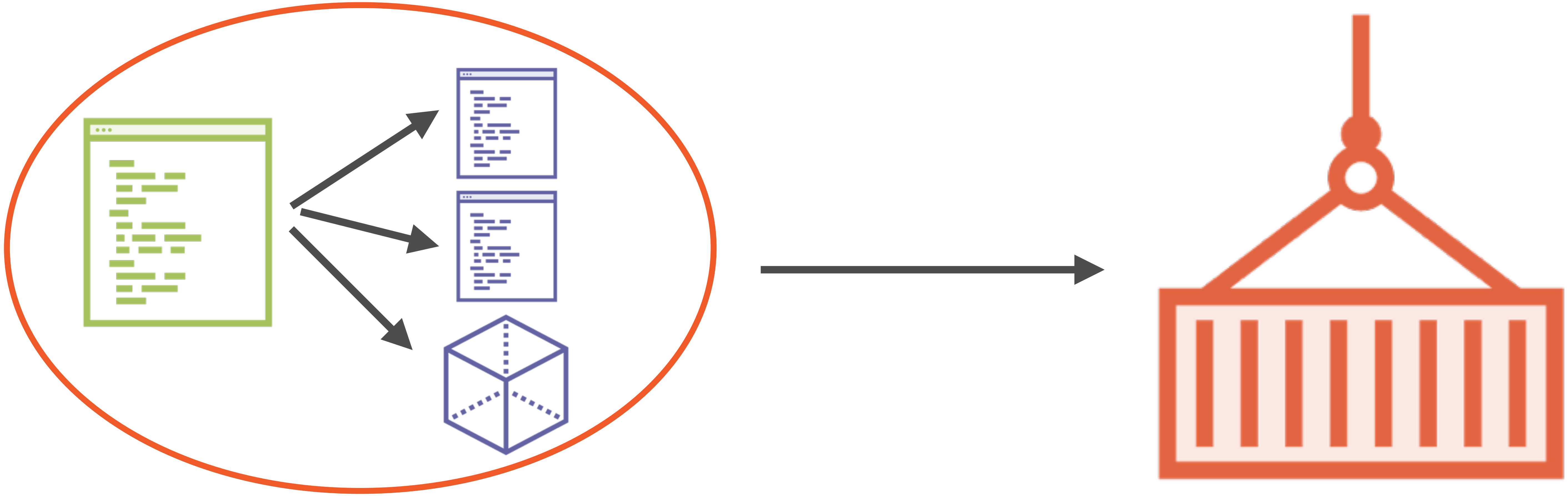
Docker packages all of this arbitrary code into an **image**

Docker Containers



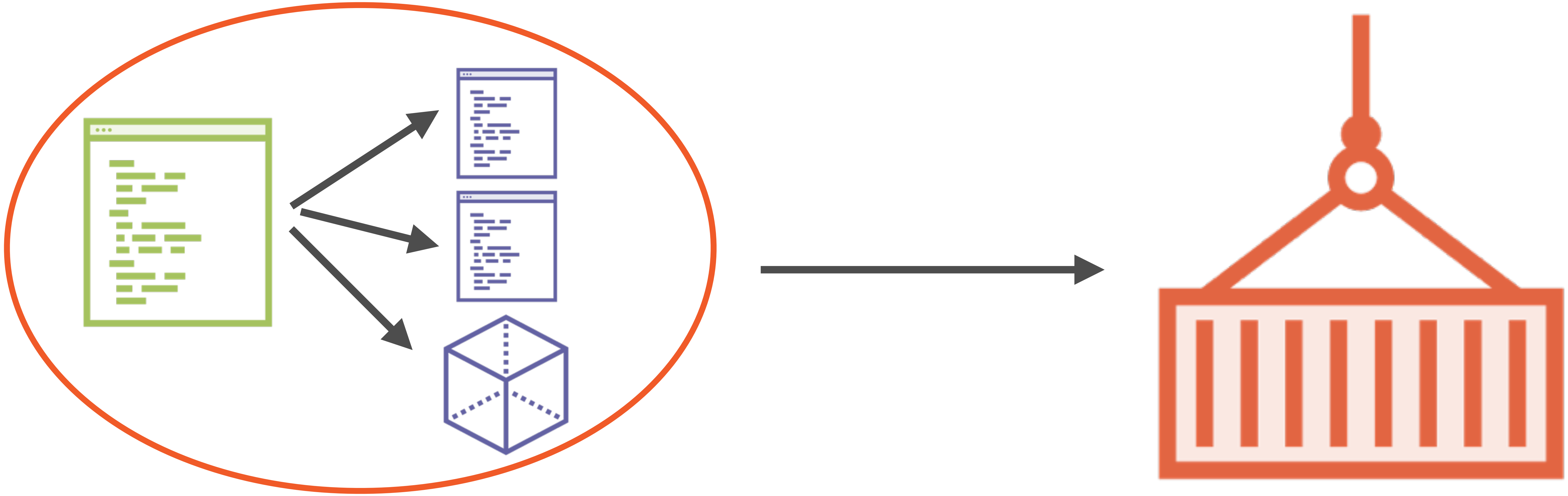
An image is completely self-sufficient

Docker Containers



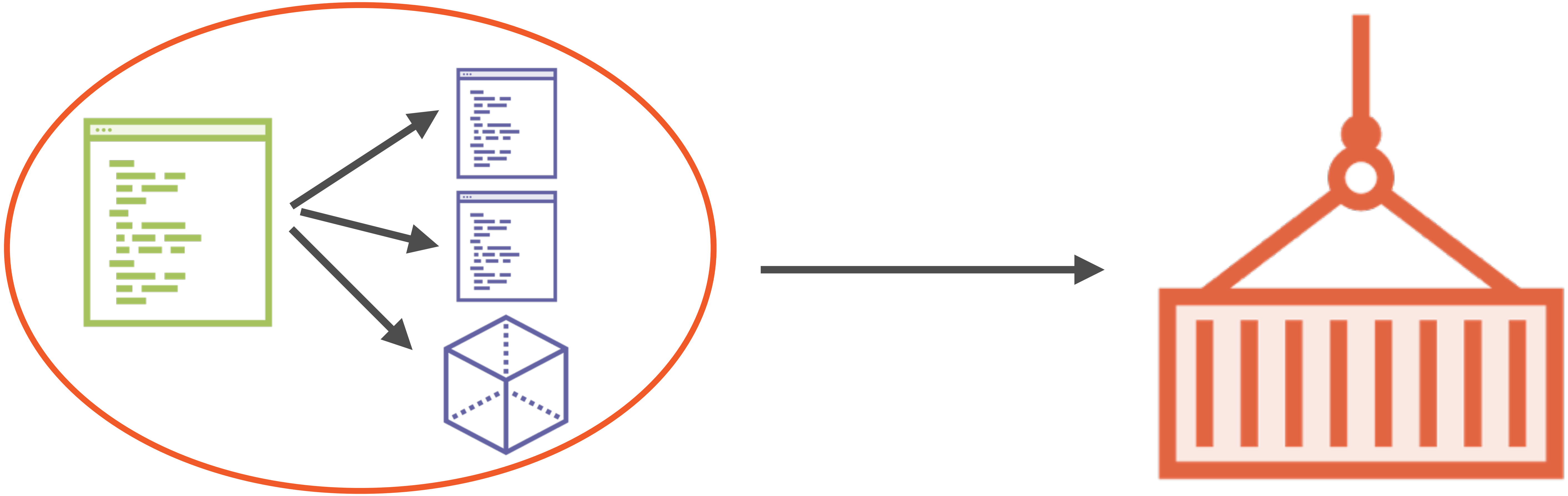
An image can be used to run a Docker container

