MLOps: ML Engineering Best Practices from the Trenches

DR. SOURAV DEY & ALEX NG

ODSC WEST

OCTOBER 31, 2019

MANIFOLD

www.manifold.ai

About Us

Manifold is a full-service AI development services firm that **accelerates AI development** for leading companies.

Our exceptional team has a proven ability to design, build, deploy, and manage complex data applications at scale.

What to Expect

- Follow-along type of workshop
- Prerequisites
  - Download Docker for your OS (Mac, Windows, Linux)
  - Download `orbyter-ml-dev` Docker image from DockerHub

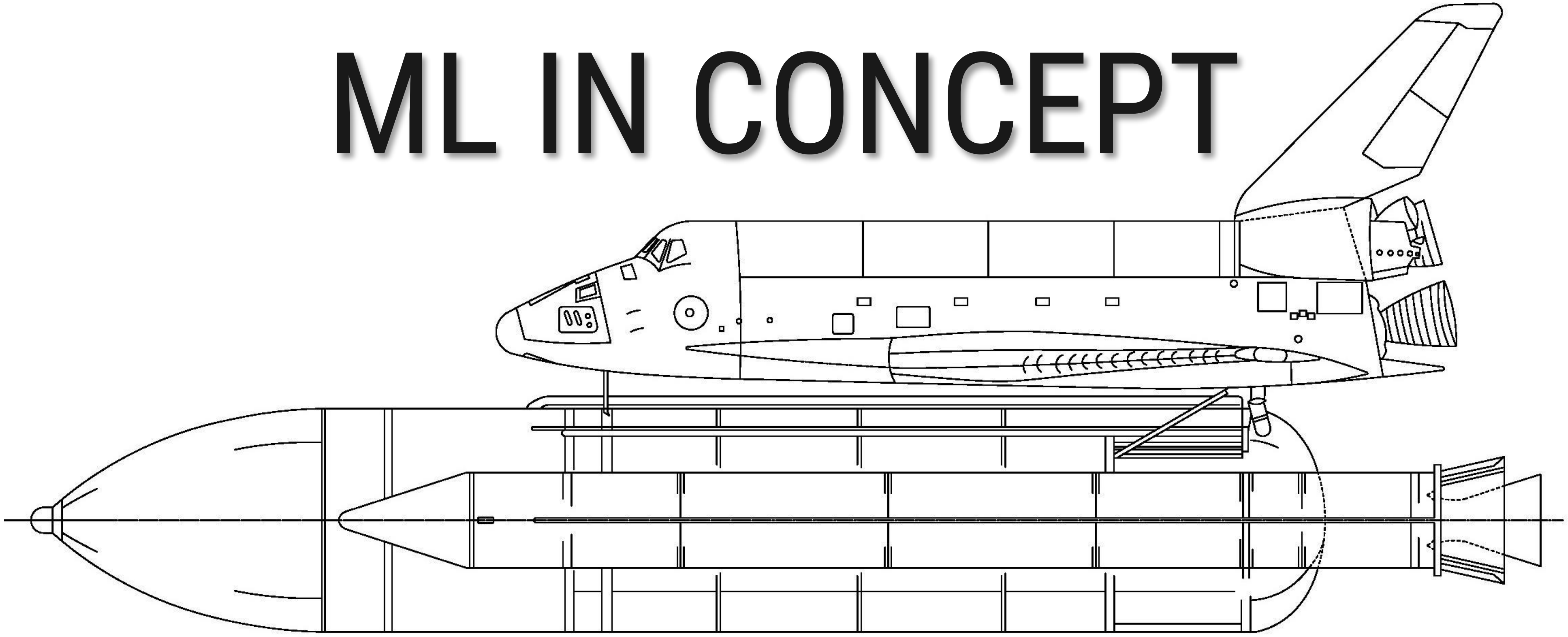
Agenda

- Background (10 mins)
- Key Lessons
  - Use Docker on Day One (25 min)
  - Use a Structured ML Software Workflow (25 min)
  - Downstream Containerization Benefits (15 min)
- Conclusion / Q&A (15 min)

The Problem

The Problem

ML IN CONCEPT



ML IN PRODUCTION

Why is This Hard?

Inherent vs. Incidental Complexity

Inherent Complexity

The Ways the Problem You're Solving is Hard

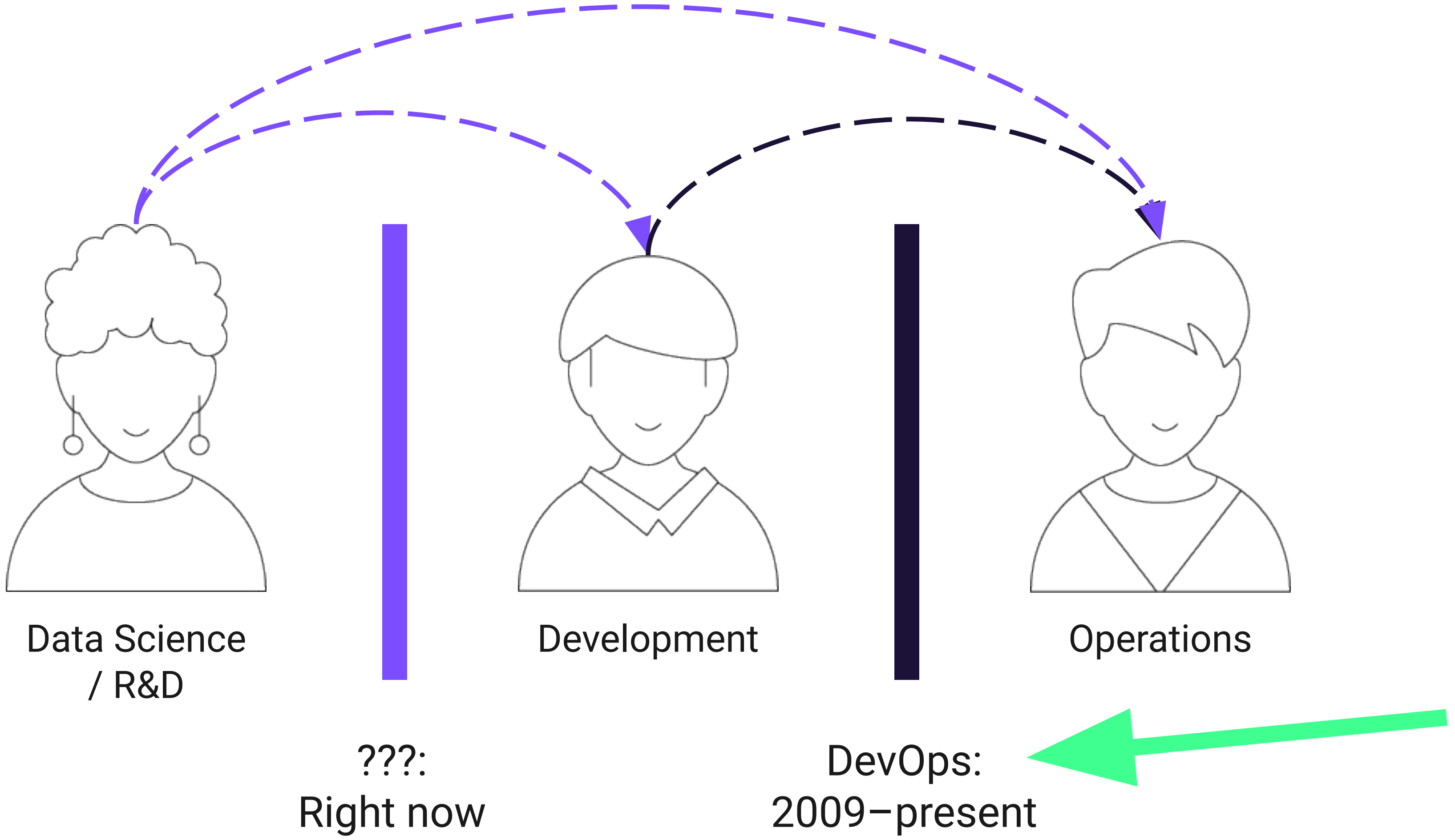
~~Incidental Complexity~~

~~When Programmers Make Their Lives Worse~~

We All Have Problems

- Wait… what version of Python should I use?
- Can I use the same version on both of my projects? How can I switch?
- Will installing new version of PyTorch mess up my other projects?
- I need a bigger machine. How do I install all of this on an EC2 instance?
- How do I deploy / deliver my work safely and reliably?

This is Not a New Problem



The Solution


Don’t be a pirate, be the Navy.

