

Day 3: Notebook to Cloud

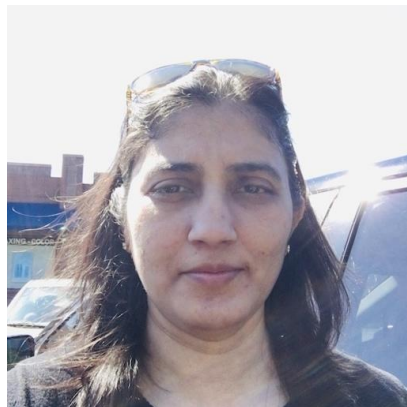
David Sondak



Shivas Jayaram



Mehul Smriti Raje



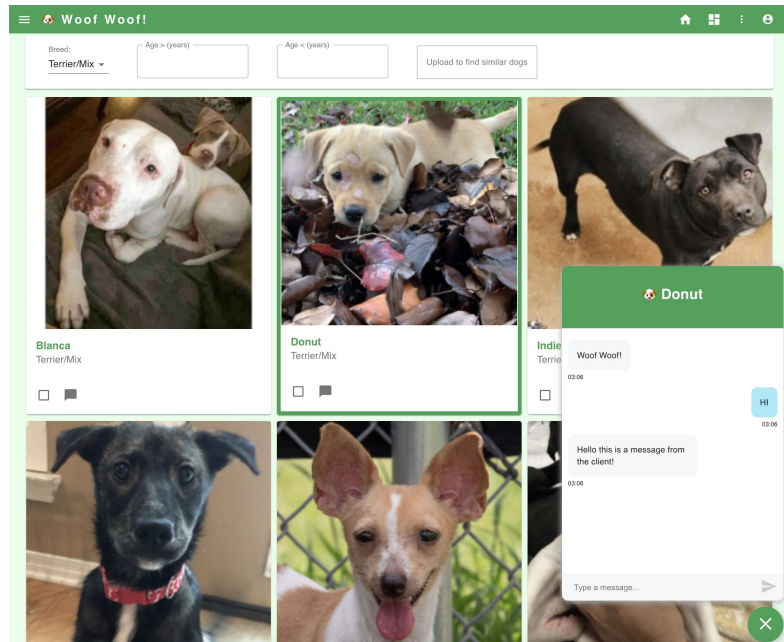
Rashmi Bathia



Connor Capitolo

Don't Forget the Big Picture

- Day 1: You built models for transfer learning
 - Find images of similar dogs
- Day 2: You built language models
 - Chat with a dog
- Today: You will put models into containers
 - Serve the embeddings and language models
 - Expose an API
- Day 4: Make it available on the cloud
 - **Deploy Day!**
 - Release the models
 - Maintain existing models



Workshop Overview for Session 1

Intro Lecture
10:00 - 10:30
David



Code Performance
10:30 - 11:30
Mehul and Connor



Containers
11:30 - 12:30
Connor and Mehul



Build out main application
1:30 - 4:30
Shivas



Lunch
12:30 - 1:30

Why Notebooks?

- Easy to use
- Great for prototyping
- Excellent for documentation and examples

We want to find where the functions $y_1 = x$ and $y_2 = \exp(-2\sin^2(4x))$ intersect. That is, for what values of x is $y_1 = y_2$? This question is equivalent to the problem of finding the zeros of

$$f(x) = x - \exp(-2\sin^2(4x)).$$

Before doing anything, it's a good idea to visualize what we're up against.