Docker Containers



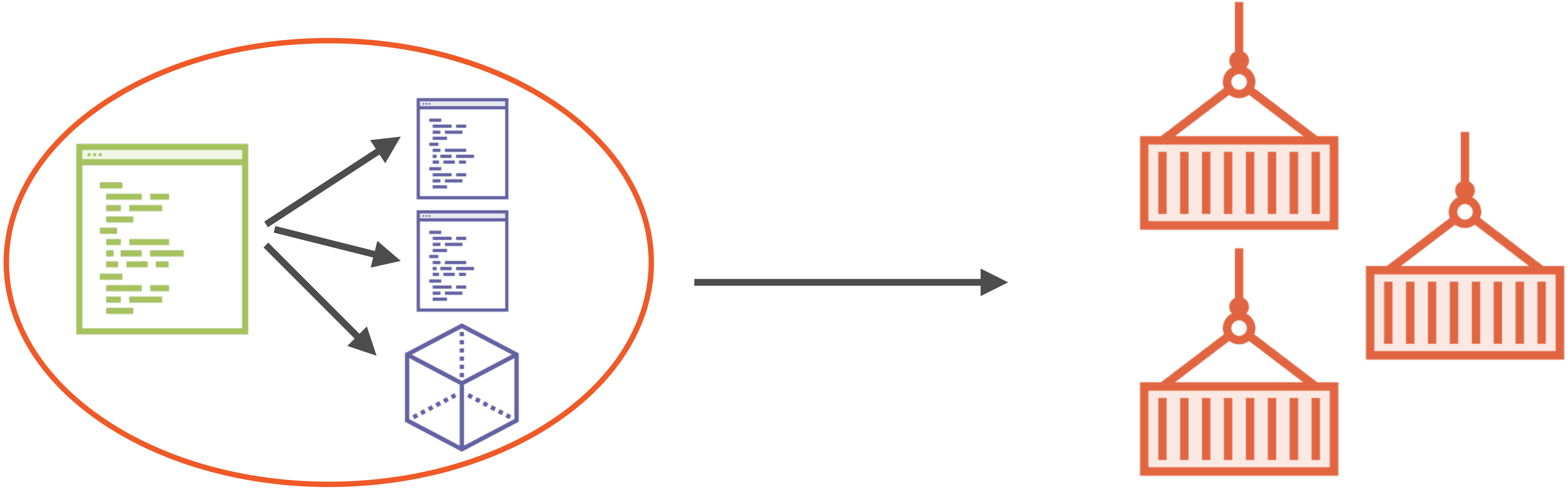
Containers are a fully self contained environment which executes code

Docker Containers

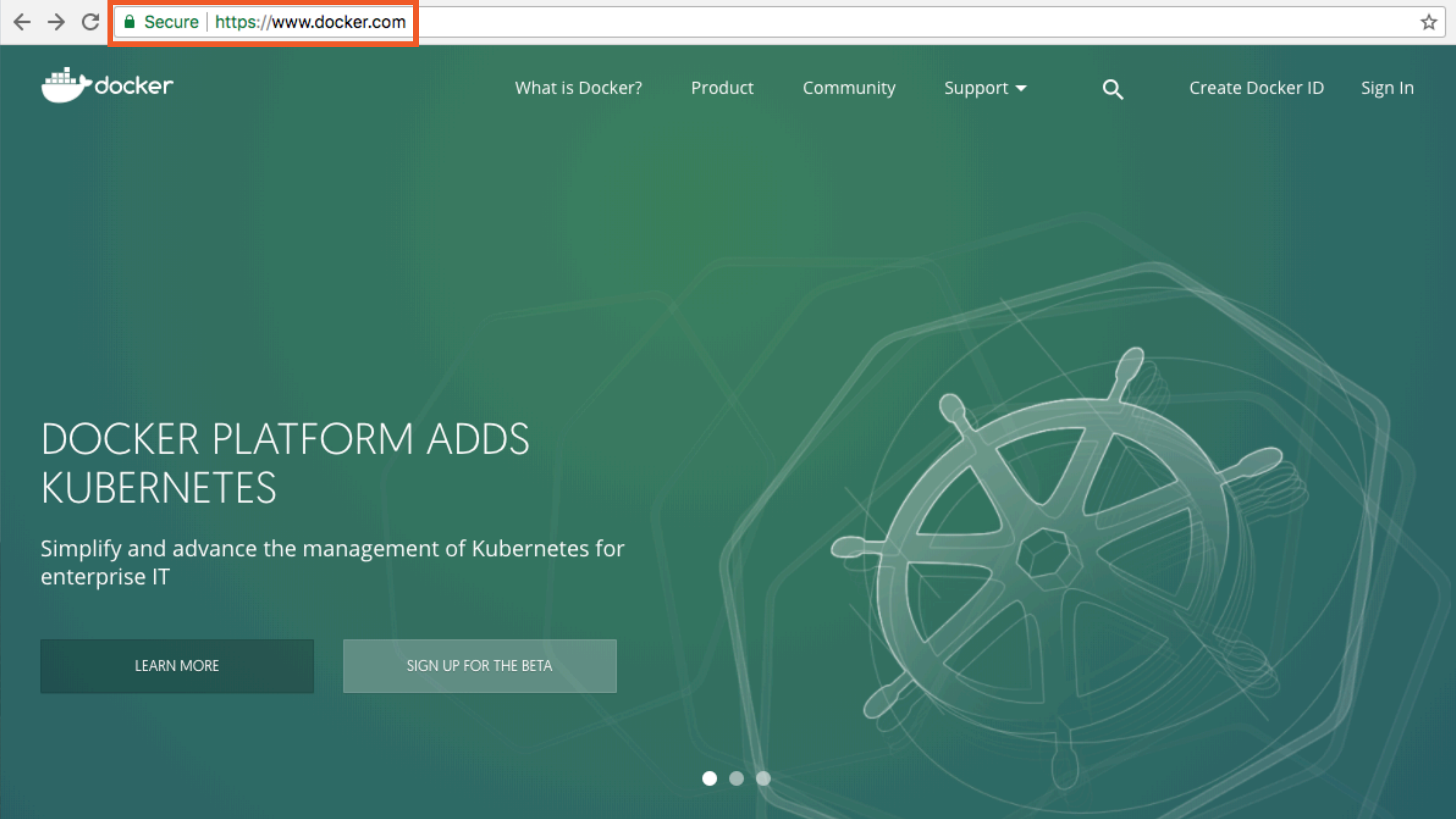


Containers do not contain the OS and your code is abstracted away from the machine

Docker Containers



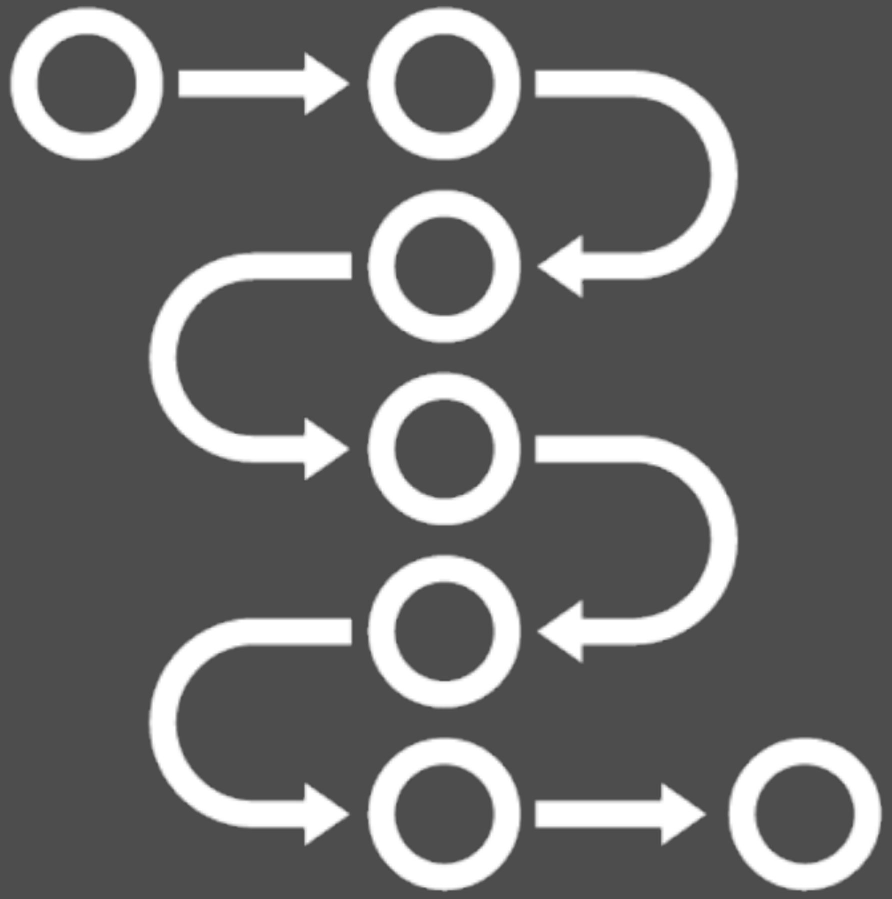
You can create as many containers as you want from the same image