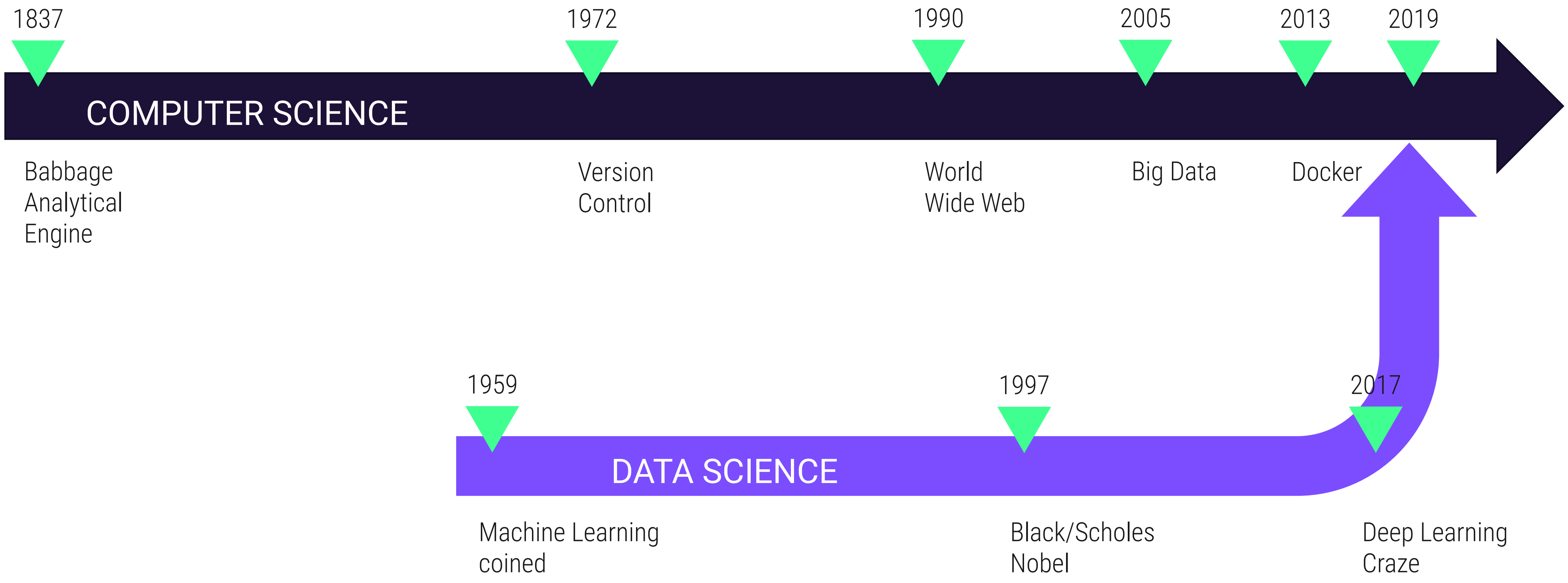
Adapt the best of SW to ML Engineering

ML Engineering

SW Engineering

Leverage the Learnings of SW Engineering

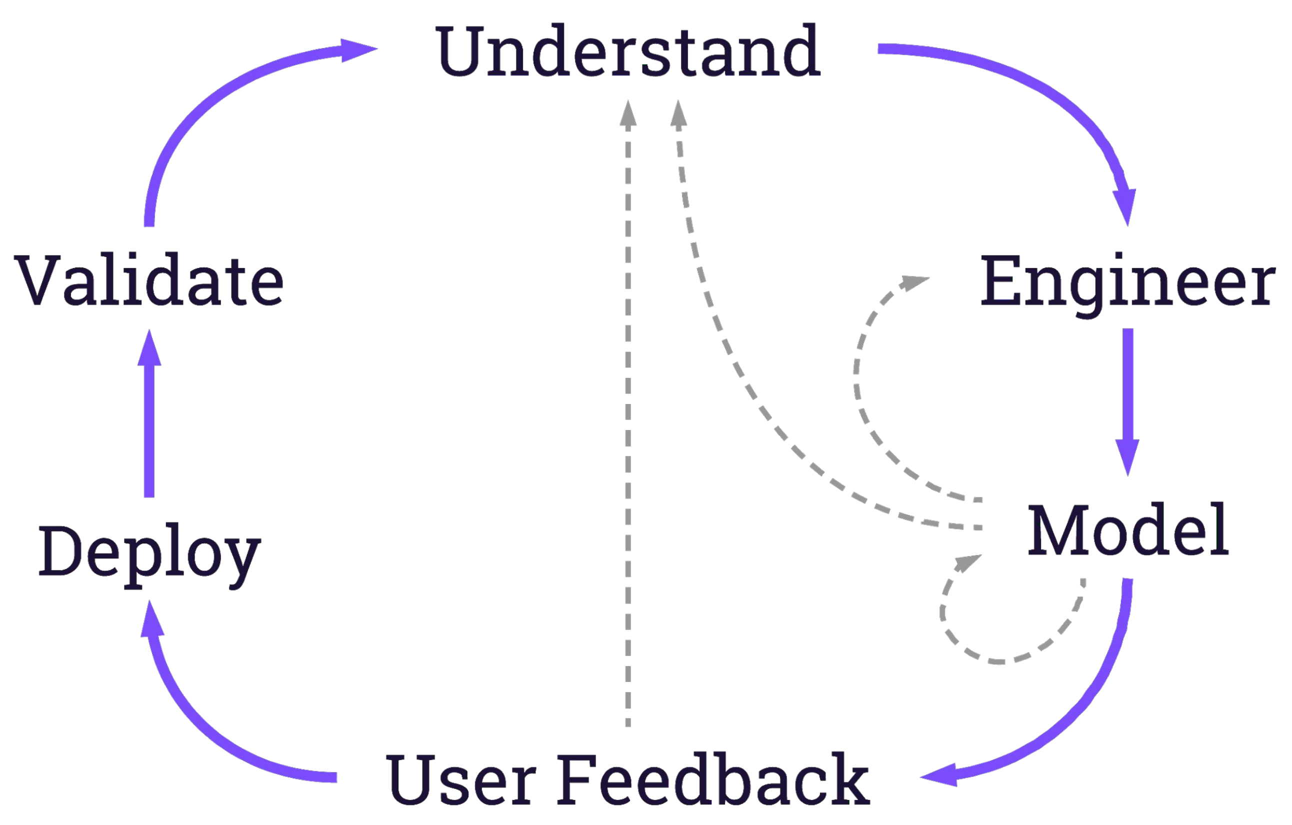
We don’t need to reinvent the wheel



Adapted from: https://twitter.com/SubbuBanerjee/status/1043993954033766400

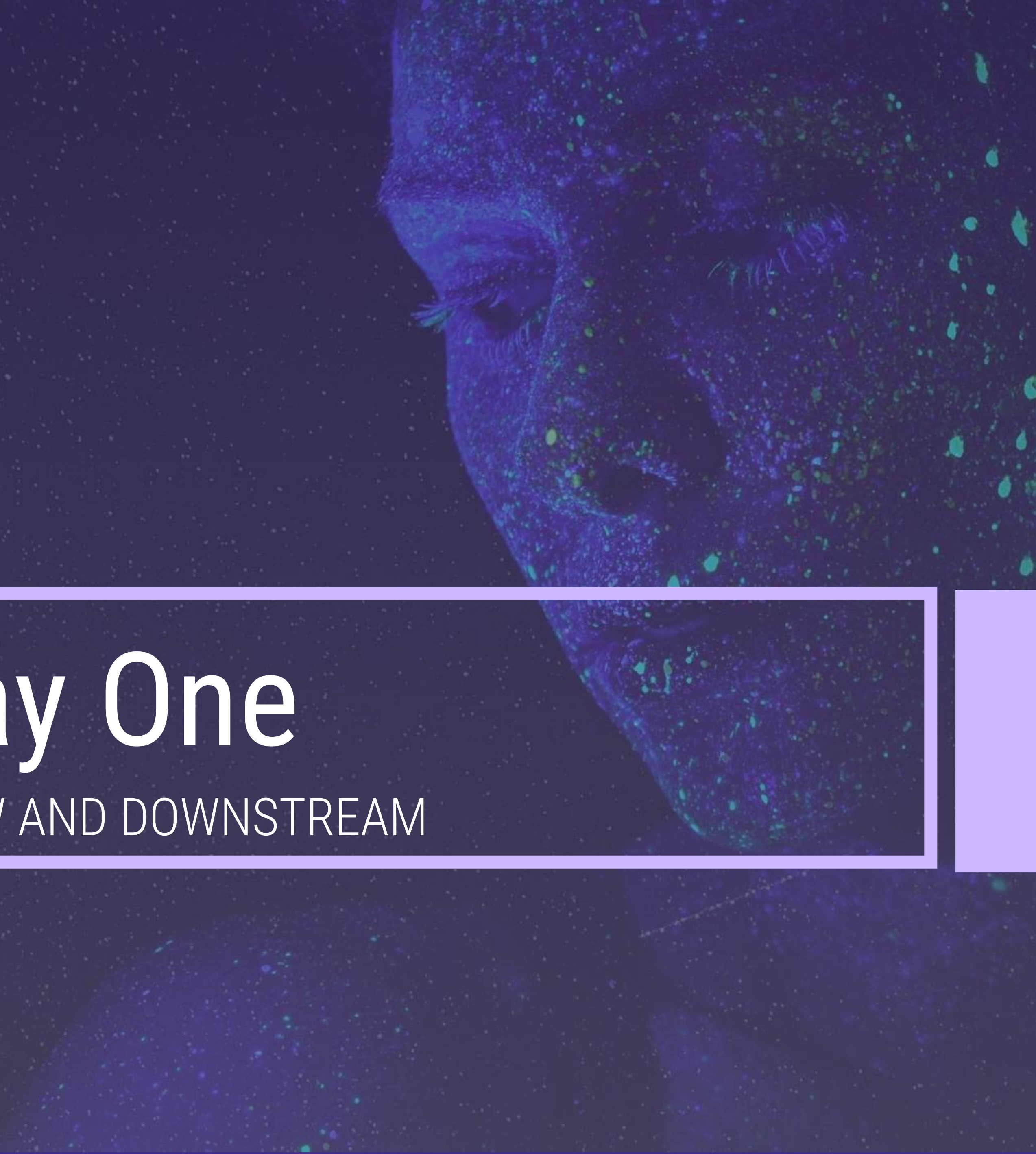
Agile SW Development → Lean AI

Adapting the product development process for ML



Three Key Lessons

1. Use Docker on Day One
2. Use a Structured ML Software Workflow
3. Downstream Containerization Benefits



Use Docker on Day One

IT MAKES EVERYTHING EASIER RIGHT NOW AND DOWNSTREAM

Use Docker on Day One

Takeaway #1: Using Docker for your ML projects is easier than you think and does not require switching to a new set of tools.