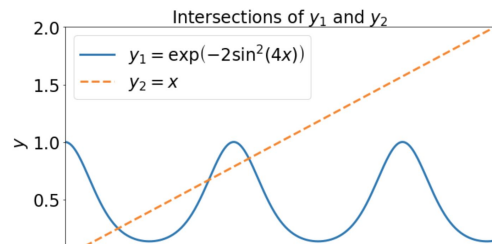
In [2]: Slide Type Sub-Slide ▾

```
# Define function
x = np.linspace(0.0, 2.0*np.pi, 1000)
y = np.exp(-2.0 * np.sin(4.0*x)*np.sin(4.0*x))
```

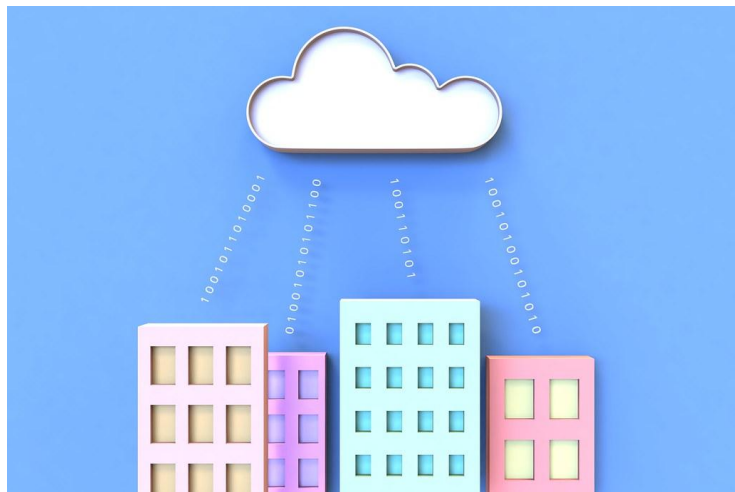
In [3]: Slide Type Sub-Slide ▾

```
# Plot the two functions to see where they intersect
fig, ax = plt.subplots(1,1, figsize=(10,6))
ax.plot(x, y, lw=3, label='y1 = \exp\left(-2\sin^2\right)\left(4x\right)\right')
ax.plot(x, x, ls='--', lw=3, label='y2 = x')

ax.set_xlim(0, 2.0)
ax.set_ylim(0, 2.0)
ax.set_xlabel('x', fontsize=24)
ax.set_ylabel('y', fontsize=24)
ax.set_title('Intersections of y1 and y2', fontsize=24)
ax.tick_params(labelsize=24)
ax.legend(fontsize=24)
plt.tight_layout()
```

