Jupyter / Docker Architecture

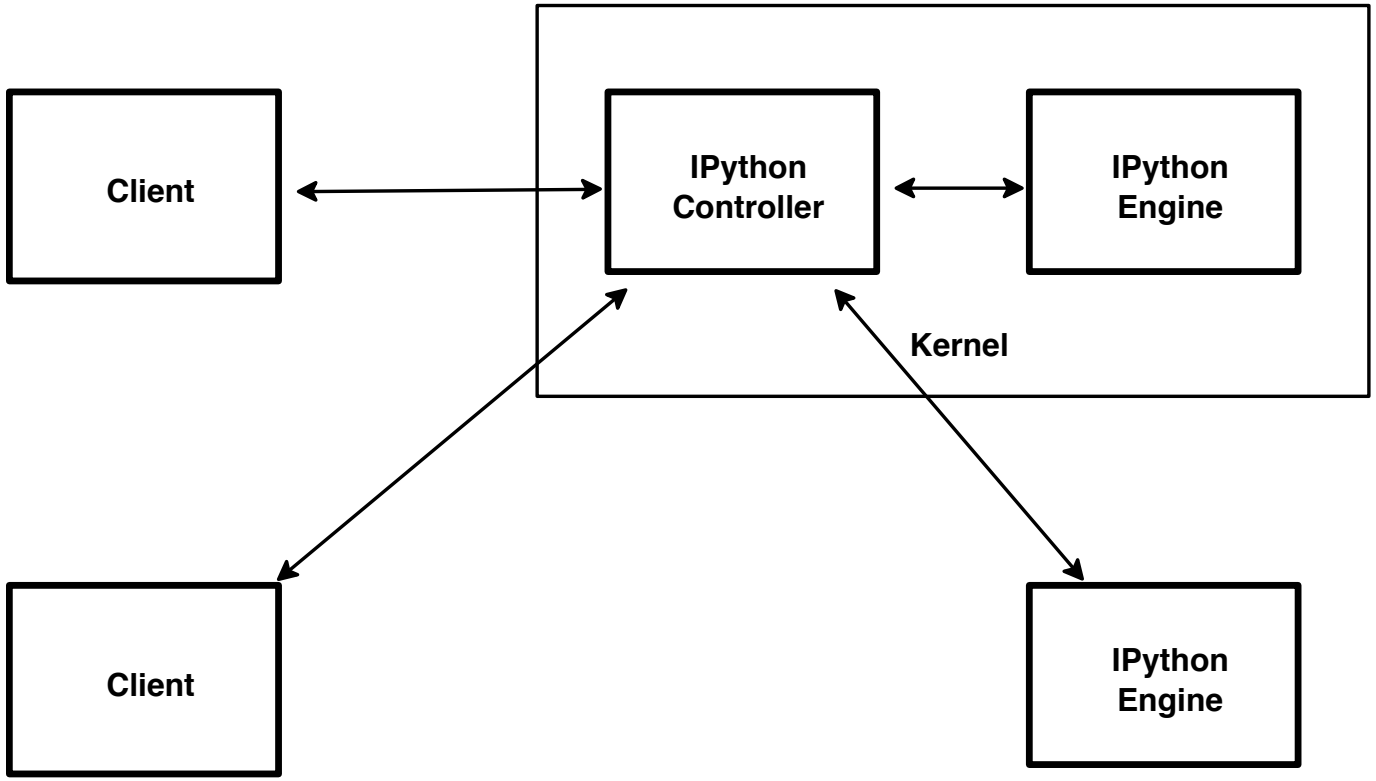
High Level Overview of what we are doing