Takeaway #2: There are several downstream benefits of using containers early in the development life cycle.

Docker One Sentence Definition

"Docker is a computer program that performs operating-system-level virtualization, also known as containerization."

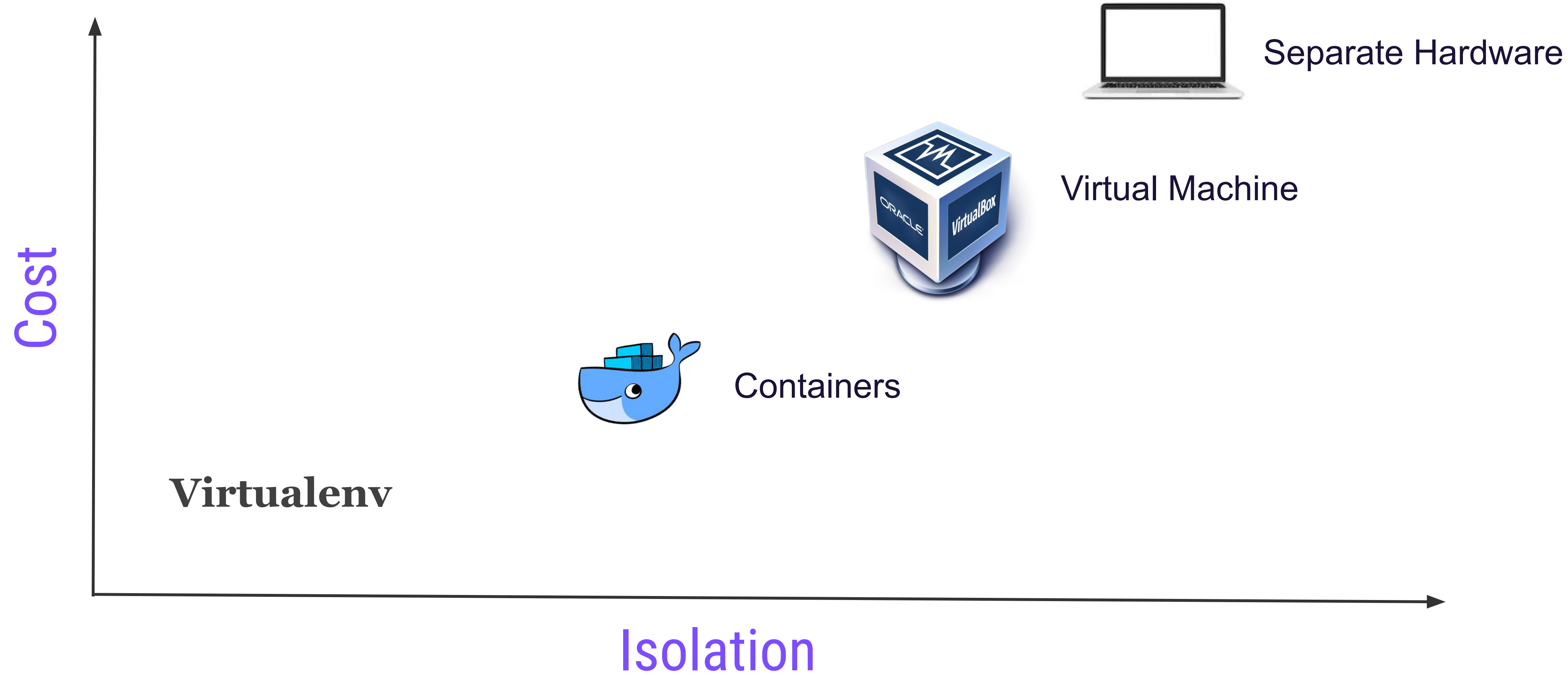
From Wikipedia, the free encyclopedia

Containers For Efficient Shipping of Product

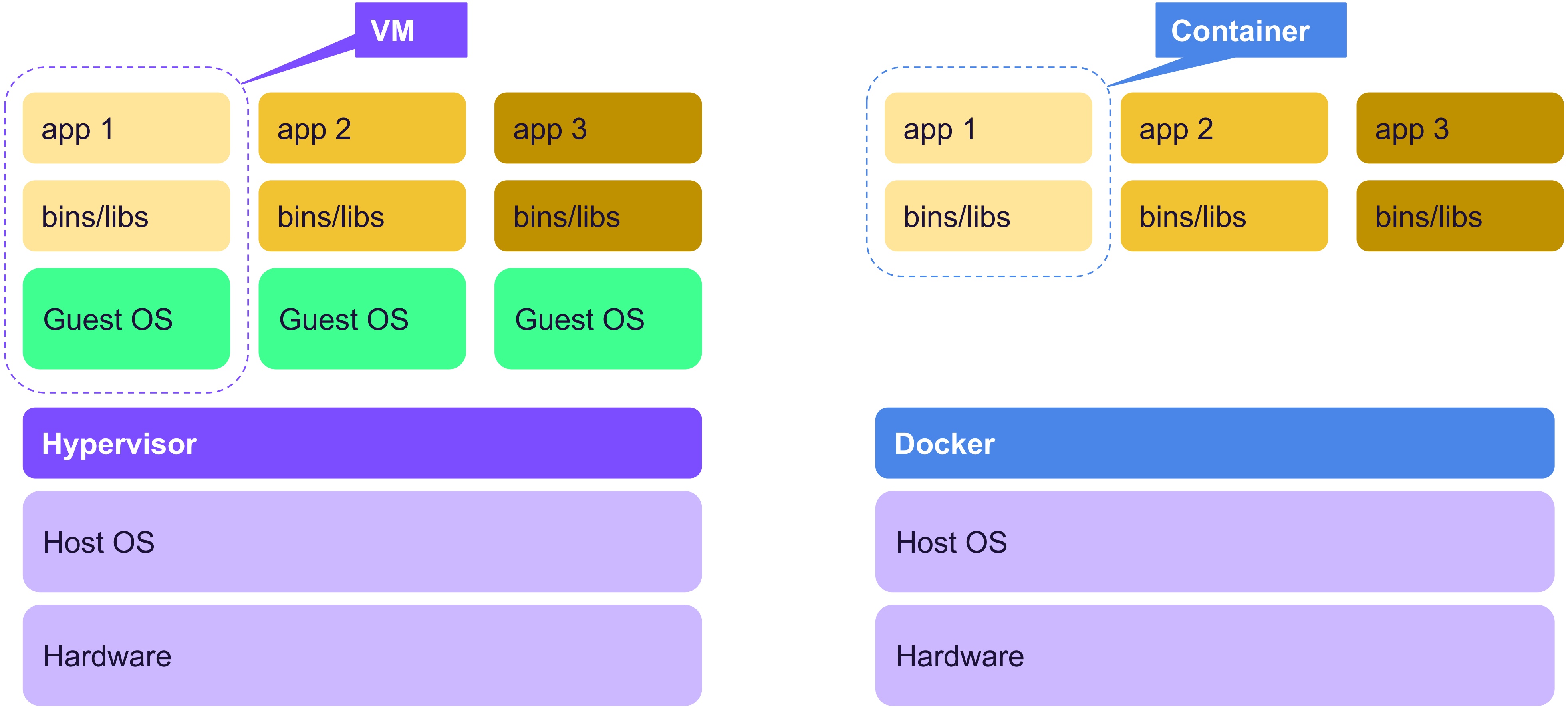
Standardized unit of fully packaged software used for:

- Local development
- Shipping code
- Deploying code

Other Methods of Isolation

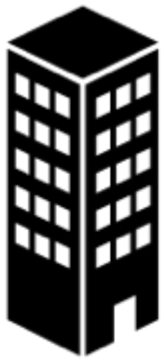


VMs vs Containers

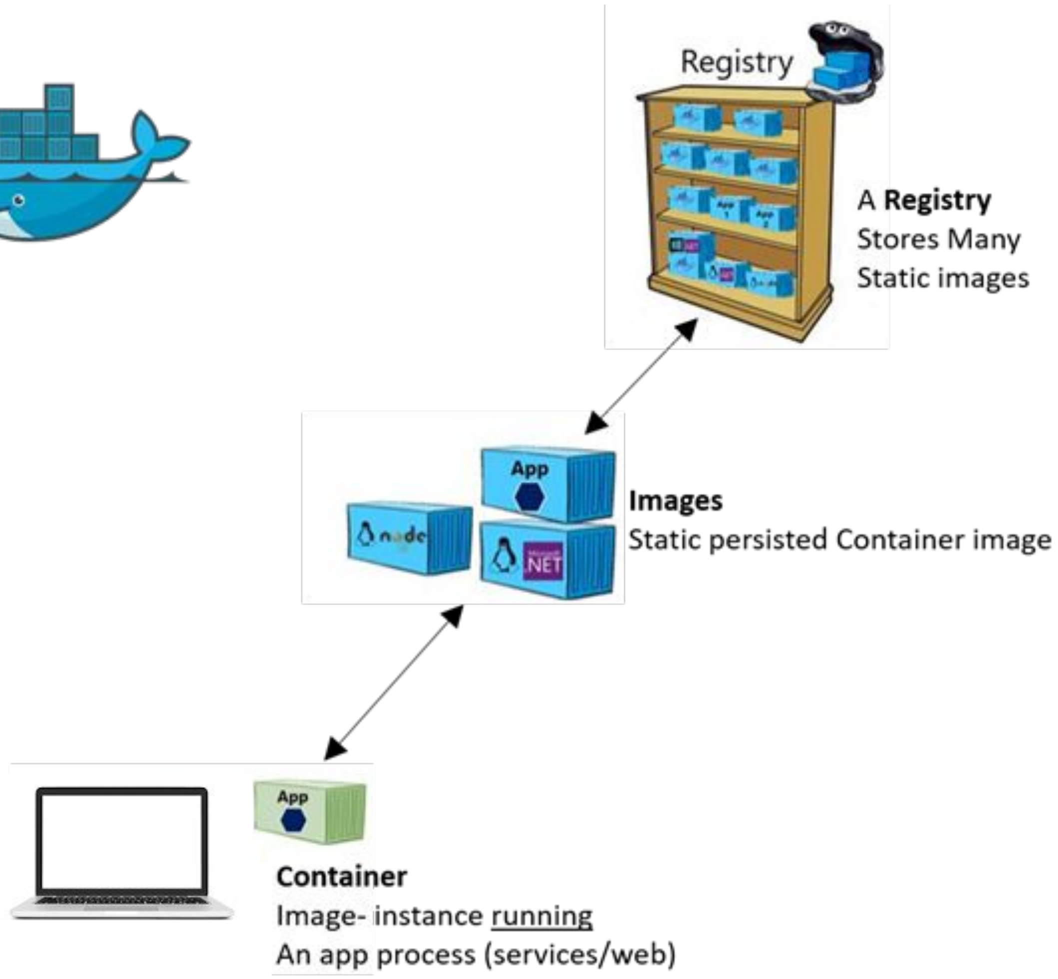


A Simplified But Helpful Analogy

Containers are like apartments and VMs are like houses



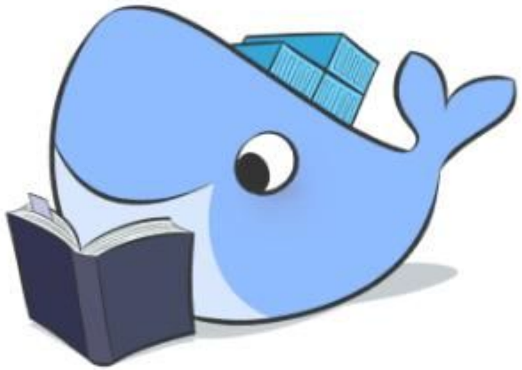
Some Docker Lingo



- Image → Repository (multiple versions)
- Repository → Registry (multiple repositories)

Helpful Docker Resources

- https://www.docker.com/sites/default/files/Docker_CheatSheet_08.09.2016_0.pdf
- https://github.com/eon01/DockerCheatSheet
- https://www.docker.com/resources



Yes But...

"I know Docker will make this easier, but I don't have the time or resources to set it up and figure that all out."

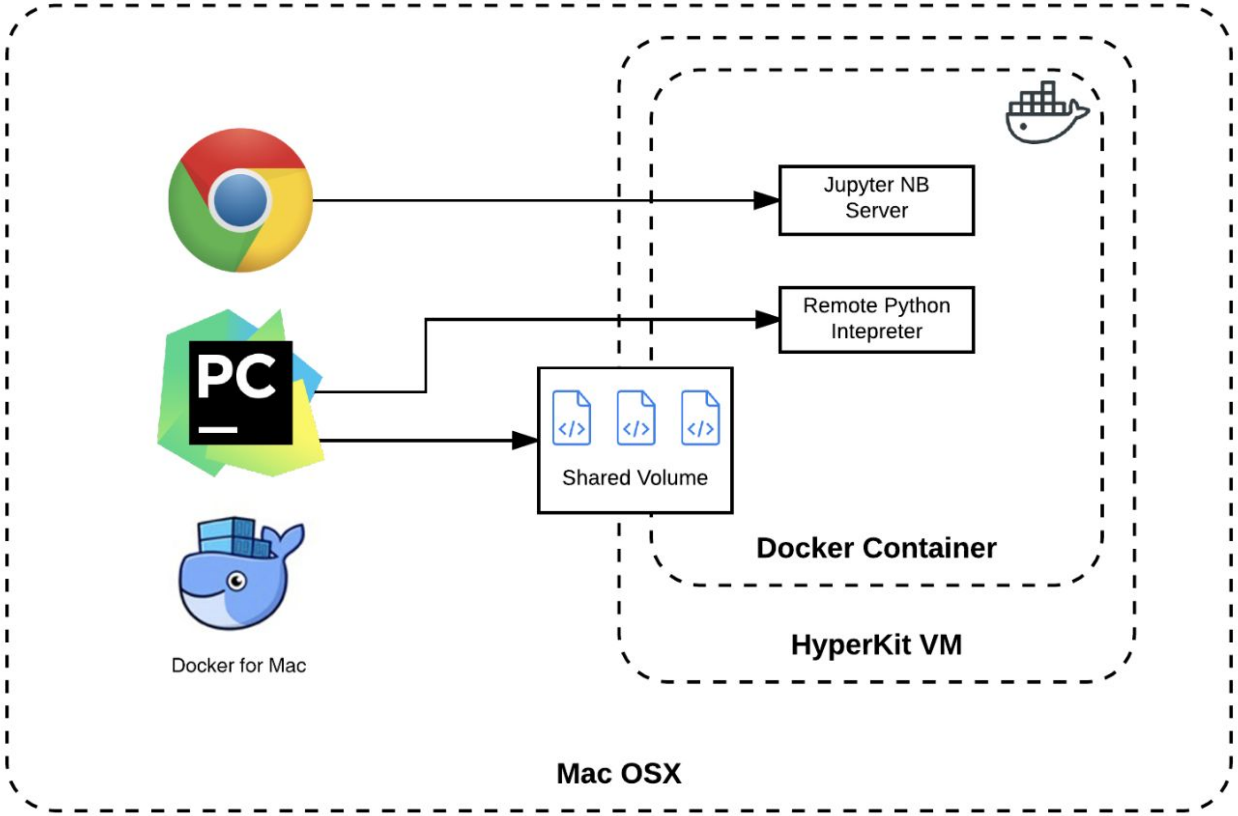
Orbyter Mission Statement

Orbyter is a framework and toolset for **helping ML teams move to a container-first workflow** to adopt DevOps best practices to increase productivity and quality of delivered work to customers.

Orbyter

What's in the box?

- Completely isolated project environments
- Ready-for-dev base images
- Consistent environments across teams
- Consistent project layouts
- Easy code portability and packaging
- **One click start-up**



Demo

Python Cookiecutter

- Automated scaffolding of new projects
- Quicker ramp up / orientation to code base
- Consistent project layouts
- Extremely configurable
- Orbyter inspired by @pjbull