DOCKER PLATFORM ADDS KUBERNETES

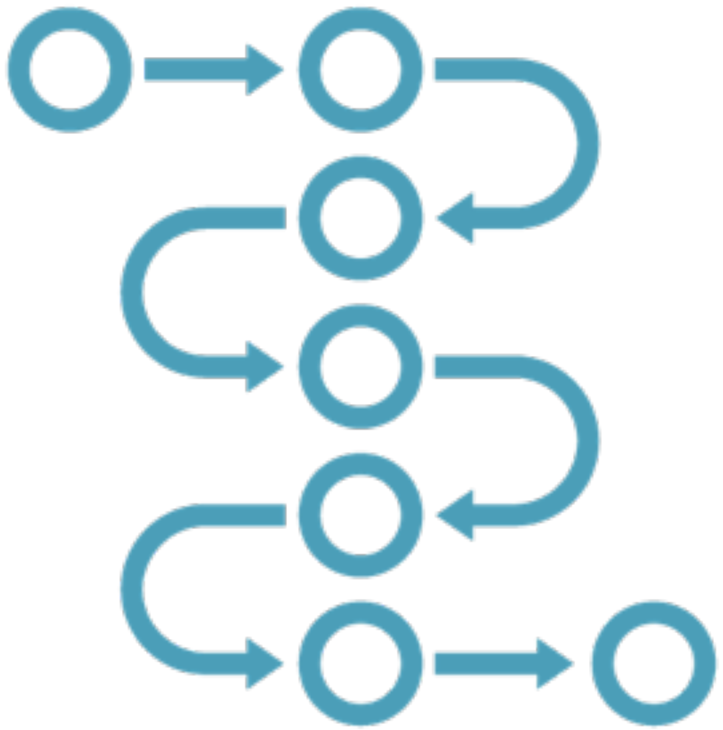
Simplify and advance the management of Kubernetes for enterprise IT

SageMaker can train and host custom ML code in Docker containers



Getting your own container set up on SageMaker is fairly complex

SageMaker has an end-to-end example which you can tweak to add in your own custom model



Custom Algorithm

Decision tree classifier from scikit-learn to classify Iris flowers

Input features: Petal and sepal length and width

Output labels: Iris Versicolor, Iris Setosa, Iris Virginica

Running the Docker Container

Training

Run the `train` script and use `/opt/ml` directory to store data

Hosting

Use `nginx, gunicorn, flask` to respond to HTTP requests

Running the Docker Container

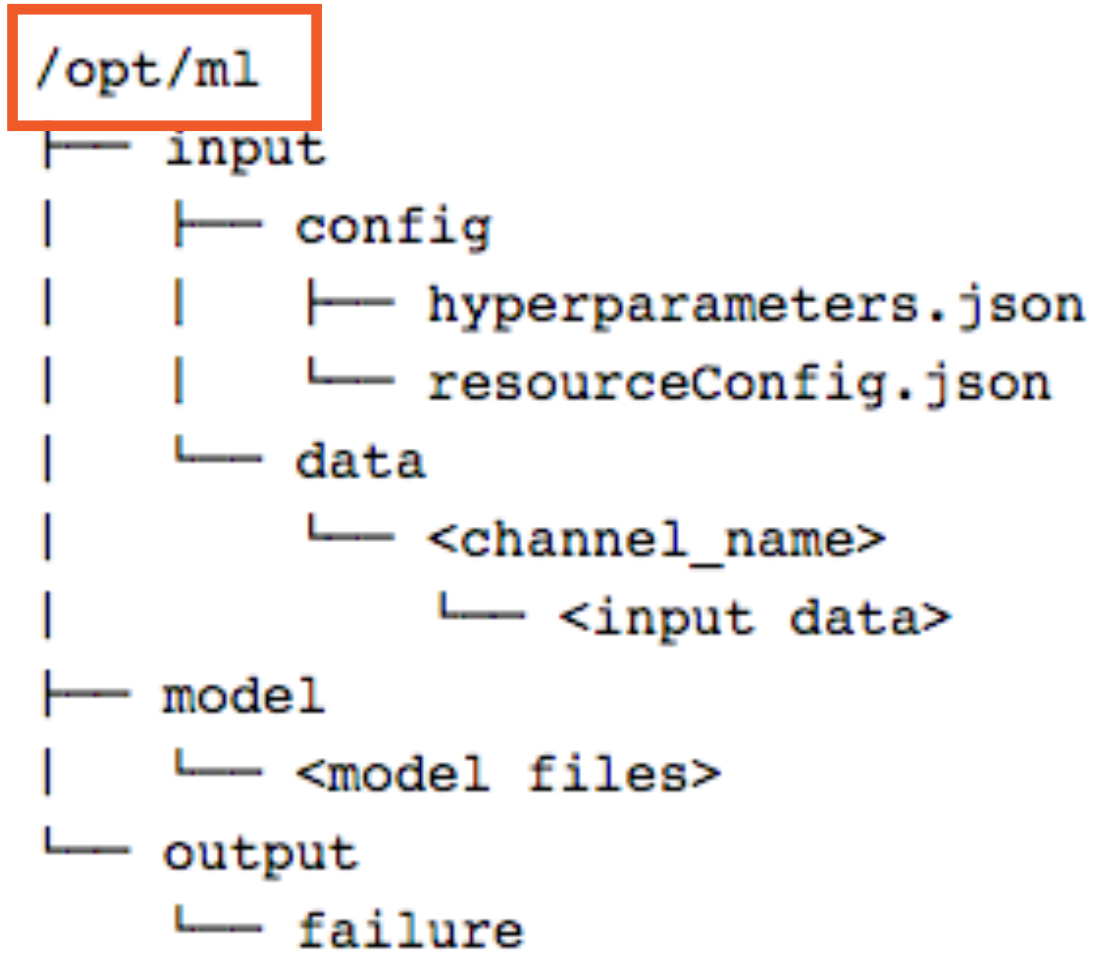
Training

Run the `train` script and use `/opt/ml` directory to store data

Hosting

Use `nginx`, `gunicorn`, `flask` to respond to HTTP requests

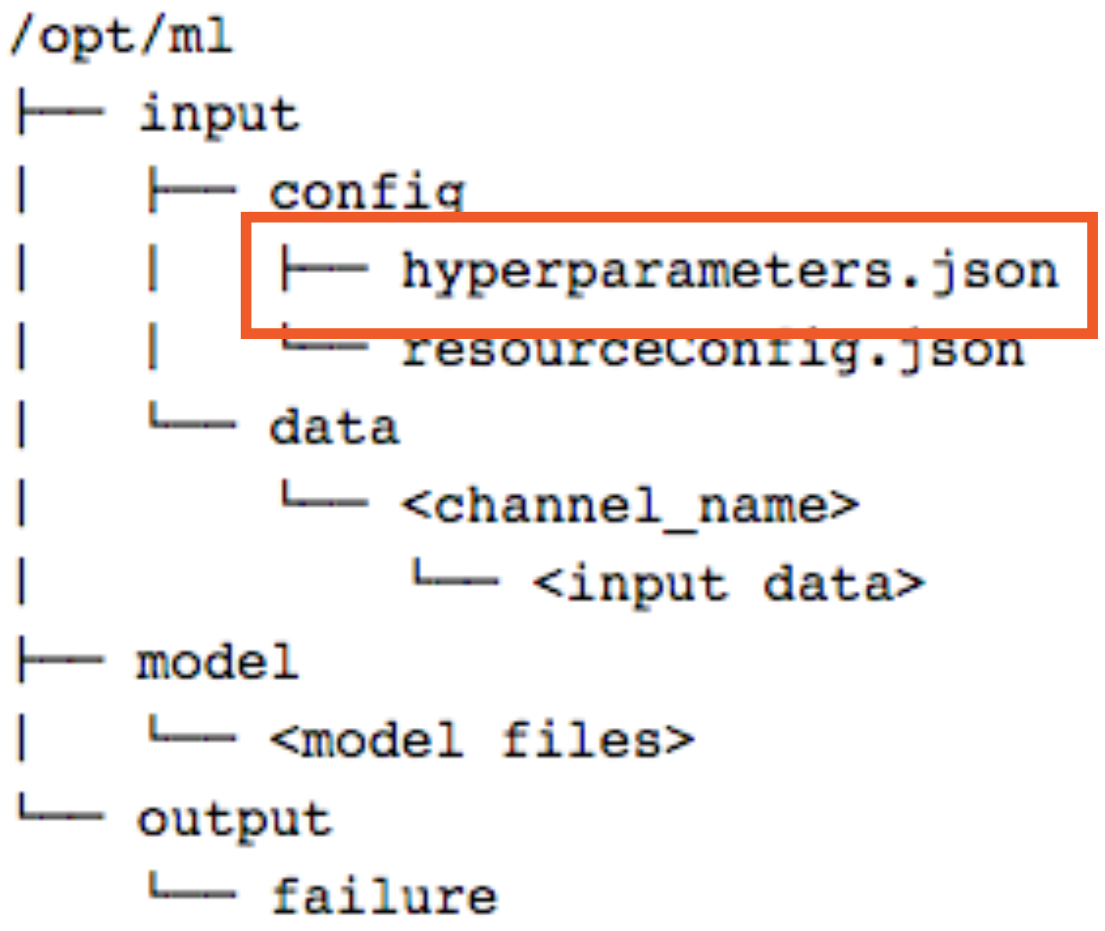
Training



```
/opt/ml
├── input
│   ├── config
│   │   ├── hyperparameters.json
│   │   └── resourceConfig.json
│   └── data
│       └── <channel_name>
│           └── <input data>
├── model
│   └── <model files>
└── output
    └── failure
```