Topics

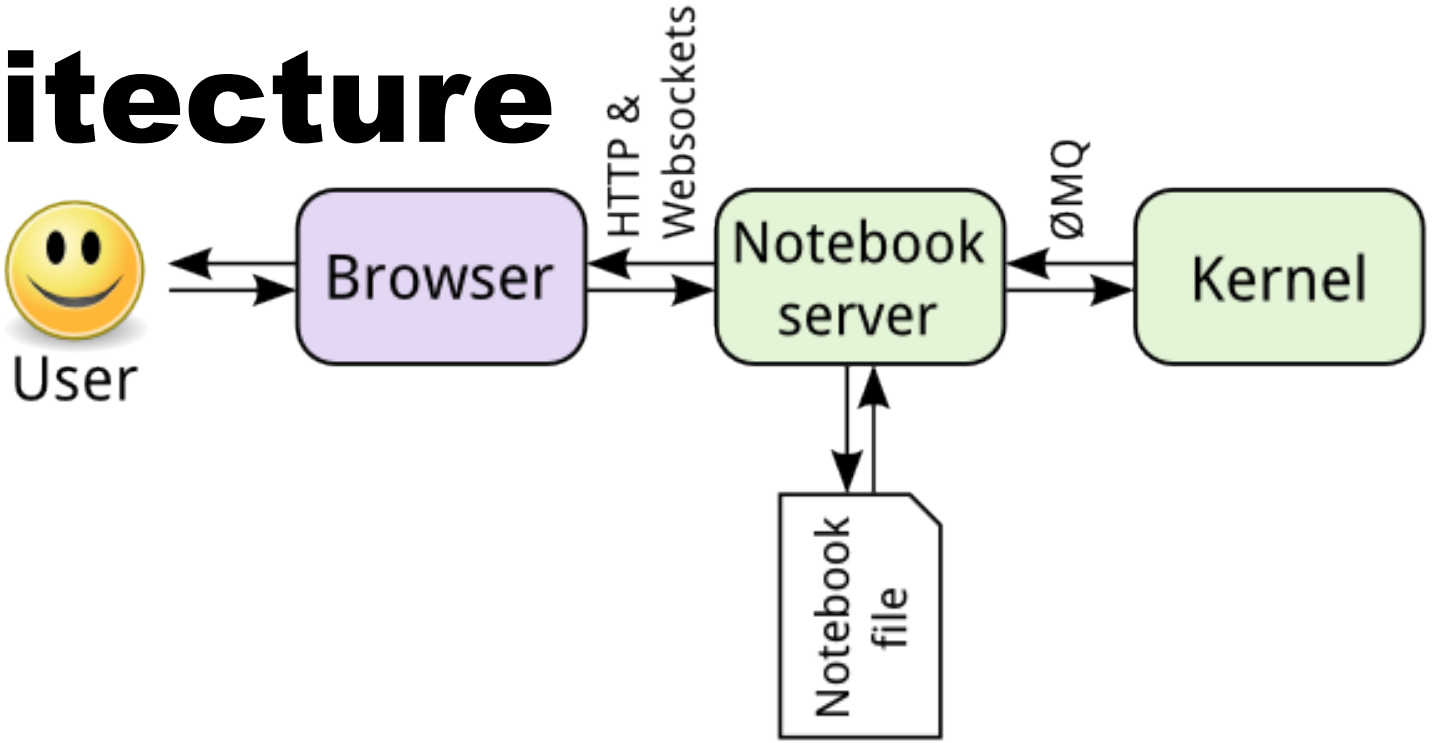
- Jupyter / Docker Architecture
- Using Dockerized Jupyter System
- Building and distributing Docker images
- Virtual Machines & Containers
- Existing Work on Building Docker Images for Jupyter
- Building on Existing Docker Images for Juptyer

IPython Architecture

- Multiple processes, maybe multi-machine
- Engines evaluate code
  - ipyparallel uses multiple engines
- Controller coordinates communication
- Client provides User Interface
  - Command Line (REPL)
  - GUI (Qt)
  - Web server proxying for browser



Jupyter Architecture



- Generalization of IPython
  - Language agnostic
  - Kernel evaluates code
  - Web server acts as proxy client
    - Same OS instance as Kernel
  - Browser provides rich multi-media user interface
    - Lots of JavaScript code running on browser