New Project

cookiecutter docker-cookiecutter-data-science

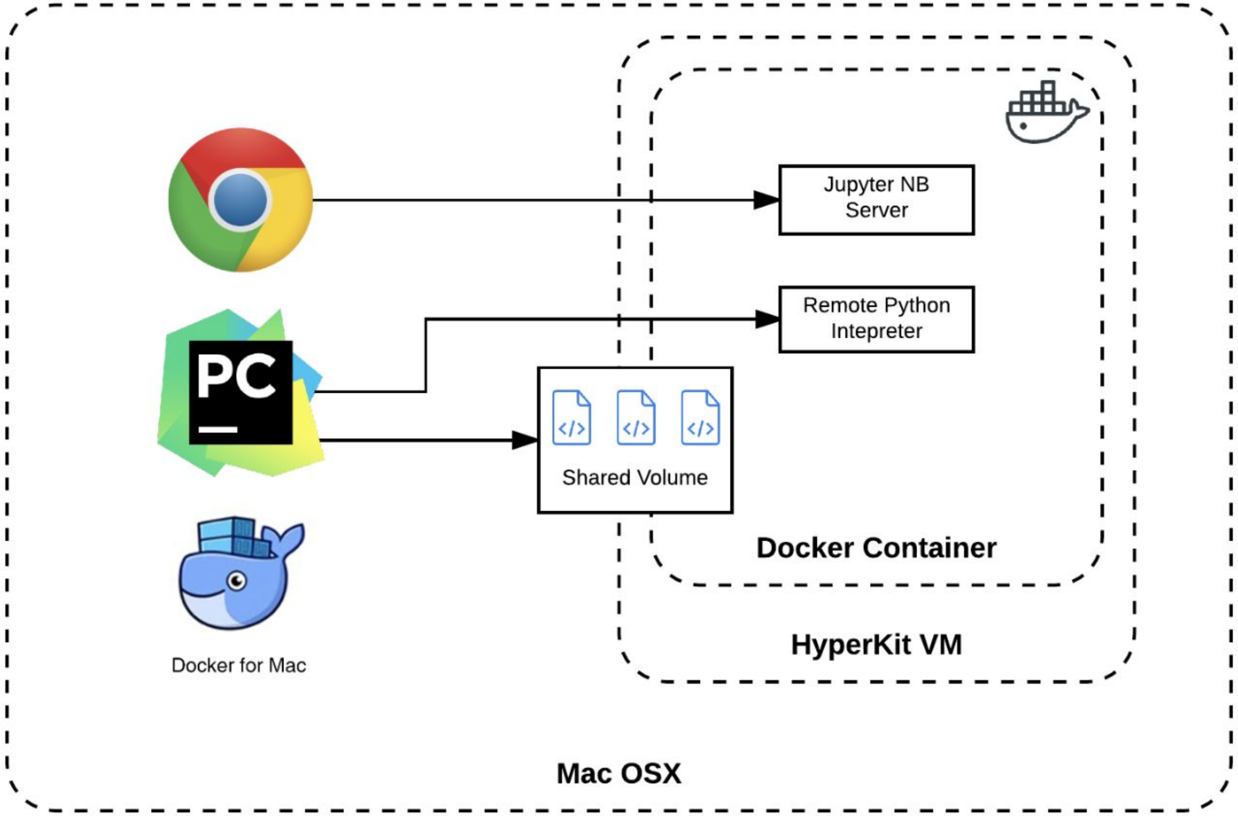
```
> ./scripts/local/start.sh
```


Existing Project

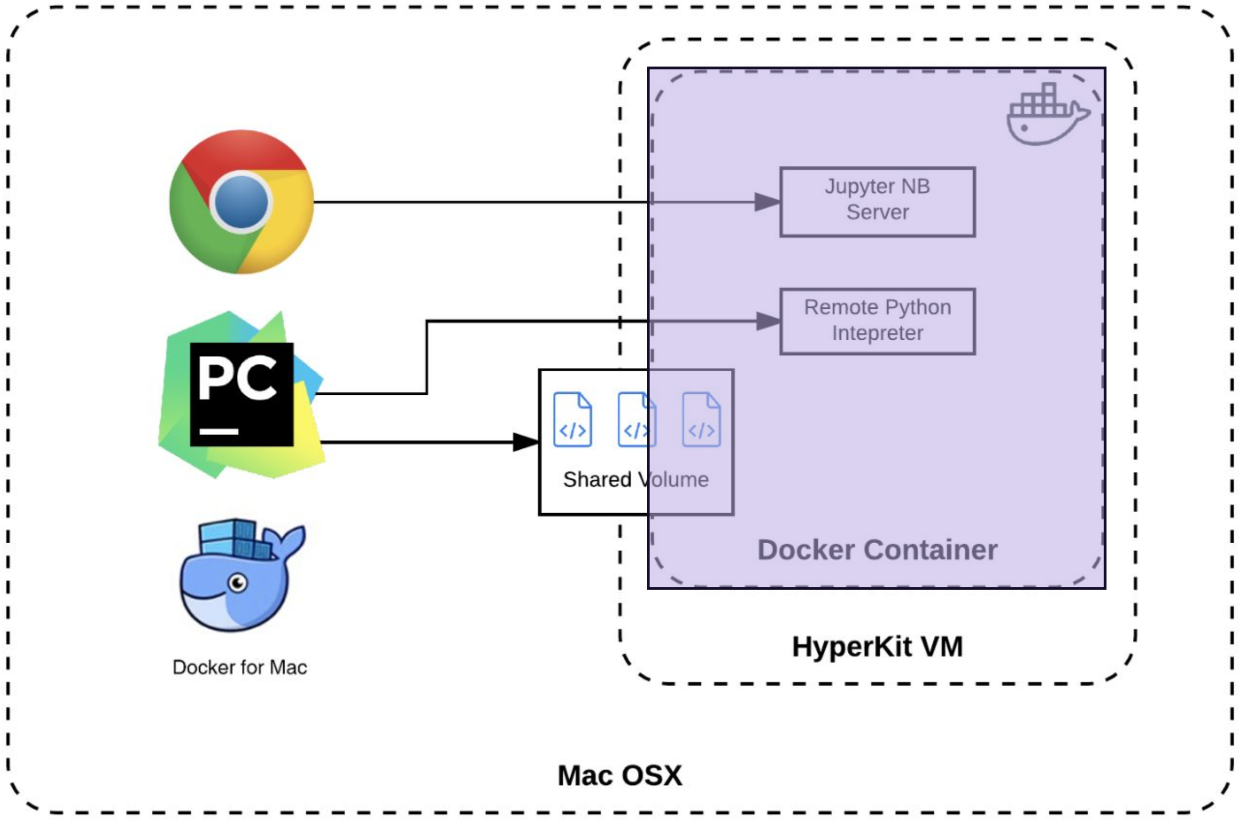
git clone: https://github.com/manifoldai/<your_repo>

```
> ./scripts/local/start.sh
```