High level directory where SageMaker stores information used during the training run

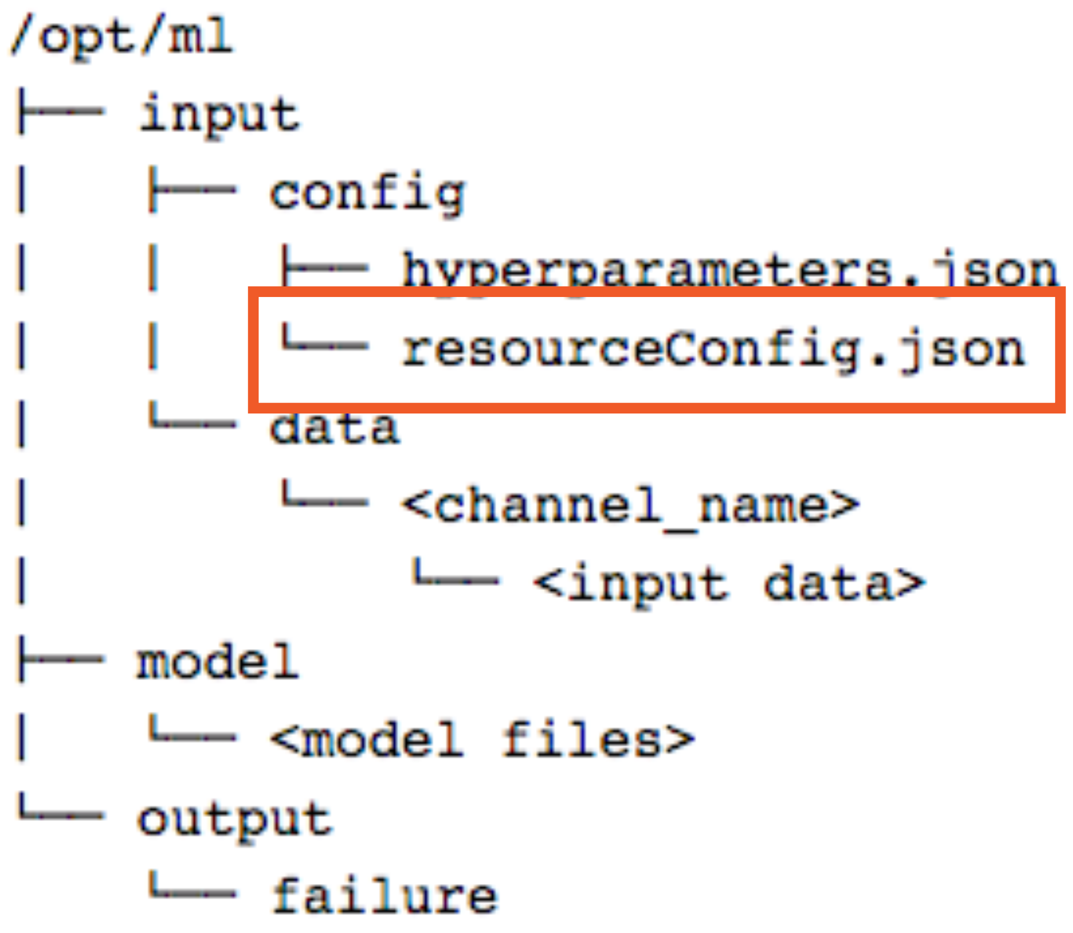
Training



```
/opt/ml
├── input
|   ├── config
|   |   ├── hyperparameters.json
|   |   └── resourceConfig.json
|   └── data
|       └── <channel_name>
|           └── <input data>
├── model
|   └── <model files>
└── output
    └── failure
```

Hyperparameters control how your program runs

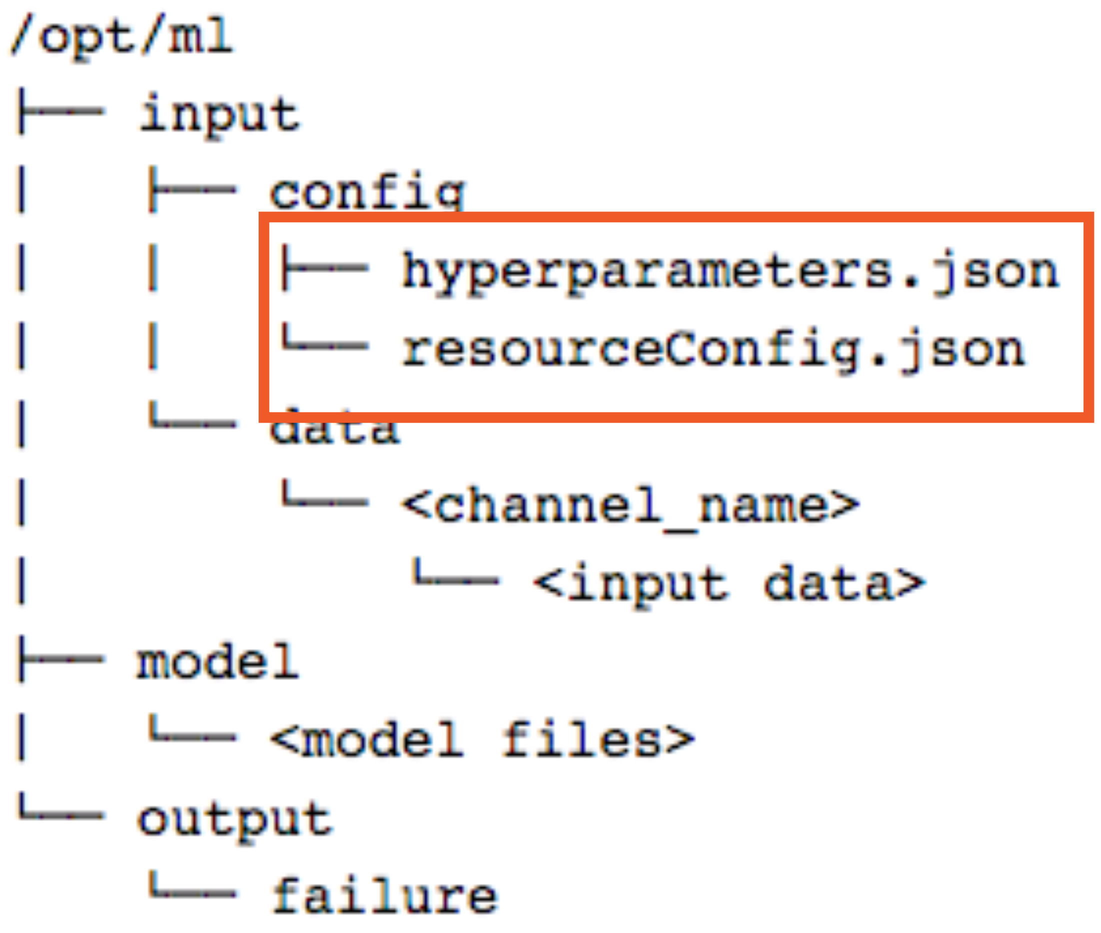
Training



```
/opt/ml
├── input
|   ├── config
|   |   ├── hyperparameters.json
|   |   └── resourceConfig.json
|   └── data
|       └── <channel_name>
|           └── <input data>
├── model
|   └── <model files>
└── output
    └── failure
```