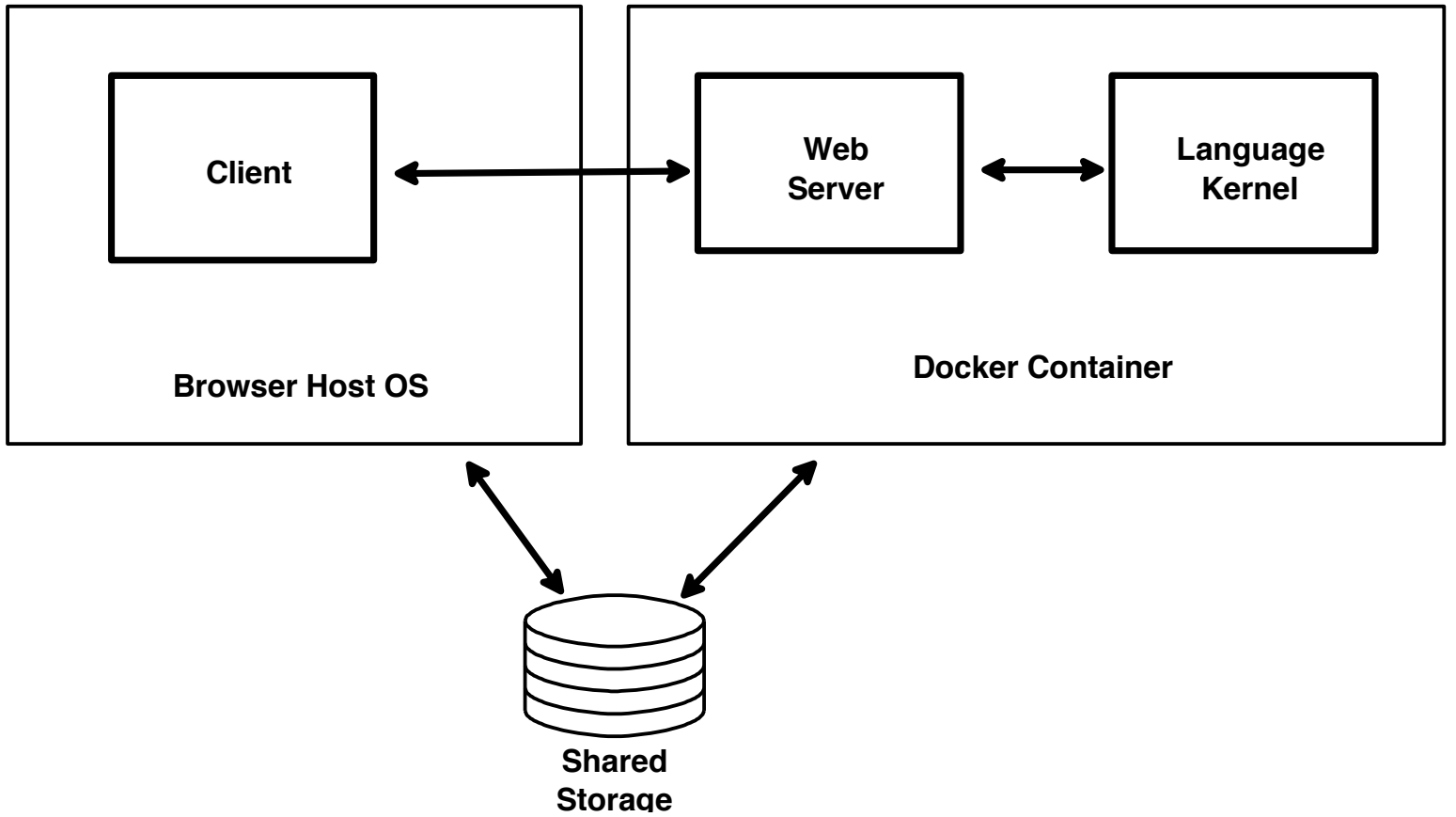
Detail in Multi-Machine System

If server and kernel on a different machine than the browser:

- Server can up/down-load notebook file to/from browser
- Persistent data for kernel non-local to browser's host
- Ideally, browser's host and server's host share a filesystem
  - "Big" data should be "near" kernel: high bandwidth & low latency
  - For cloud-hosted server, cloud-storage handy: e.g., Google Drive

Docker-based Jupyter system

- Docker Container will not have a persistent filesystem
  - Allows Docker image to be updated / upgraded
- Web server started upon container initialization.
- Client must authenticate
  - by server-supplied token
  - server has no way to provide & save user-specific credentials



Multi-User System

- More advanced, not being covered
- Run JupyterHub on server-side, and it will
  - Manage authentication
  - Start a Docker container for each user
  - Act as proxy between browser and web server in container
  - May start containers on multiple machines to spread out the load

Using Dockerized Juptyer

What we can do
once we have
all the pieces put together

Twitter: @paulebuis #PyDataIndy

Example System

- Windows 10 (Education, Pro, or Enterprise)
  - Hyper-V role installed as optional feature
  - Docker Desktop CE starts Linux VM on system boot
  - Docker containers run in Linux VM
  - Use PowerShell to interact with Docker

- Upcoming WSL2 will be used with future version of Docker
  - WSL2 provides tighter integration of Windows and Linux
  - Linux kernel “inside” Windows 10, still using virtualization HW

Start-up Procedure (1)

Command Line (Windows PowerShell):

```
docker run --rm -p 8888:8888 -v $HOME/work:/home/joyvan img
```

`img` name of Docker filesystem image to start in new container

`--rm` remove changes to filesystem image on exit

`-p 8888:8888` port map between `localhost` and container

`-v abc:xyz` volume map between user's home and container

Will fail if container already running using port 8888

Start-up Procedure (2)