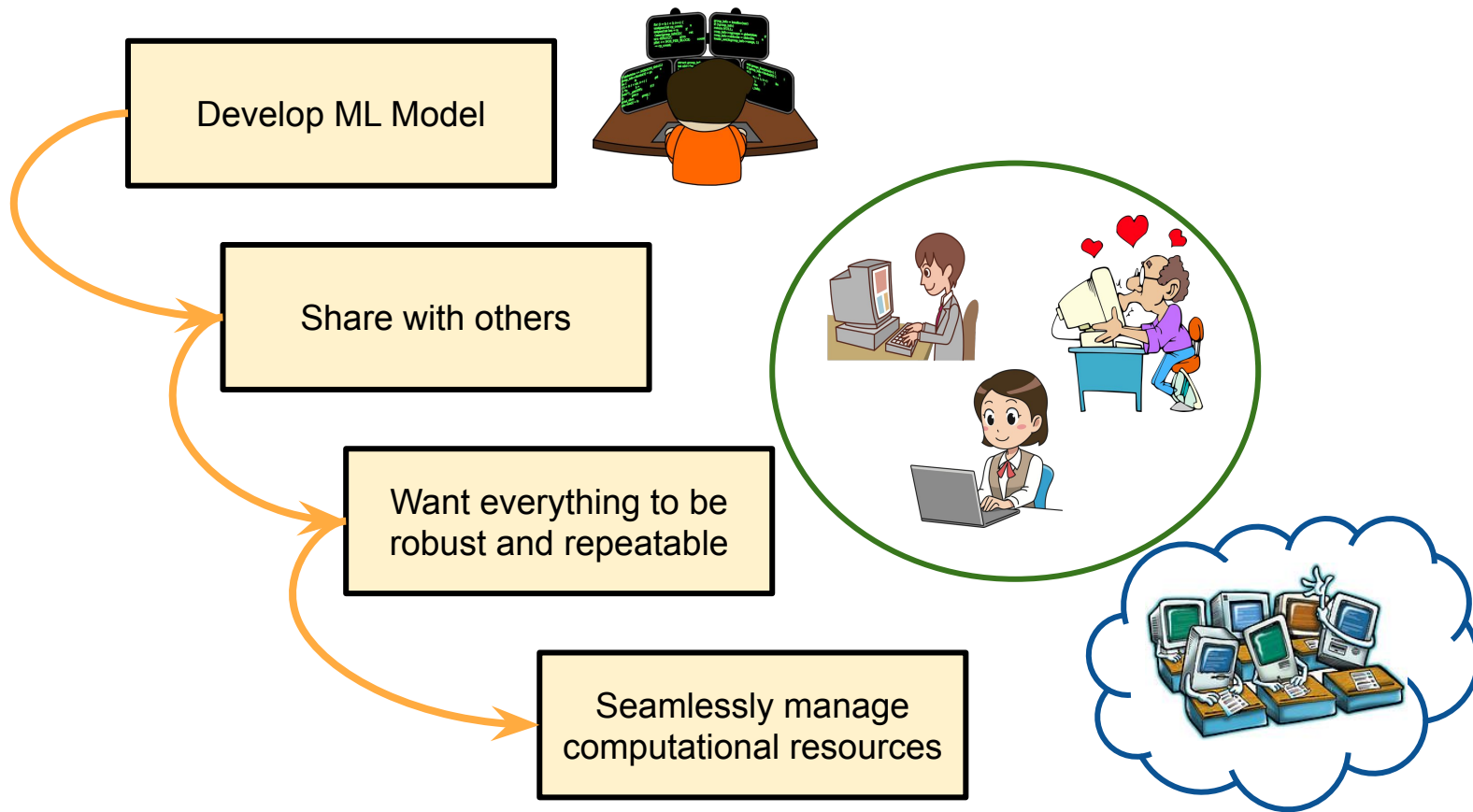


Why Cloud?



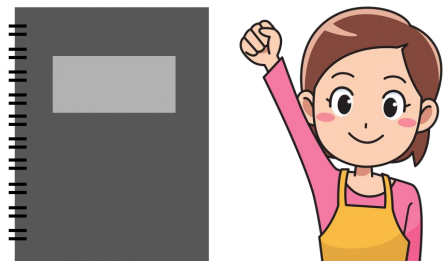
- *Many* resources beyond what you have locally
 - CPUs
 - GPUs
 - Storage
 - ...
- Resources maintained by experts
- Easier to reproduce results
 - Collaborators have replicable platforms
- Easier to host applications
 - Virtual machines
 - Containers
 - ...