Specify the network layout for distributed training

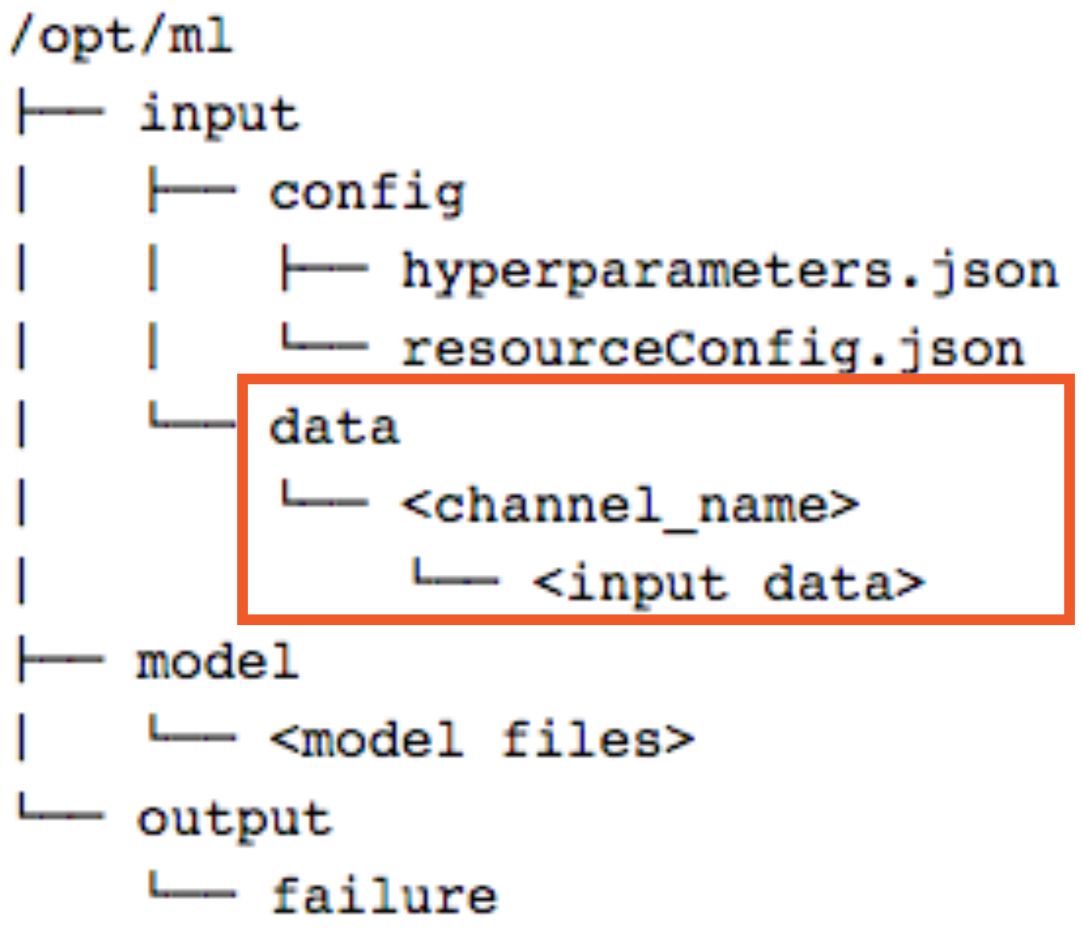
Training



```
/opt/ml
├── input
|   ├── config
|   |   ├── hyperparameters.json
|   |   └── resourceConfig.json
|   └── data
|       └── <channel_name>
|           └── <input data>
├── model
|   └── <model files>
└── output
    └── failure
```

scikit-learn libraries cannot be run in a distributed manner - we won't be using both of these

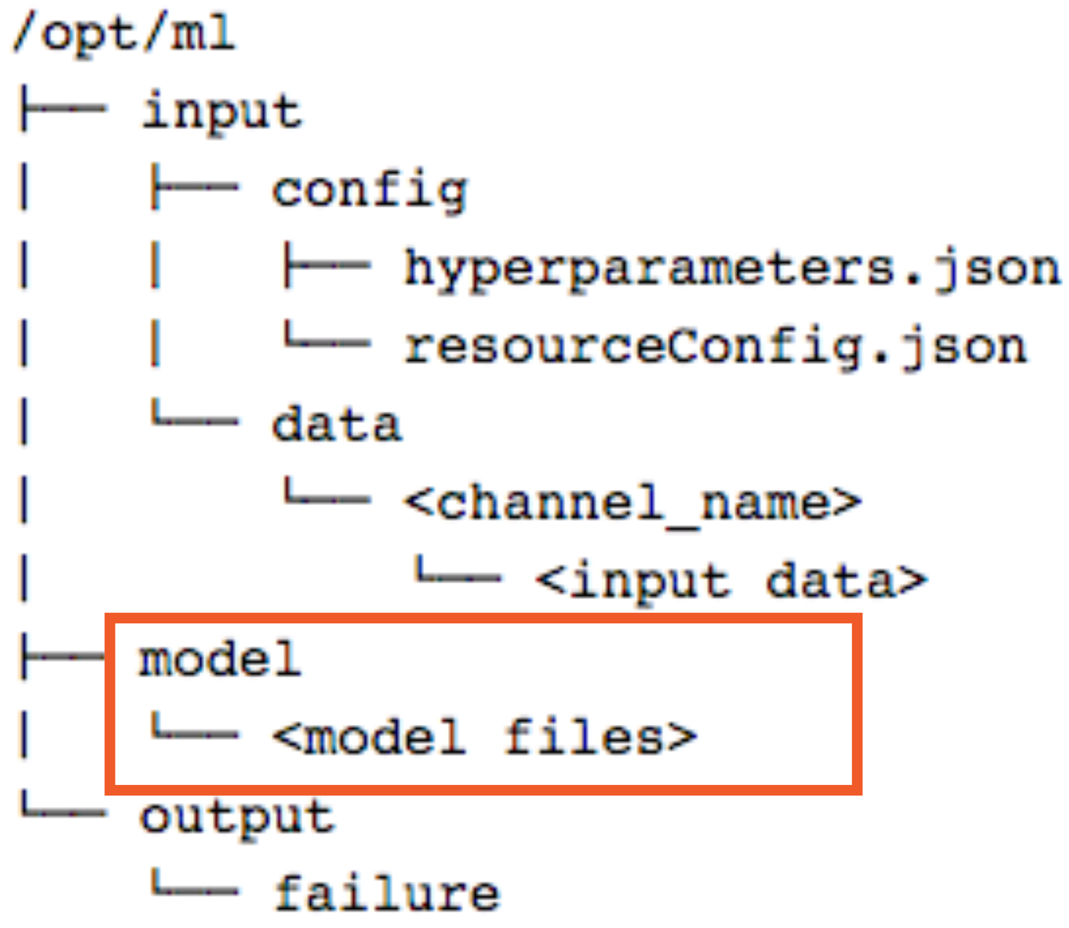
Training



```
/opt/ml
├── input
|   ├── config
|   |   ├── hyperparameters.json
|   |   └── resourceConfig.json
|   └── data
|       └── <channel_name>
|           └── <input data>
├── model
|   └── <model files>
└── output
    └── failure
```