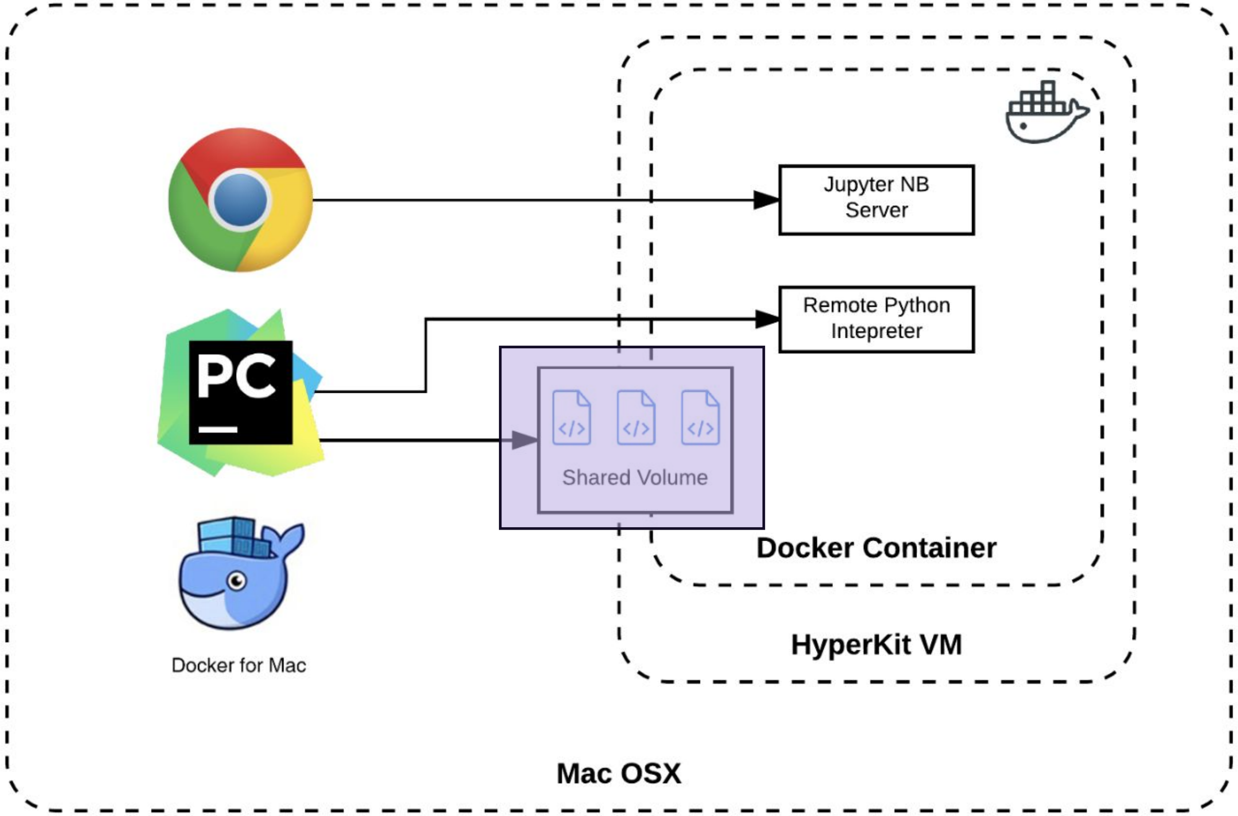

Local Dev Setup



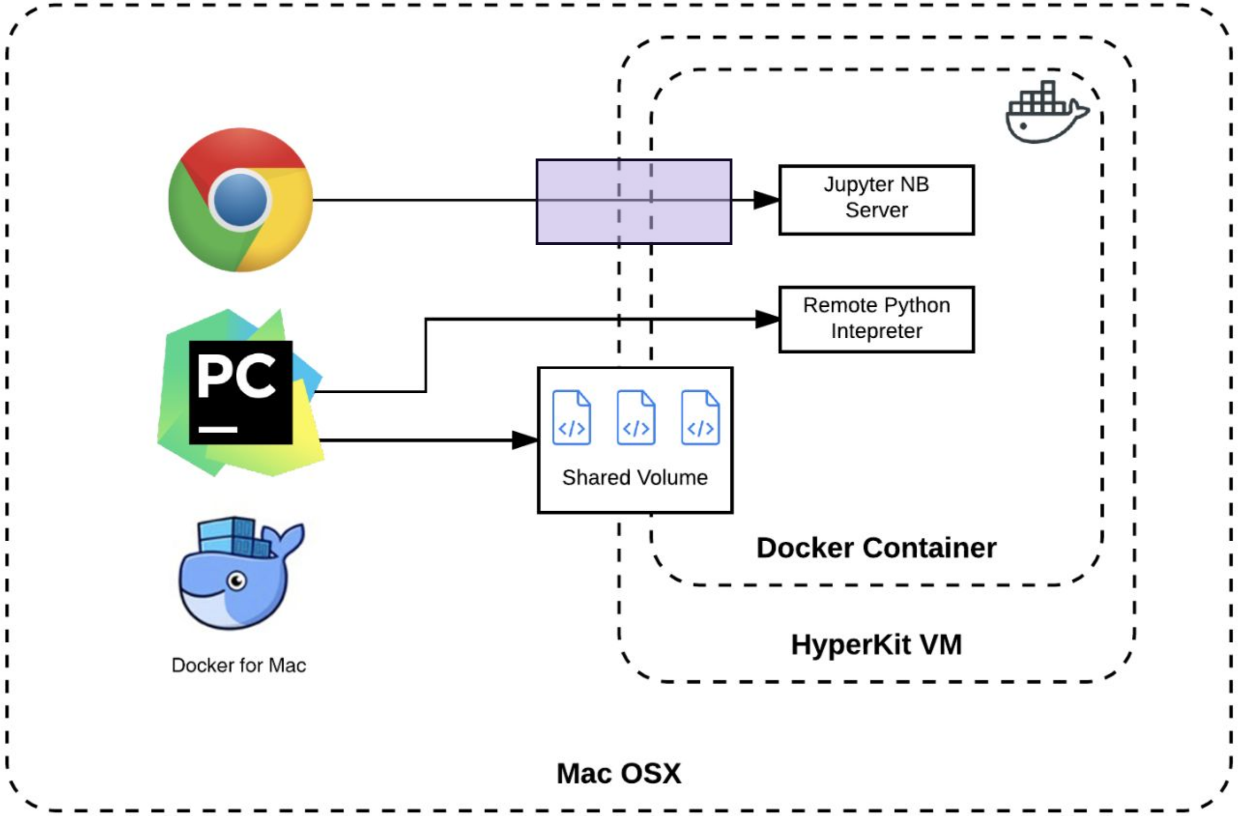
Pre-baked Dev Image



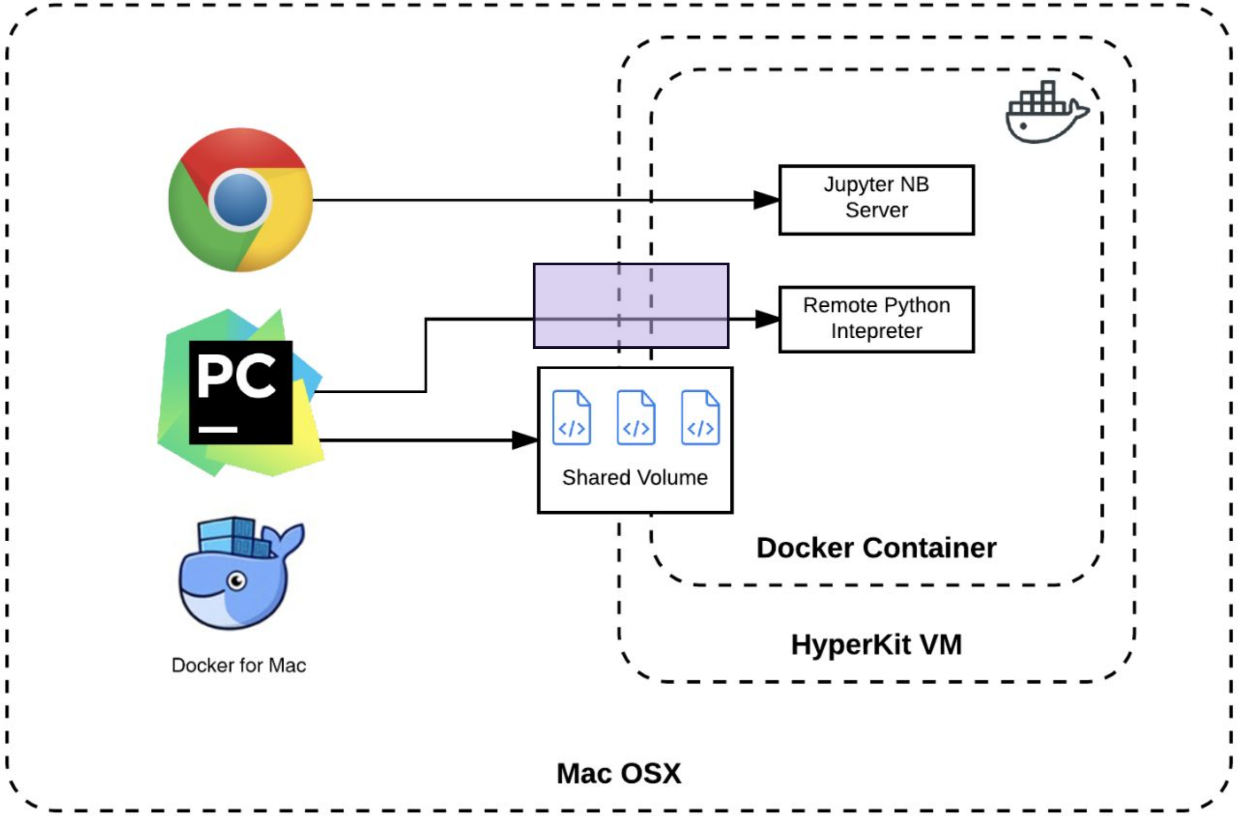
Bind Mount


Jupyter NB
Server
Remote Python
Intepreter
Shared Volume
Docker Container
Docker for Mac
HyperKit VM
Mac OSX

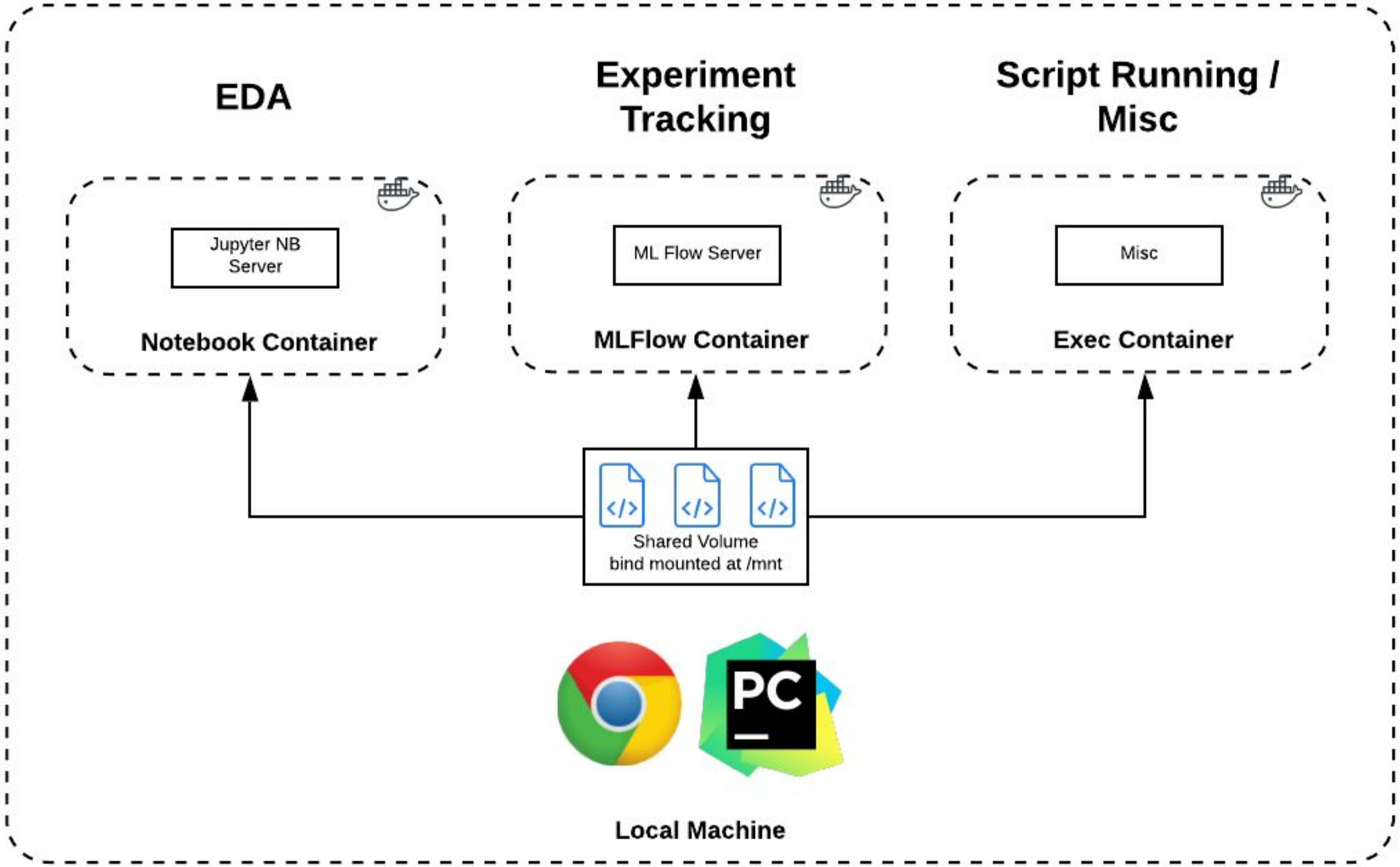
Port Forwarding



Remote Debugging



Jupyter is Just One Piece



Orbyter Resources

- **Overview–** https://www.manifold.ai/project-orbyter
- **Cookiecutter–** https://github.com/manifoldai/docker-ml-cookiecutter
- **Base Image Dockerfile–** https://github.com/manifoldai/orbyter-docker
- **Dockerhub Repo–** https://hub.docker.com/r/manifoldai/docker-ml-dev

Use A Disciplined ML Workflow

Use a Disciplined ML Workflow

Takeaway: There are a number of best practice ML engineering processes and tools you can use, right now, to make your life an an MLE easier.

You will:

- Use a scaffolded repo to train and evaluate a model.
- Learn how **track experiments effectively**.

Considerations for Applied ML

We want many of the tried and true principles from SW engineering (readability, orthogonality, 12 Factor App, etc.), **plus**:

- **Flexibility**—being able to rapidly iterate, add new data sources and features, try new algorithms.
- **Observability**—being able to observe the inputs and outputs of every stage in the pipeline.
- **Reproducibility**—being able to reproduce results, across team members or over time.

An Incomplete MLE Manifesto

Inspired by the 12 Factor App

- Standardize repo structure.
- Use the pipeline abstraction.
- Log using the right tools.
- Use configuration files to run experiments.
- Track experiments using a tool.
- Standardize your metrics.
- Automated tests using a subset of real data.
- Police code quality.