A Dream Scenario

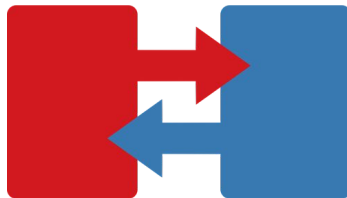


A Common Approach

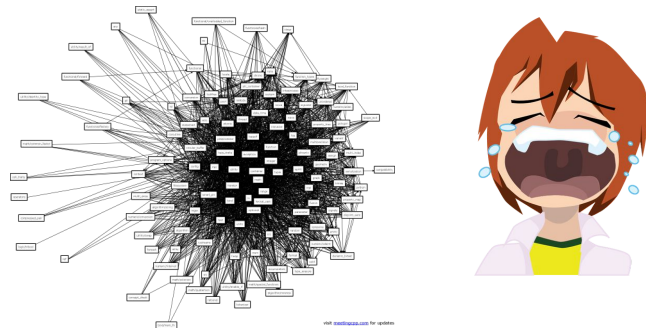
Develop code base in
Jupyter notebooks



Pass around to
colleagues and friends



Tweak depending on versions and
platforms people are using

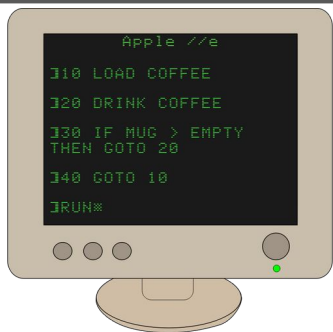


A Modern Approach

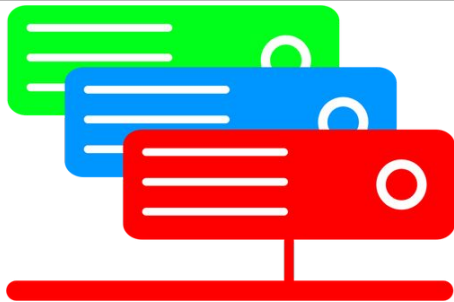


Prototype, document,
and examples in
Jupyter notebooks

1

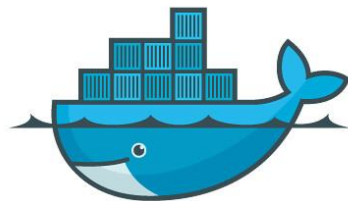


Large code-bases
written in scripts



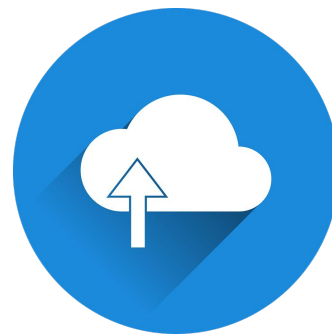
Managed with
version control

2



Ship package in
lightweight containers

3



Deploy to
the cloud

4



Automatically manage
containers in cloud

5

Towards Collaboration: What Can Go Wrong

- Congrats! You've developed a nice application.
- But you can't work in isolation:
 - Need help and input from colleagues
 - Want to have other people use your awesome app
- What can go wrong?
- Turns out, a lot!
 - Different software versions (e.g. Python 2.7 vs. 3.7)
 - Different operating systems (Windows vs. Linux vs. MacOS)
 - Different packages required than are available locally

Achieving Collaborative Isolation

- We develop on specific platforms using specific versions of software and dependencies
- How can we make sure everyone works with the same environment?
 - Virtual environment --- Still depends on operating system
 - Virtual machines --- Full isolation
 - Containers --- Only virtualize OS (not the hardware)

Virtual Environments



- ✓ Requirement A
- ✓ Requirement B
- ✓ Requirement C
- List requirements in special file
- Automatically install dependencies

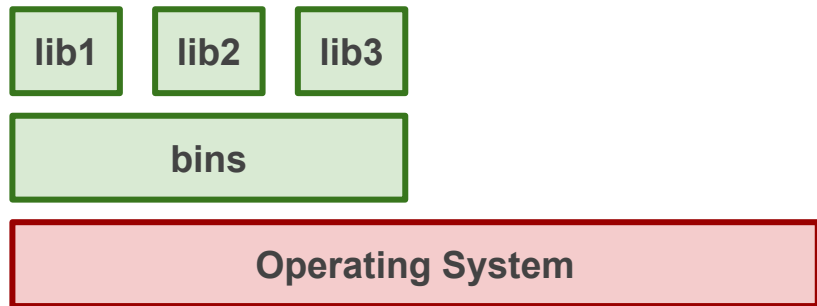


- Create virtual environment
- Install package
- Dependencies installed too
- ✓ Requirement A
- ✓ Requirement B
- ✓ Requirement C

Example Workflow

Sally usually installs modules into `bin`, `lib`, or `include` directories because this is the default behavior

```
$ which python  
/Users/sally/opt/anaconda3/bin/python
```