- Server starts up in container and prints out URL w/ token
  - URL ends with `:8888/`
- Start browser
- Copy URL starting with `:8888/` from CLI where container started
- In browser's URL entry field:
  - Type `http://localhost`
  - Paste text copied from CLI
  - Should end up with something like

    `http://localhost:8888/?token=123abc789def`

Start-up Procedure (3)

- Browser displays Jupyter start page
  - JuptyerLab
    - Command palette and list of available kernels
  - Jupyter Notebook
    - Listing of directories & files in mapped directory

Misc. Docker Maintenance (1)

To list available images:

```
docker image ls
```

To remove an image:

```
docker rmi foouser/barimage:revision
```

- Can’t remove an image that is used by another image
- Can’t remove an image used by an active or inactive container

Misc. Docker Maintenance (2)

To remove all untagged, unused images:

```
docker image prune
```

To remove all inactive containers:

```
docker container prune
```

To list active containers:

```
docker container ls
```

To remove an active container:

```
docker kill foo_bar
```

Building a Docker file system image (1)

Running Docker-Build is easy:

```
docker build -t paulbuis/cool-jupyter:2019-08-20
```

paulbuis	account name on Docker Hub
cool-juptyer	tag name of image
2019-10-11	revision of tag (**latest** if omitted)
	I use YYYY-MM-DD tags

Building a Docker file system image (2)

Understanding what happens when **`docker build`** runs is not easy:

- Delayed to later part of talk

Distributing Image via DockerHub

Place copy of image on hub.docker.com:

docker push paulbuis/cool-jupyter:2019-08-20

will be prompted for credentials

Fetch copy of image from hub.docker.com:

docker pull paulbuis/cool-jupyter:2019-08-20

Virtual Machines & Containers

Heavy & Light

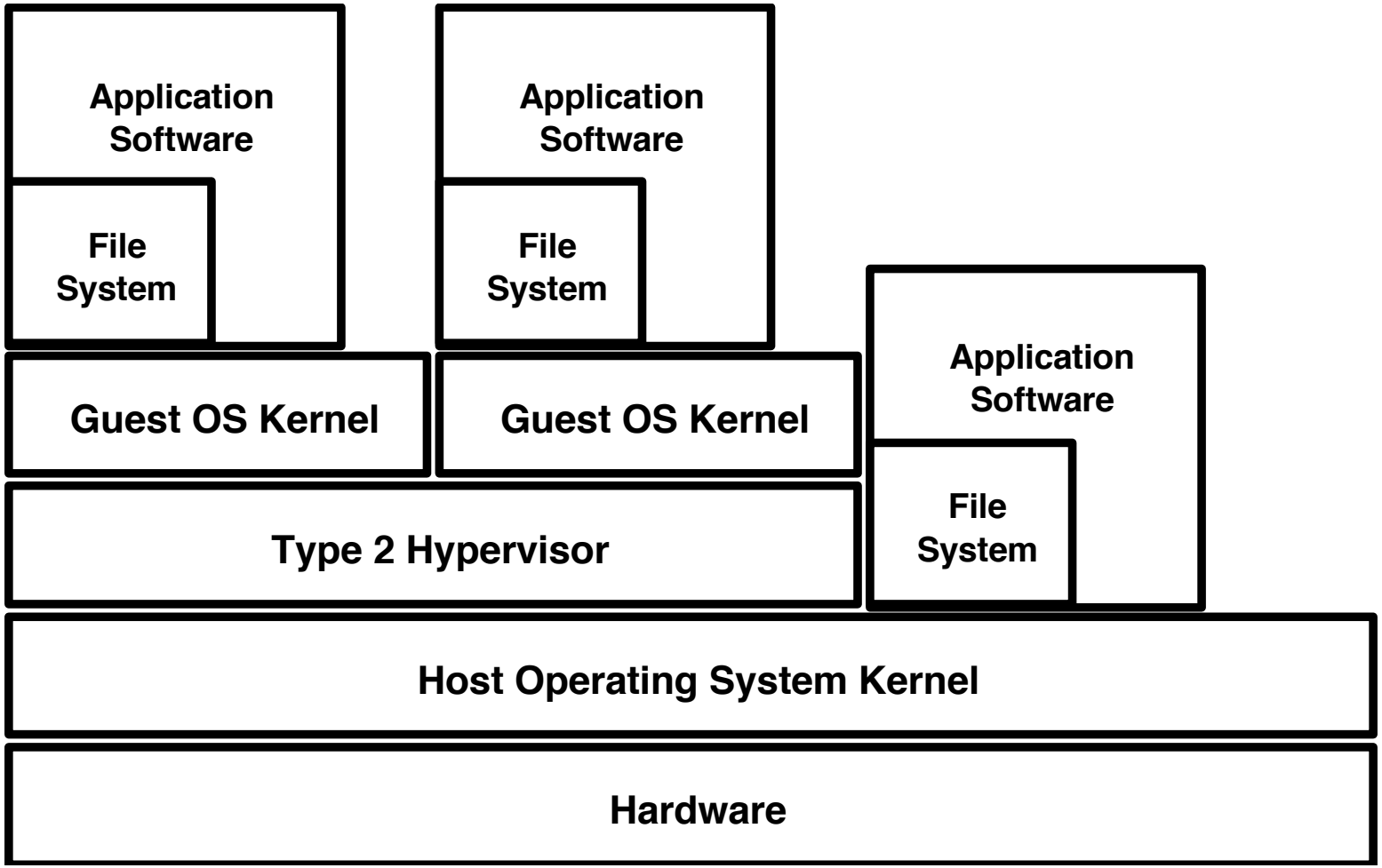
Process Isolation Techniques

Virtual Machines – Type 2

Type 2 Hypervisor runs on top of Host Operating System

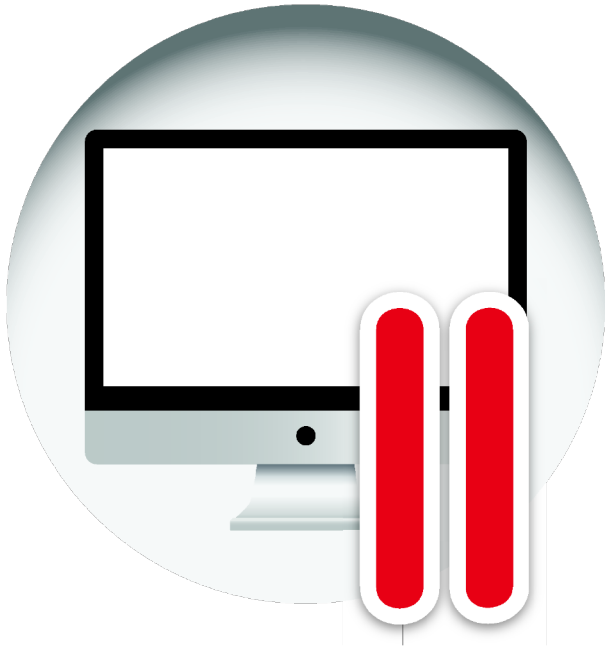
Guest Operating Systems runs on top of Hypervisor