The input data is copied from the S3 bucket to this location

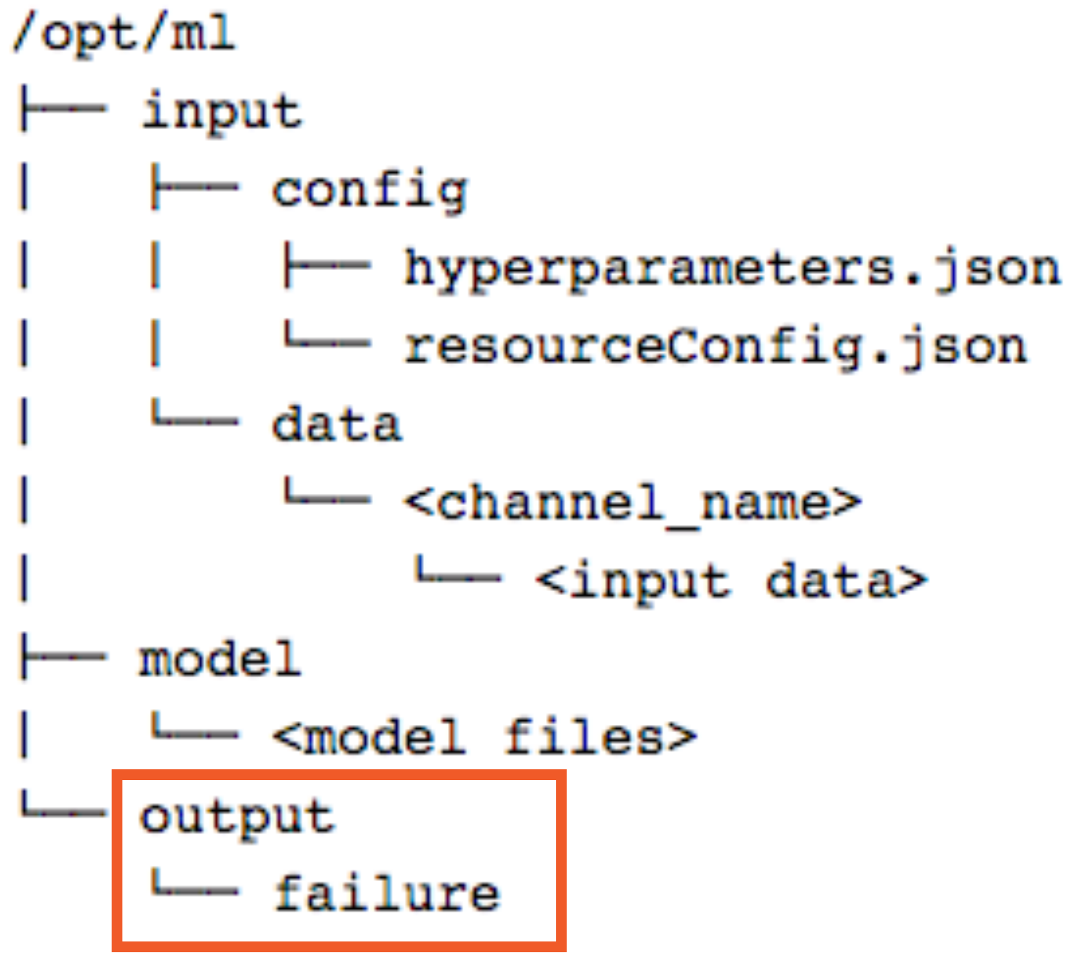
Training



```
/opt/ml
├── input
|   ├── config
|   |   ├── hyperparameters.json
|   |   └── resourceConfig.json
|   └── data
|       └── <channel_name>
|           └── <input data>
├── model
|   └── <model files>
└── output
    └── failure
```

Saved model parameters which are then uploaded to S3

Training



```
/opt/ml
├── input
|   ├── config
|   |   ├── hyperparameters.json
|   |   └── resourceConfig.json
|   └── data
|       └── <channel_name>
|           └── <input data>
├── model
|   └── <model files>
└── output
    └── failure
```