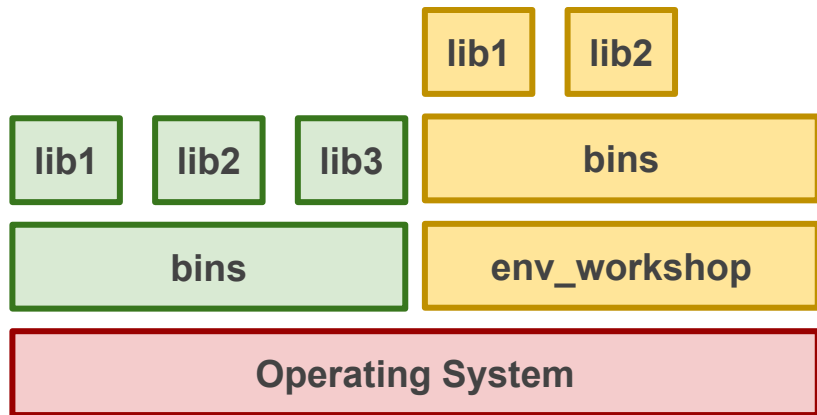


Sally

Example Workflow

Now she decides that she wants to isolate a new environment to avoid conflicts with packages.

```
$ which python  
/Users/sally/opt/anaconda3/envs/env_workshop/python
```



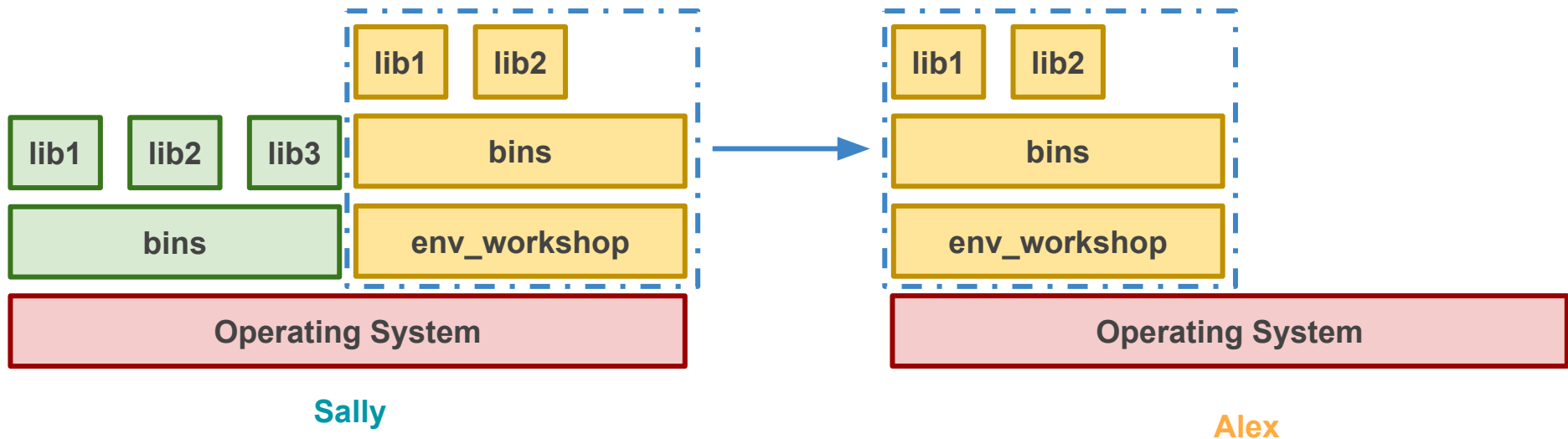
Sally

`env_workshop` is a virtual environment. It helps her organize modules and avoid clashes.