12 Factor App https://12factor.net

Cookiecutter Encodes This Manifesto

IT MAKES IT EASIER TO DO THE RIGHT THING

- Demo repo scaffolded using docker-ml-cookiecutter: https://github.com/manifoldai/odsc_west
- Clone to follow along:
  - `git clone git@github.com:manifoldai/odsc_west.git`
  - `git clone https://github.com/manifoldai/odsc_west.git`

Standardized ML Repo Structure

OPINIONATED FROM EXPERIENCE

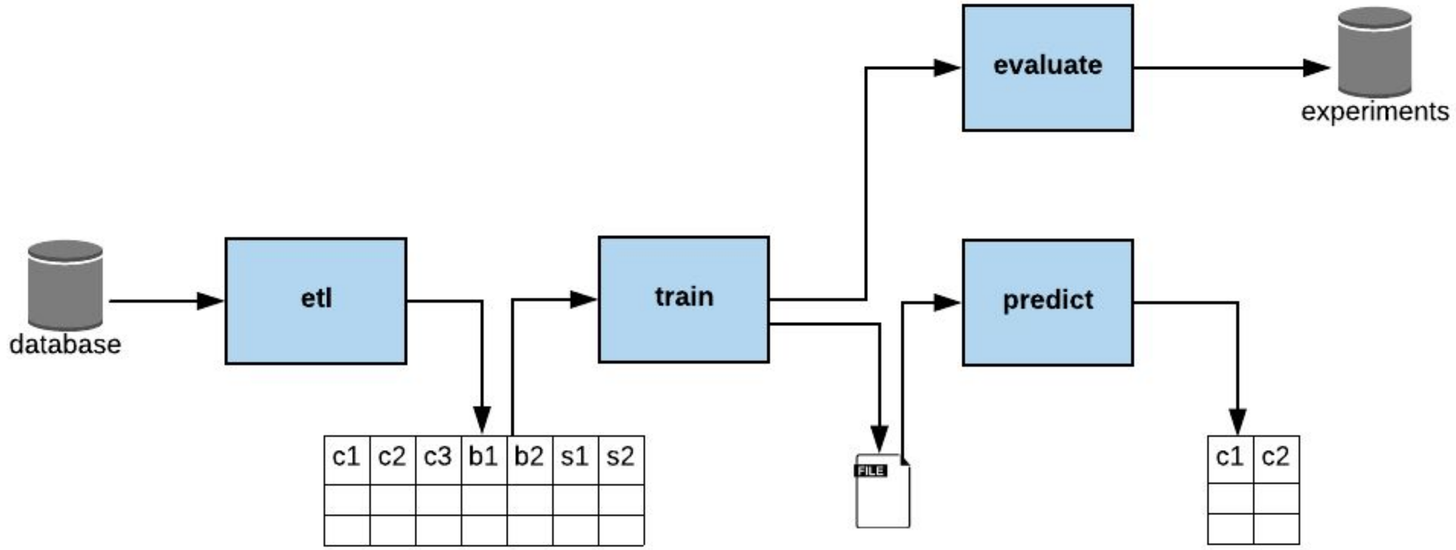
```
> .circleci
> .pytest_cache
> configs
> data
> docker
> docs
> experiments
> logs
> model_cache
> notebooks
> scripts
> strata_nyc
  .env
  .gitignore
  env_template
  LICENSE
  logging.yml
  pull_request_template.md
  pyproject.toml
  README.md
  tox.ini
```

```
v strata_nyc
  > features
  > models
  v scripts
    > __pycache__
      etl.py
      evaluate_test.py
      evaluate.py
      predict_test.py
      predict.py
      train_test.py
      train.py
  v util
      .gitkeep
      config.py
      logging.py
  > viz
    __init__.py
```


Use the Pipeline Abstraction

FLEXIBILITY TO ADD FEATURES, PARTIALLY RUN, AND MORE

- Use `click` to make atomic and idempotent pipeline stages that can be orchestrated using `bash` (simple) or Airflow (more complex)
- Use sklearn pipelines for core ML flows



Log Using the Right Tools

- Use configurable `logging` library.
    - Do not use `print`
    - It has many drawbacks
- Use `parquet` for intermediate data.
    - Do not use CSV
    - Parquet is faster, smaller, and typed!

```yaml
# We can raise the message level and add additional handles to specific
# to specific modules. The names of the import must match the key
# name under loggers
loggers:
    predict:
        level: INFO
        handlers: [console, info_file_handler, error_file_handler]
        propagate: no
    train:
        level: INFO
        handlers: [console, info_file_handler, error_file_handler]
        propagate: no
    evaluate:
        level: INFO
        handlers: [console, info_file_handler, error_file_handler]
        propagate: no

# We can raise the message level and add additional handles
# to root, i.e., the function called from the command line
# These are displayed as root, or __main__ in the message
root:
    level: DEBUG
    handlers: [console, info_file_handler, error_file_handler]
```


Use Config Files for Reproducible Experiments

RUN EXPERIMENTS WITHOUT CHANGING SOURCE CODE

config.yml

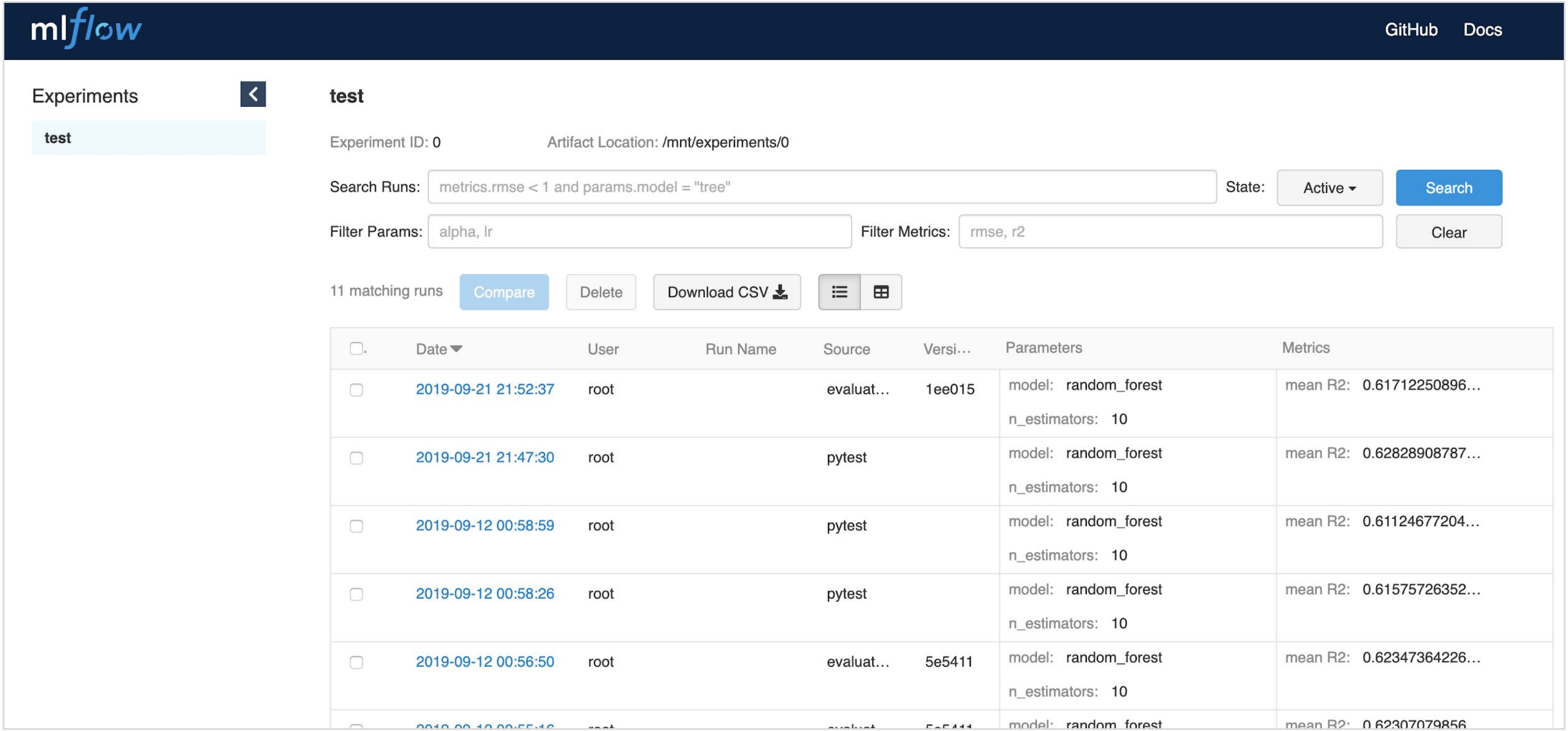
```yaml
global:
  raw_data_dir: "data/demo/raw"
  processed_data_dir: "data/demo/processed"

model:
  model_name: "random_forest"
  model_params: {}
  # dir to save model weights
  model_path: "model_cache/demo_model.pkl"

evaluate:
  # Flag to retrain the model or not. If False, then t
  retrain: True
  # MLFlow experiment name
  experiment_name: "demo"

predict:
  model_path: "model_cache/demo_model.pkl"
  data_path: "data/demo/processed/X.pqt"
  predictions_path: "data/demo/predictions/yhat.pqt"
```

```
python
odsc_west/scripts/evaluate.py
configs/config.yml
```