Written out only in the case of failed jobs

Running the Docker Container

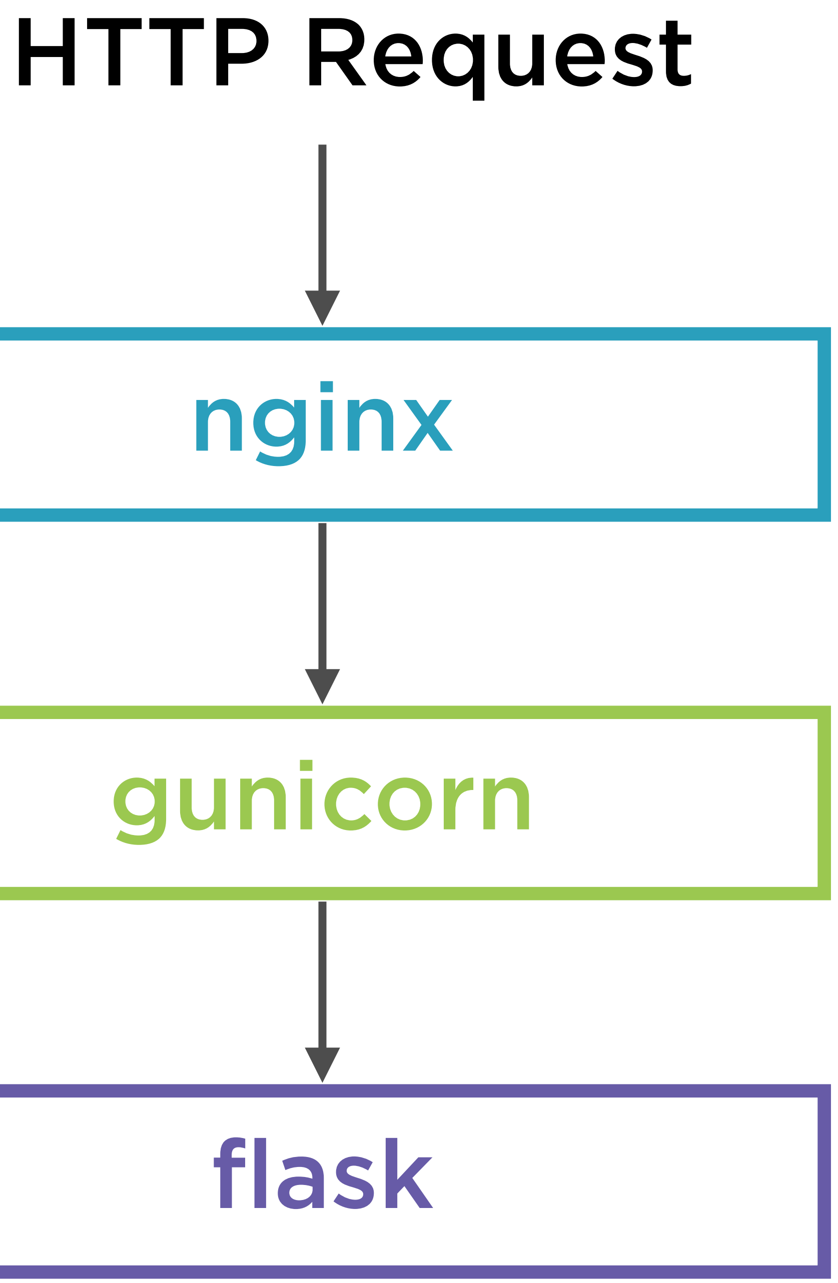
Training

Run the `train` script and use `/opt/ml` directory to store data

Hosting

Use `nginx, gunicorn, flask` to respond to HTTP requests

Hosting


HTTP Request

nginx

gunicorn

flask

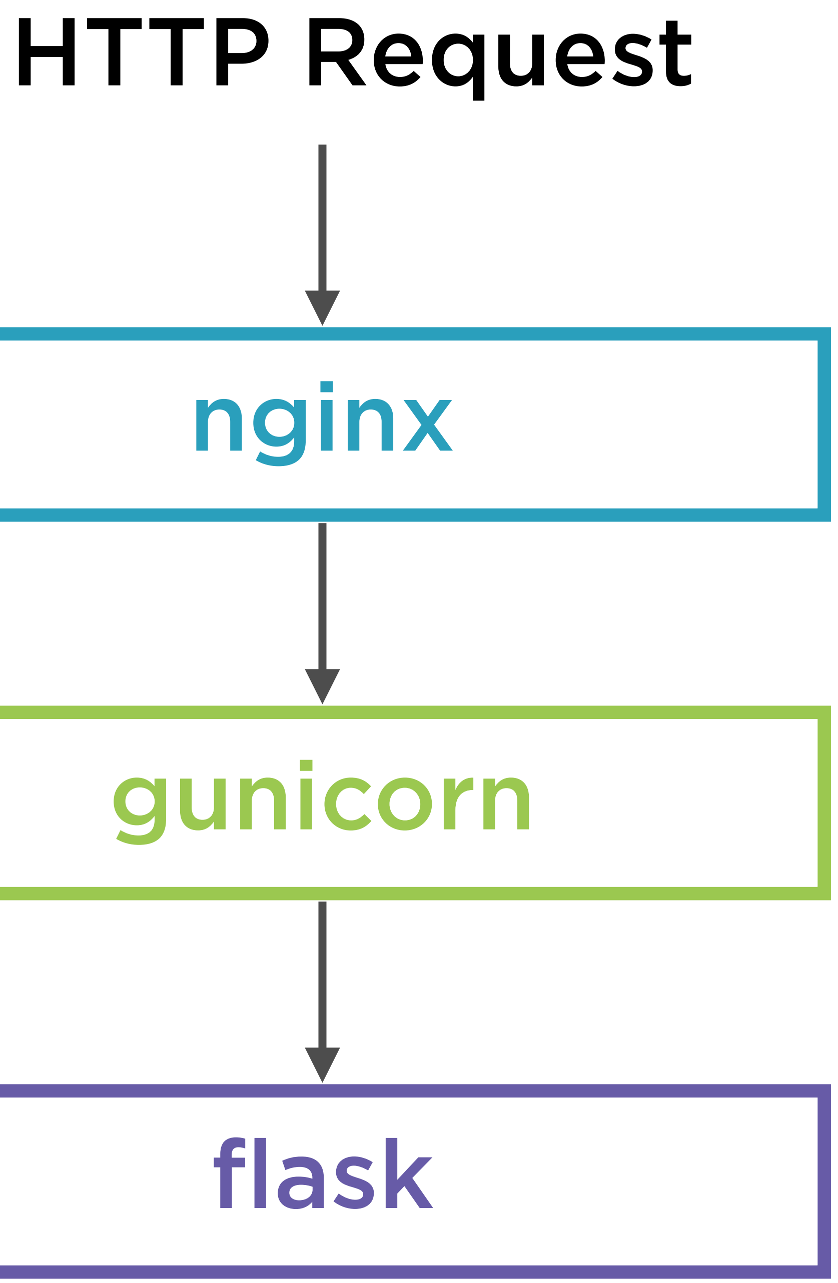
nginx

Open source software for web serving, reverse proxying, caching, load balancing

Reverse proxy:

- Sits behind a firewall and directs requests to the appropriate backend
- Additional level of abstraction between client and server

Hosting



HTTP Request

nginx

gunicorn

flask

gunicorn

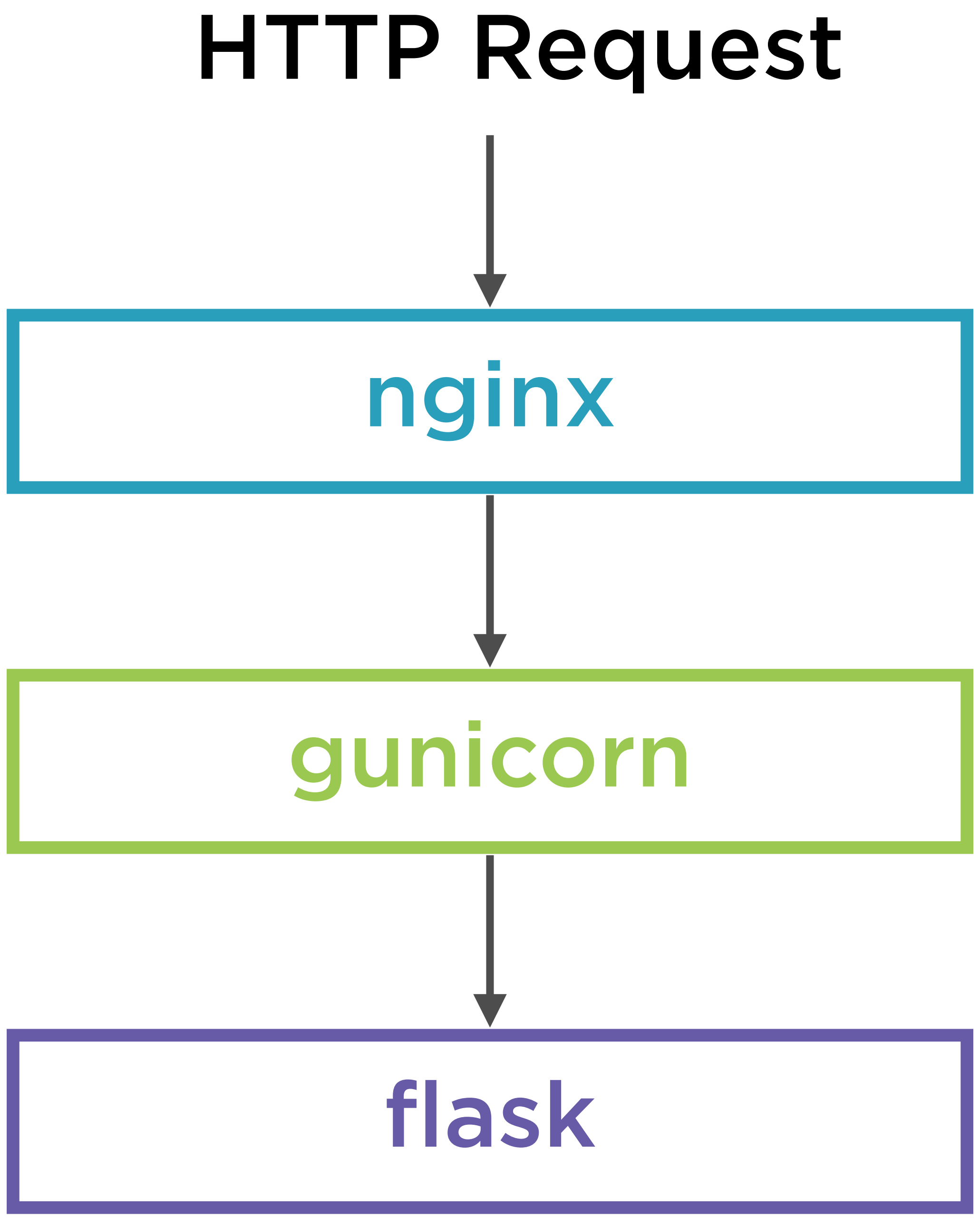
gunicorn

Web server for Unix

WSGI HTTP Server:

- WSGI (Web Server Gateway Interface) is a Python standard which determines how a web server communicates with applications
- Simple, lightweight, fast and works with many web frameworks