3 layers of software between applications and hardware



Example Type 2 Hypervisors

- Oracle VirtualBox
- Parallels
- VMware
  - Workstation
  - Fusion



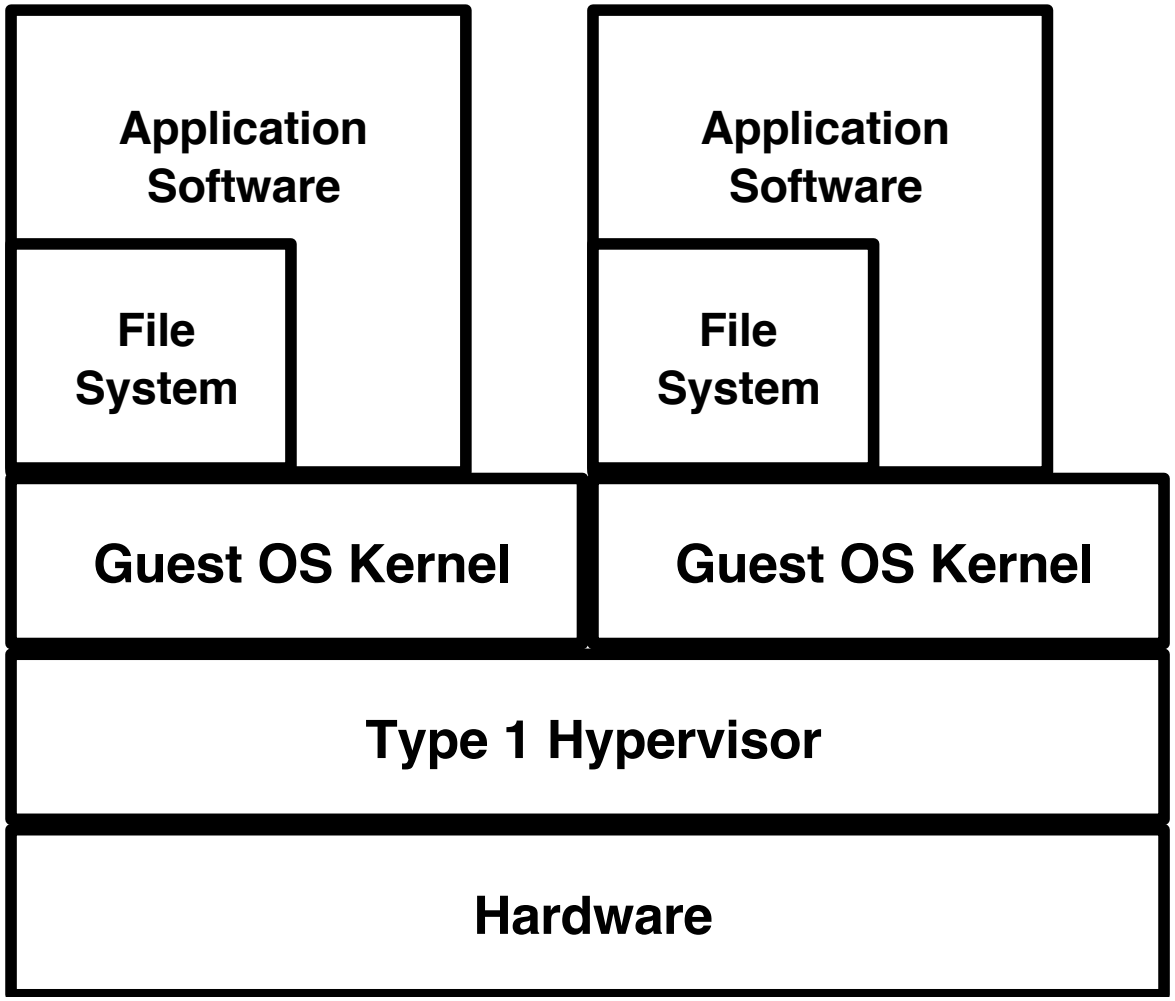
vmware®

Virtual Machines – Type 1

Type 1 Hypervisor runs on directly on top of Hardware

Guest Operating Systems runs on top of Hypervisor

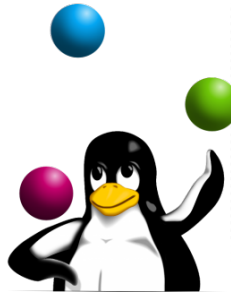
2 layers of software between applications and hardware