Example Workflow

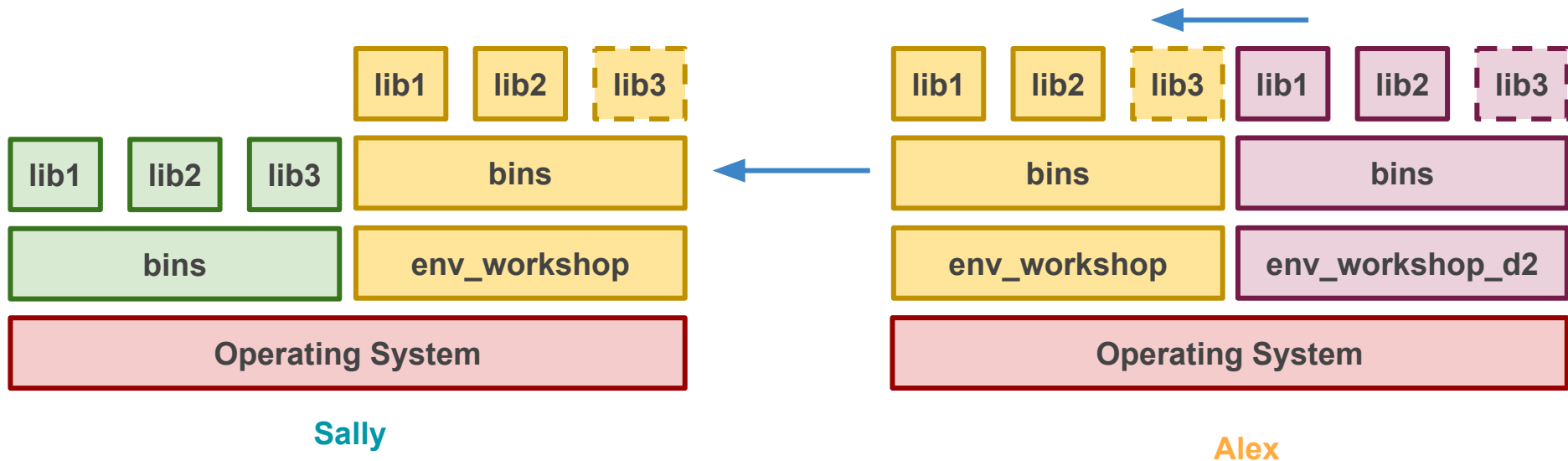
Sally and Alex now decide to collaborate. They may have different packages installed or different package versions. Instead of forcing Alex to update his base installation, Sally shares her virtual environment with him.

This can be done through a `.yaml` file.