Track Experiments Using a Tool

LIKE MLFLOW

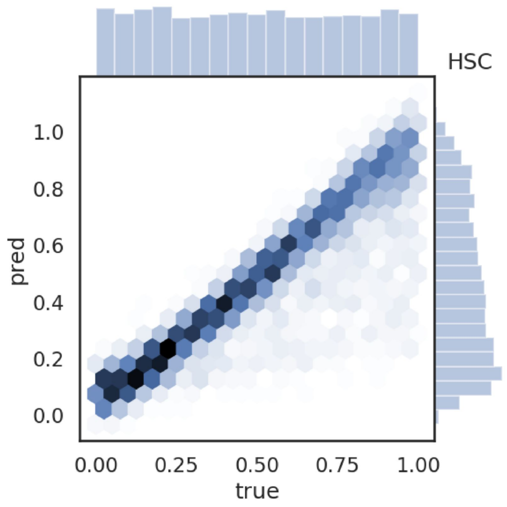
- Parameters
- Metrics
- Artifacts

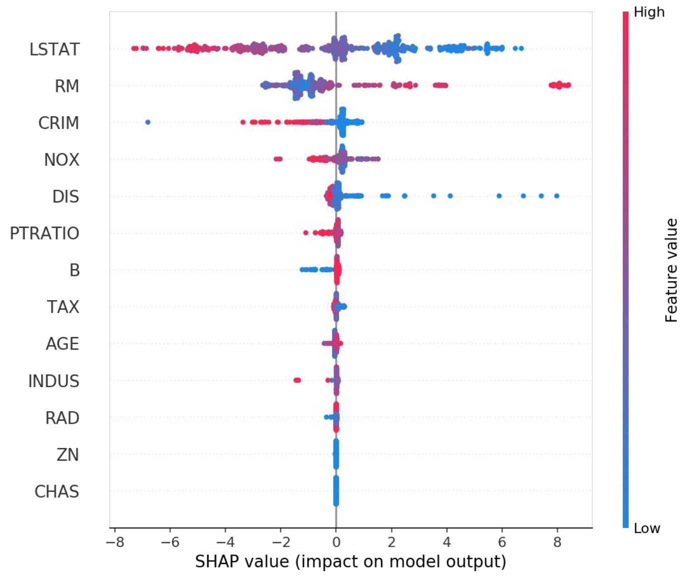


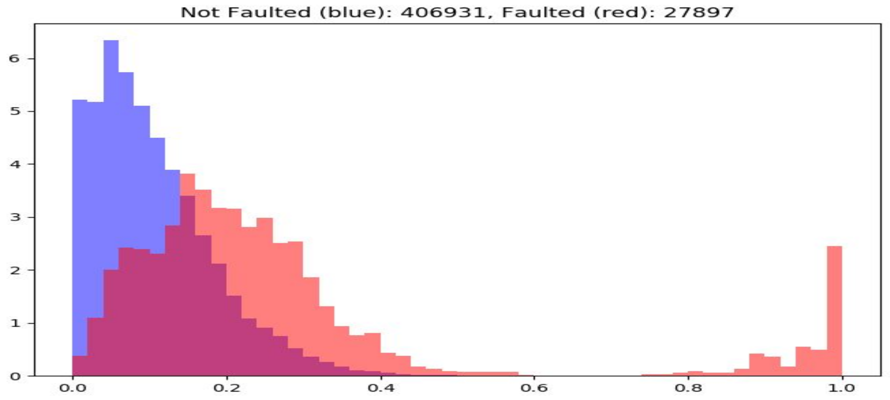
Standardize Your Metrics

CONSISTENCY HELPS REDUCE COGNITIVE LOAD

- Aggregate Metrics
  - Train, validation, test loss and metrics
  - ROC, precision-recall, calibration curve for classification
  - Shapley feature importance
- Individual Metrics
  - prediction probability distribution for classifiers
  - Y vs. yhat hexbin for regression
  - Representative examples







Automate Tests Using a Subset of Real Data

- Create a small piece of test data
  - For us this is under `/mnt/data/tests/`
  - N = 100
- Test using `pytest`
  - Unit test functions
  - Click tests to ensure each pipeline stage works

pytest

```python
import pytest
from click.testing import CliRunner

from strata_nyc.scripts.evaluate import evaluate


@pytest.mark.parametrize("config_file", [("/mnt/configs/test_config.yml")])
def test_evaluate(config_file):
    runner = CliRunner()
    result = runner.invoke(evaluate, [config_file])
    assert result.exit_code == 0
```


Police Code Quality

MAKE IT EASY TO WRITE CLEAN CODE

- Block merging to master
- Require code reviewed pull requests
- Lint using black and flake8
- Require tests to pass
- Use continuous integration
  - CircleCI or GitHub
  - Docker container makes this easy
- Use a coverage tool [coming soon!]

```
#/bin/bash
#
# Local tests of CI jobs
# Run this from CI job docker container
set -ex

echo 'Running black'
black --check strata_nyc

echo 'Running flake'
flake8 strata_nyc

echo 'Running pytest'
pytest strata_nyc

echo 'Finished tests'
```


An Incomplete MLE Manifesto

Inspired by the 12 Factor App

- Standardize repo structure.
- Use the pipeline abstraction.
- Log using the right tools.
- Use configuration files to run experiments.
- Track experiments using a tool.
- Standardize your metrics.
- Automated tests using a subset of real data.
- Police code quality.

12 Factor App https://12factor.net

Demo

- Evaluate simple baseline model
  - `python orbyter_demo/scripts/evaluate.py configs/config.yml`
- Look at MLFlow
- Train a model, change the config file, train again
- Predict using a trained model
- Run `autoformat.sh` and `ci.sh`
- Run unit tests using `pytest`
- Look at CI on GitHub

Downstream Containerization Benefits

Deployments Are Hard

Leverage Existing Deployment Infrastructure

Containerization Gives You Options

By moving to a Docker-first workflow, you are well positioned to take advantage of a rich ecosystem of tooling and libraries for training and deployments in the cloud.

amazon web services

kubernetes

Orbyter

Google Cloud Platform

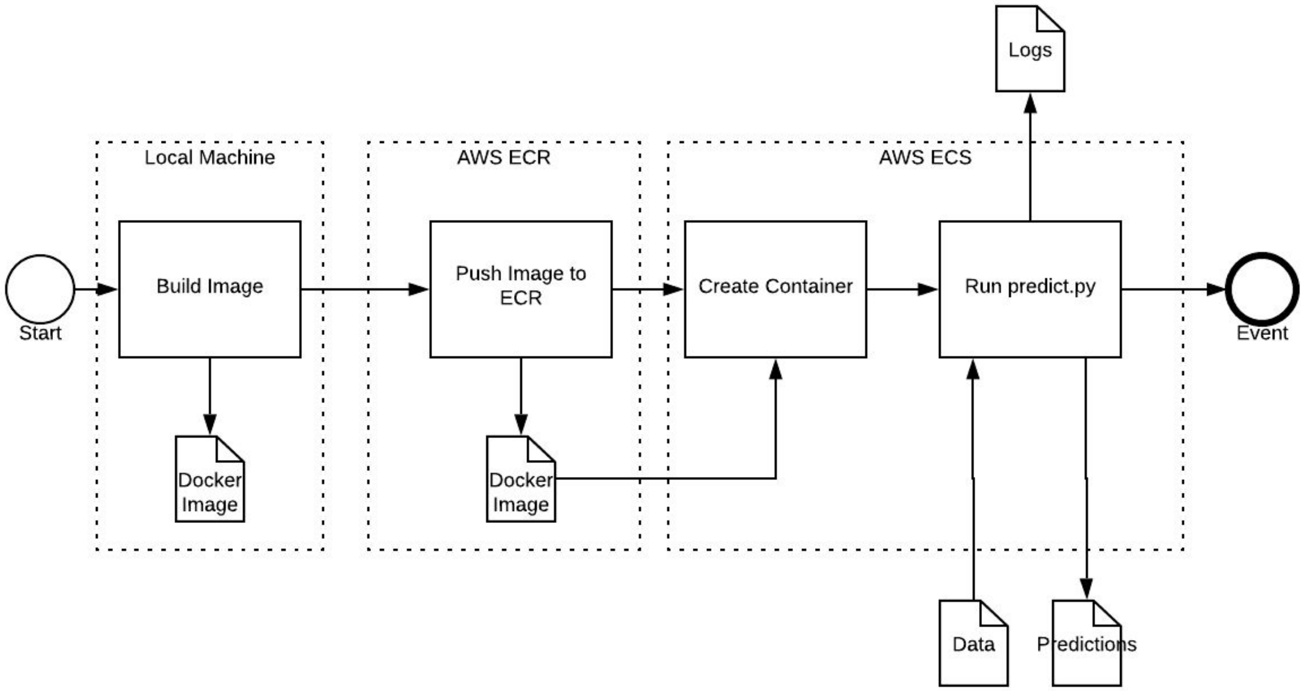
HELM

Microsoft Azure

Experiment at Scale

Most experiments will yield bad results. To get value, you'd need to perform lots of experiments efficiently.

This is easy with Docker, e.g. using AWS Batch.



BYOC - Bring Your Own Container

For training and serving

Cloud AI

Amazon SageMaker

Azure Machine Learning

Stay tuned!

Real World Example #1

Client: Bio-pharma research company

Objective: Train a deep learning model to predict drug effects to prevent adverse events

How did Docker help: Easily converted Docker training images to Singularity images and deployed to HPC using by research team

Real World Example #2

Client: Broadcast media company

Objective: Train models to predict new song popularity

How did Docker help: Trained multiple models and baked into production images pushed to customer registry that runs nightly jobs on AWS to generate song predictions

Real World Example #3

Client: Laser manufacturing company

Objective: Train predictive maintenance models to inform early intervention and prevention of field unit failures

How did Docker help: Client infrastructure in Azure and deploying to Azure Kubernetes Service (AKS) using Azure Container Registry (ACR) was very easy

Conclusions

Adapt the best of SW to ML Engineering


Don’t be a pirate, be the Navy.

MANIFOLD

GET THE SLIDES:
www.manifold.ai/2019ODSCWest

THANK YOU

Dr. Sourav Dey | sdey@manifold.ai | @resdntalien
Alex Ng | ang@manifold.ai | @abng88