Hosting



HTTP Request

nginx

gunicorn

flask

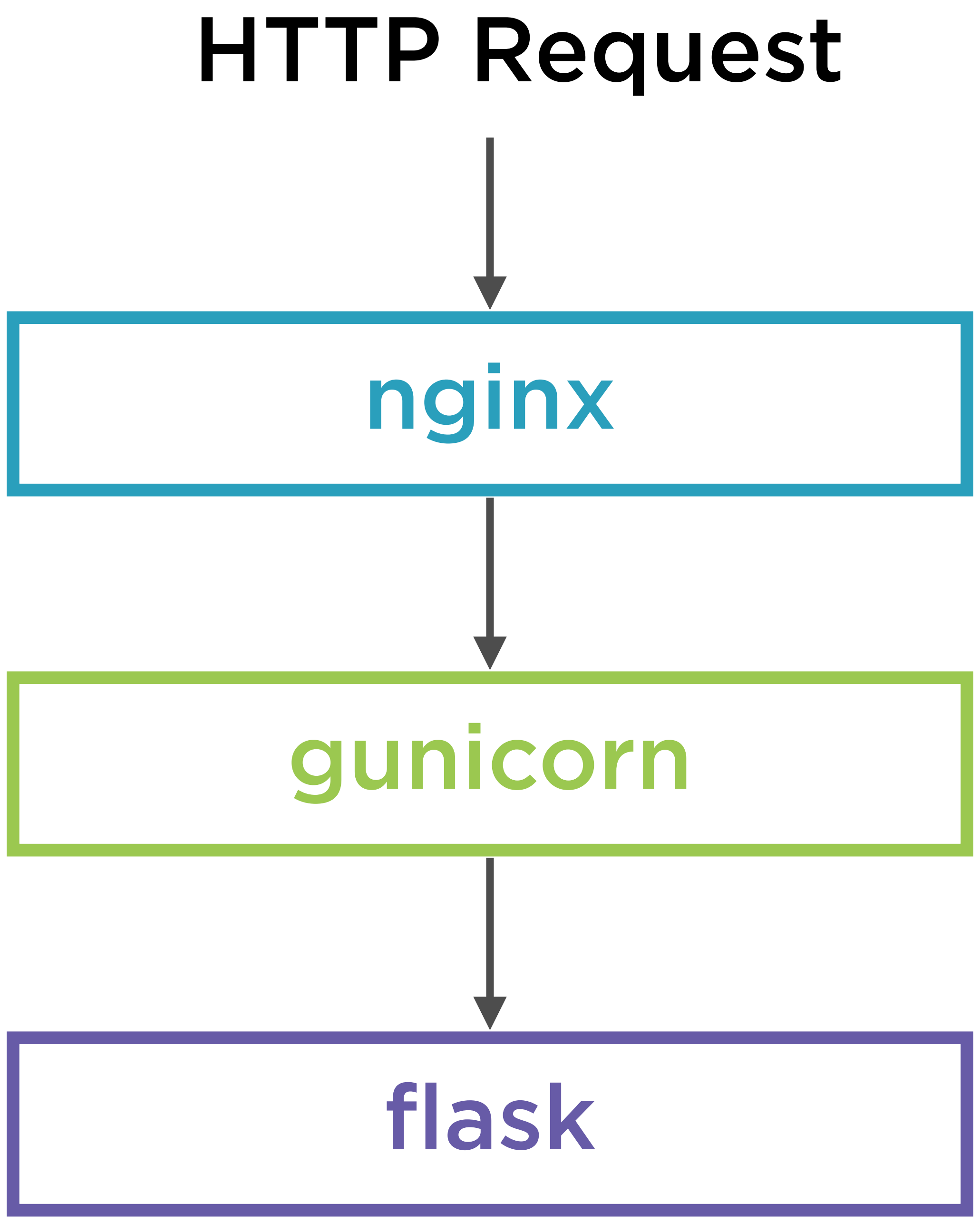
flask

Microframework for Python web app development

Worker:

- The actual instance of the application which hosts the inference code
- Loads the trained model and returns prediction results

Hosting



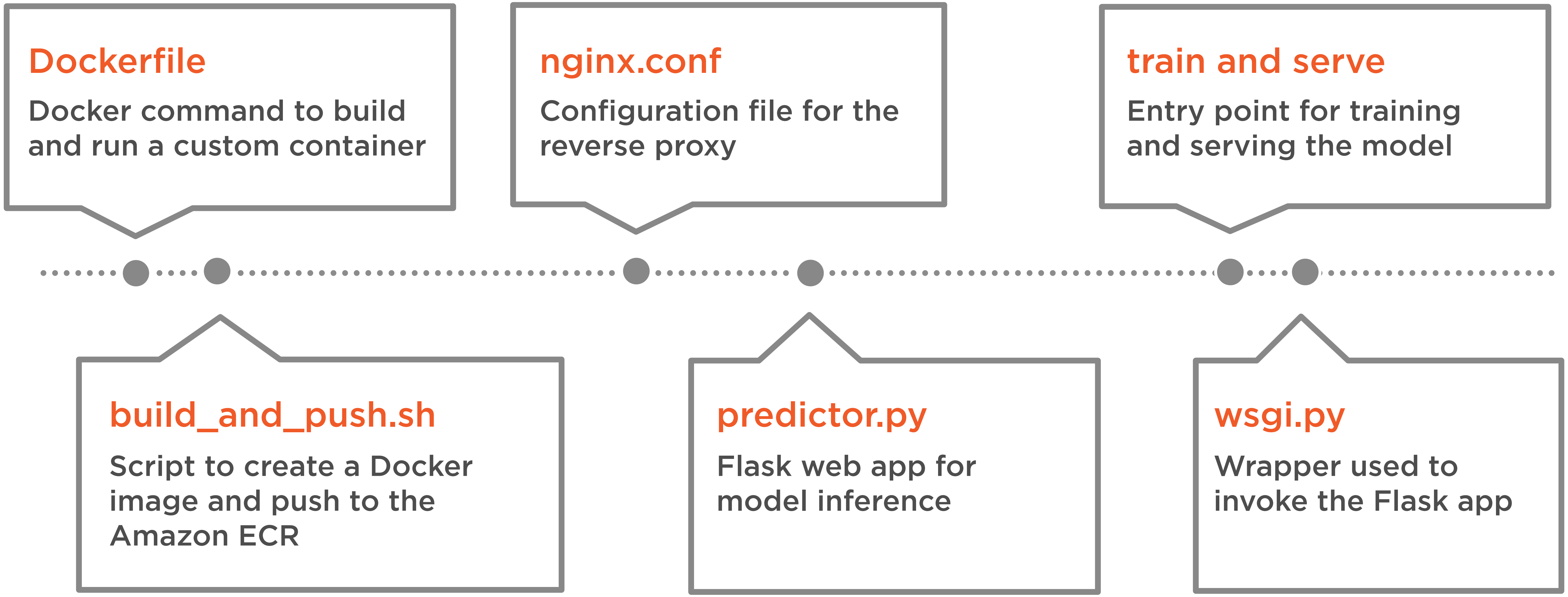
HTTP Request

nginx

gunicorn

flask

Custom Container Components



Dockerfile

Docker command to build and run a custom container

build_and_push.sh

Script to create a Docker image and push to the Amazon ECR

nginx.conf

Configuration file for the reverse proxy

predictor.py

Flask web app for model inference

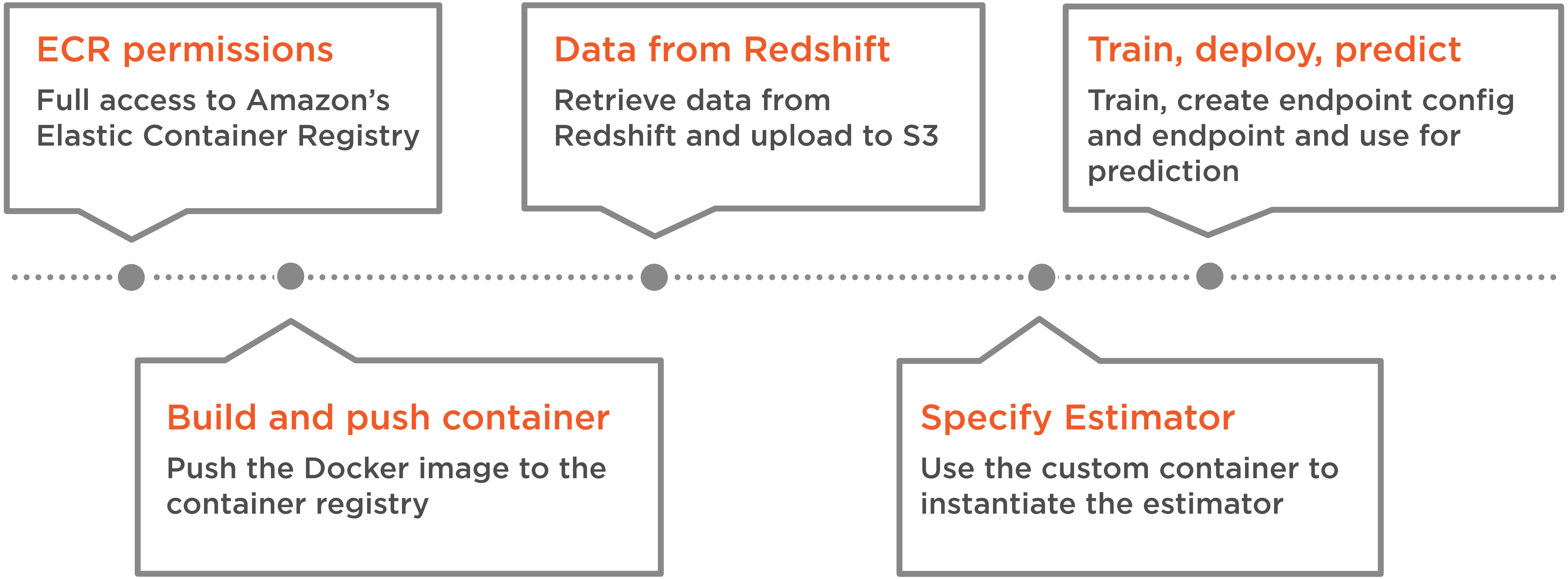
train and serve

Entry point for training and serving the model

wsgi.py

Wrapper used to invoke the Flask app

Custom Container on SageMaker



ECR permissions

Full access to Amazon's Elastic Container Registry

Build and push container

Push the Docker image to the container registry

Data from Redshift

Retrieve data from Redshift and upload to S3

Specify Estimator

Use the custom container to instantiate the estimator

Train, deploy, predict

Train, create endpoint config and endpoint and use for prediction

Demo

Use a Docker container with custom code for training and hosting

Access data from Redshift and store to S3 buckets

Use decision trees in scikit-learn for Iris dataset classification

Summary

Develop custom models using code in Apache MXNet and TensorFlow

Host pre-trained model artifacts on SageMaker containers

Train custom models on your own containers and host them

Use other Amazon services such as Redshift from within SageMaker