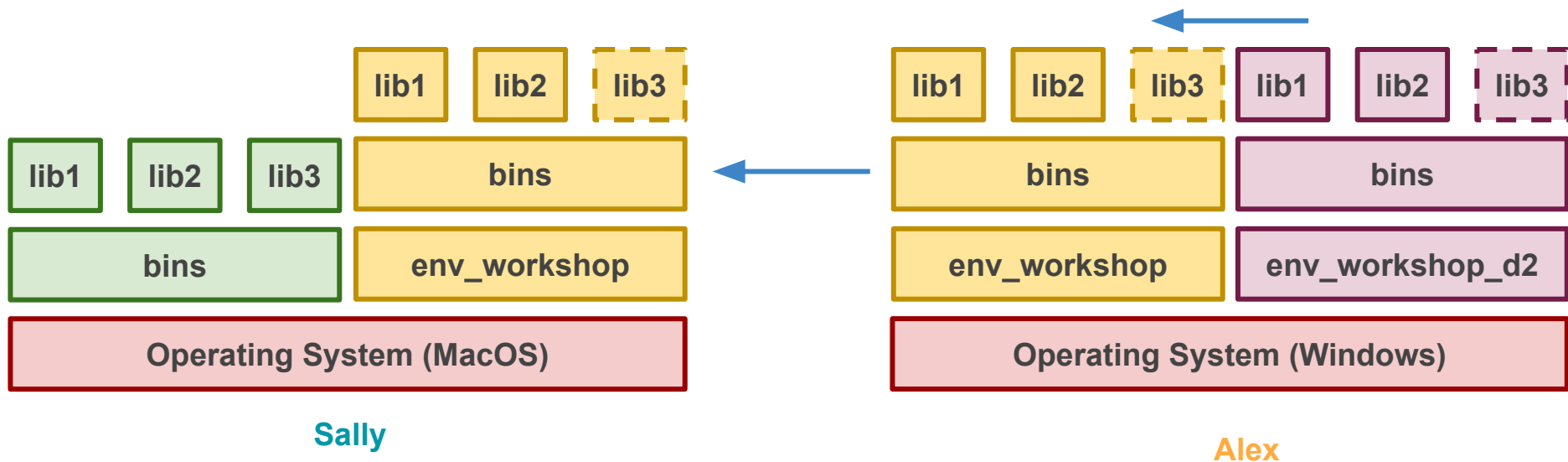
Example Workflow

Now Alex adds a new library based on some really cutting edge ideas that he's been meaning to try out. He updates the .yml file and sends it along with the code that he just developed. Now Sally can update her environment and reproduce Alex's results.



What could go wrong?

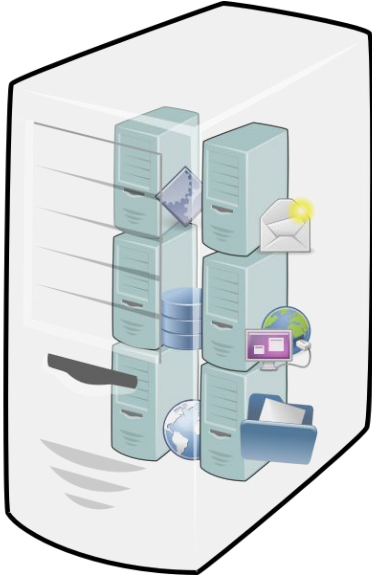
This workflow is very convenient and powerful. However, if Sally and Alex are using different operating systems, then they may still not be able to reproduce each other's results.



Comments on Virtual Environments


- Many options for virtual environments
 - virtualenv
 - conda
- Very useful for working with fellow developers
- Convenient, lightweight method to achieve code portability
- Not as helpful when deploying a package to a larger audience
- They do not provide complete isolation
 - Still depends on your operating system
 - Uses global packages and dependencies of the operating system

The Other End of the Spectrum: Virtual Machines



- A virtual machine (VM) is a file that acts like a separate computer system.
- You can install a completely different operating system on this virtual machine.
 - e.g. You run MacOS; create a VM and install Windows on it
- VMs have their own virtual hardware
 - CPUs, memory, hard drives, etc.
- Software inside the VM can't affect the actual computer
 - Sandboxing
 - Safe place to test virus-infected software

Comments on Virtual Machines

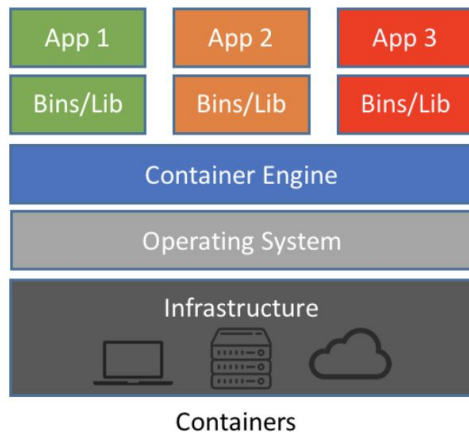
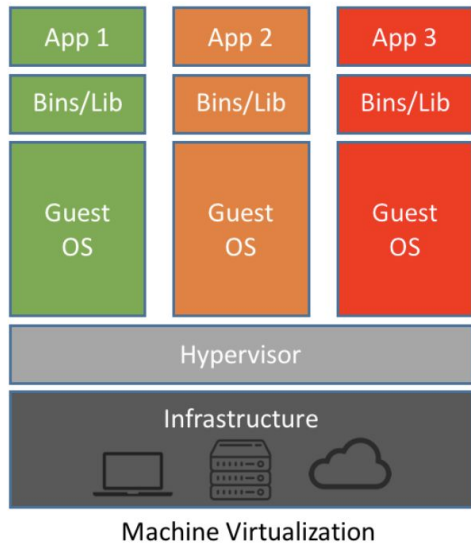
- They can help lower costs and can be more efficient
 - e.g. no need to spend money on physical hardware and cooling systems
 - [Best Virtual Machines of 2021](#)
 - VMware
 - VirtualBox
 - There is overhead associated with VMs
 - They may not be as fast as the host system
 - They may not have the same graphics capabilities
 - They can take some time to start up (order of minutes)
- 
- Powerful but heavyweight