Example Type 1 Hypervisors

- Microsoft Hyper-V
- Linux KVM
- VMware
  - ESXi

Windows Server
Hyper-V



KVM

vmware®

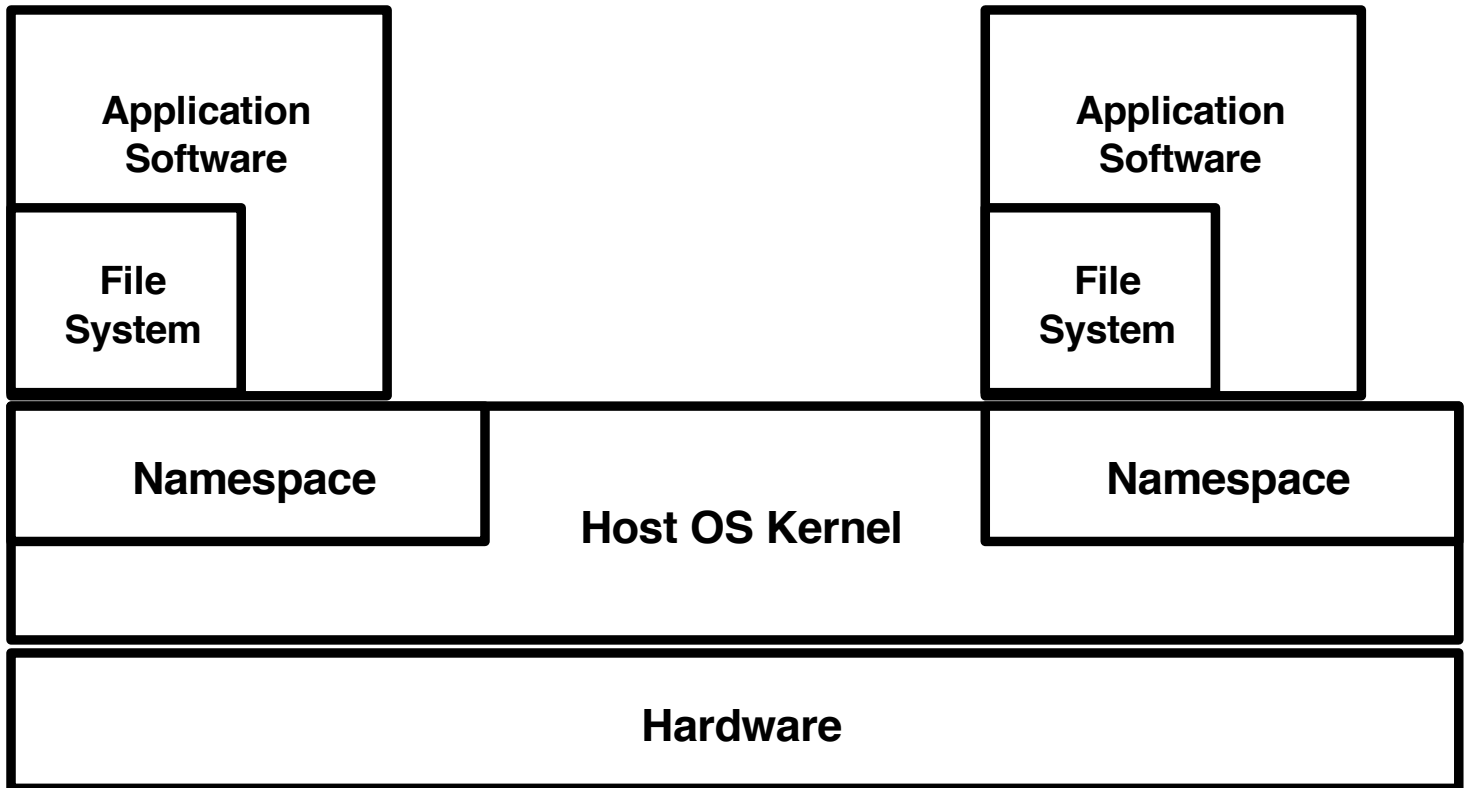
Containerization

OS provides resource namespaces

Processes limited to resources in a namespace

Container started with limited initd process

1 layer of software between applications and hardware

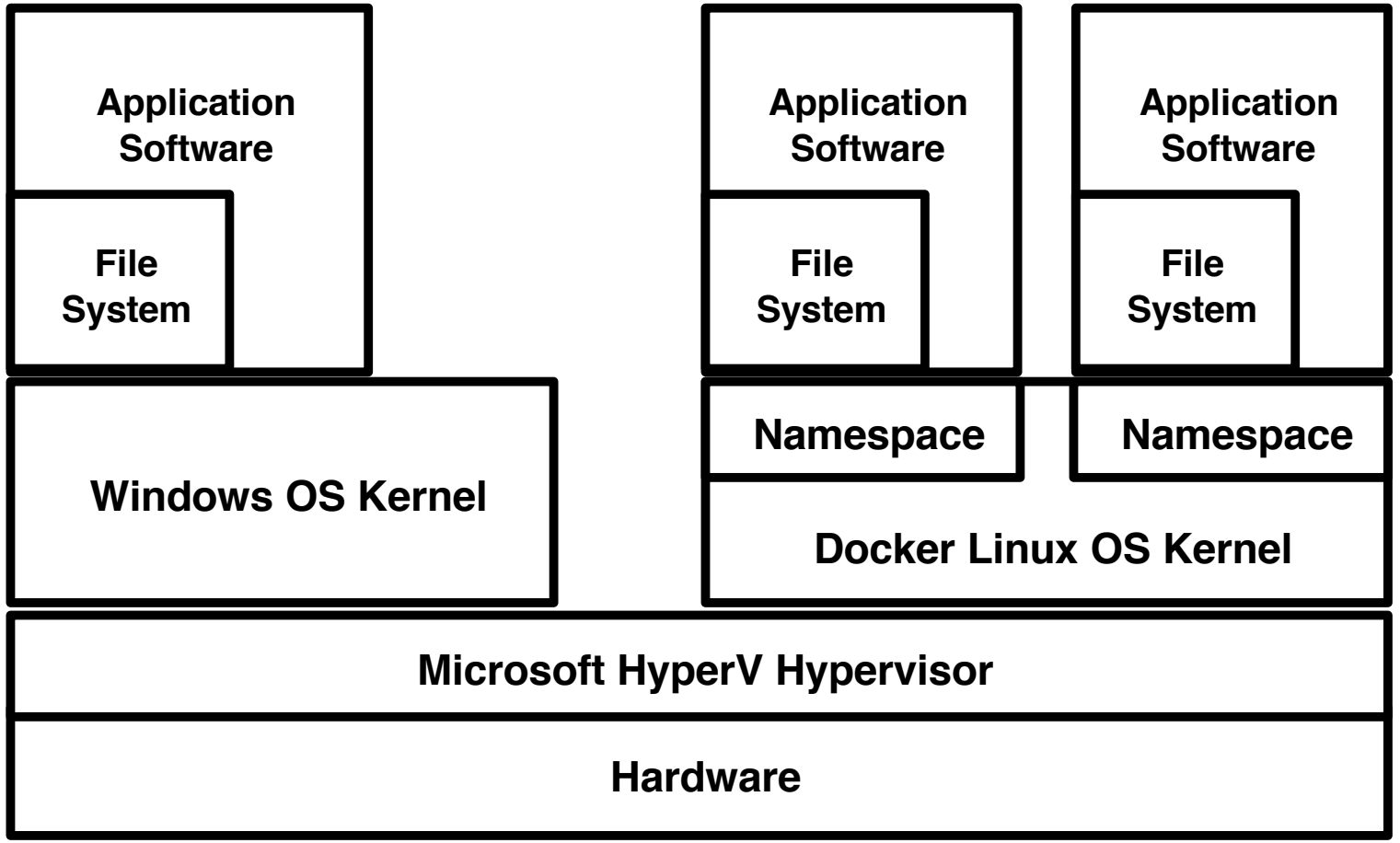


Docker on Windows (or MacOS) with Linux Containers?

Docker supplied Linux OS Kernel runs Docker containers

Windows & Linux at same layer of stack

2 layers of software between applications and hardware



Another Linux / Hyper-V Alternative

- Microsoft provides Linux Distros for Hyper-V, including Ubuntu LTS
- Configured to use XRDP with Hyper-V specific client
- Feels similar to using Linux on Virtualbox

- My desktops & laptops have no problem running Hyper-V with
  - Windows 10 Guest
  - Docker's Linux Guest running an HTTP server (no UI)
  - Microsoft's supplied Ubuntu LTS distro
  - All 3 kernels running at the same time: 16 GB RAM, SSD, 2 i7 cores

Jupyter: Architectural Overview

A Distributed System

with Standards-based Communication Protocols

Twitter: @paulebuis #PyDataIndy

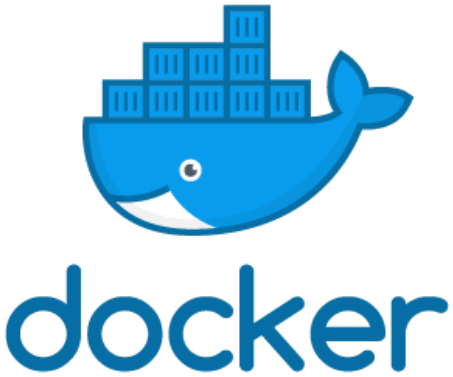
Building Docker Images for Jupyter

Existing GitHub Repositories

Twitter: @paulebuis #PyDataIndy

Building Docker Images: DevOps

- DevOps strives to build computing environments in a reliably reproducible manner using scripting and source code control.
- Used to build platforms on which to deliver commercial software.
- Prevents “It worked on my/our machine, OS & installed software”
- Docker VFS images built from Git repositories with a Dockerfile



git

The Jupyter Project Provides Base Images

Source:

https://github.com/jupyter/docker-stacks

Binary Releases:

https://hub.docker.com/u/jupyter

A Docker *image* is a set of layers forming a VFS

Jupyter Project base image uses:

- Official Docker Ubuntu LTS Image
- MiniConda Python Distro from Anaconda.org

Sample Dockerfile Commands

```
FROM ubuntu:bionic-20190612
USER root
RUN apt-get update && apt-get -yq dist-upgrade && \
    apt-get install -yq --no-install-recommends \
    wget bzip2 ca-certificates sudo locales fonts-liberation
ENV MINICONDA_VERSION=4.6.14 CONDA_VERSION=4.7.10
RUN cd /tmp && \
    wget --quiet https://repo.continuum.io/miniconda/Miniconda3 …
    /bin/bash Miniconda3-${MINICONDA_VERSION}-Linux-x86_64.sh –f  …
…
```

Sample Dockerfile Commands

```
…
EXPOSE 8888
ENTRYPOINT ["tini", "-g", "--"]
CMD ["start-notebook.sh"]
COPY start-notebook.sh /usr/local/bin/
USER 1000
```

Overall Pattern of Juptyer Dockerfiles

- Start with a base image
- Use OS package manager to install packages
- Use `wget` to download other software, then install it
- Use `conda` & `pip` to download and install Python packages
- Use conda downloaded tools

Jupyter/Docker-Stacks Hierarchy

My GitHub Repository

paulbuis/jupyter-many

Builds on

minimal-notebook