Amazon Web Services
(AWS)

Google Cloud

Microsoft Azure

Containers



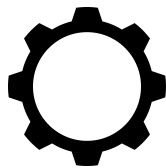
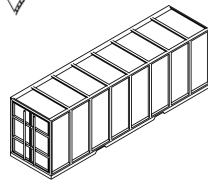
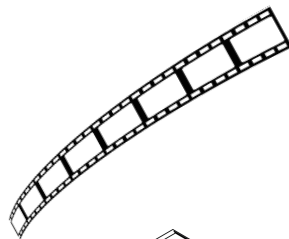
- Only virtualize the OS
- Give impression of separate OS
 - Much cheaper than VMs
- e.g. Create container on Mac, but install Linux OS
 - Container still works on Mac
 - Inside container it's like Linux
- Benefits:
 - Lightweight
 - Quick start-up time
 - Pseudo-isolation
 - Run many at once on a system

Why Containers?

- Fast startup time (seconds vs. minutes for VMs)
- Pseudo-isolation
 - More secure than virtual environments but less secure than virtual machines
- A container is deployed from the container image
 - Offers an isolated executable environment for the application
- Containers can be deployed from a specific image on many platforms
 - Workstations, VMs, public cloud, etc
- Containers are growing in popularity
- Docker is the most popular container project

Docker Concepts

- Docker image
 - The basis of a Docker container
- Docker container
 - The standard unit in which the application service resides and executes
- Docker engine
 - Creates, ships, and runs Docker containers
 - Deployable on a physical or virtual host
 - Deployable locally, in a datacenter, or by a cloud service provider
- Registry service (e.g. Docker Hub)
 - Cloud or server-based storage and distribution service for your images



Images and Containers

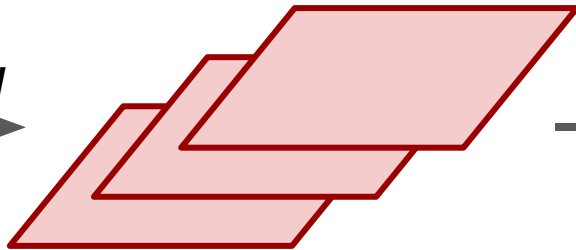
- A Docker image is a template to create a running docker container.
 - Docker uses the information in the image to create / run the container
- Analogies (credit: Pavlos)
 - An image is like a recipe and a container is like a dish
 - An image is like a class and a container is like an instance of that class

```
FROM ubuntu:latest  
RUN apt-get update  
COPY syllabus.md /syllabus.md
```

Create a Dockerfile

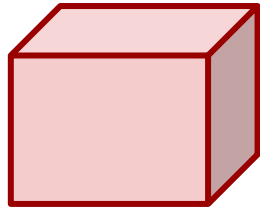
(This is just a text file.)

Build



Docker image

Run



Docker container

The Very Basic Anatomy of a `Dockerfile`

FROM: Tells the daemon which base image to use when creating the new Docker image.

RUN: Instructs the Docker daemon to run the given commands when creating the image.

Note: There can be multiple `RUN` commands; each one creates a new layer.

ENTRYPOINT: Used when the container should run the same executable every time. Usually used to specify the binary.

CMD: Sets the default command and parameters when a Docker container runs.

Note: `CMD` can be overwritten from the command line.

To the Cloud: Managing Containers with Kubernetes



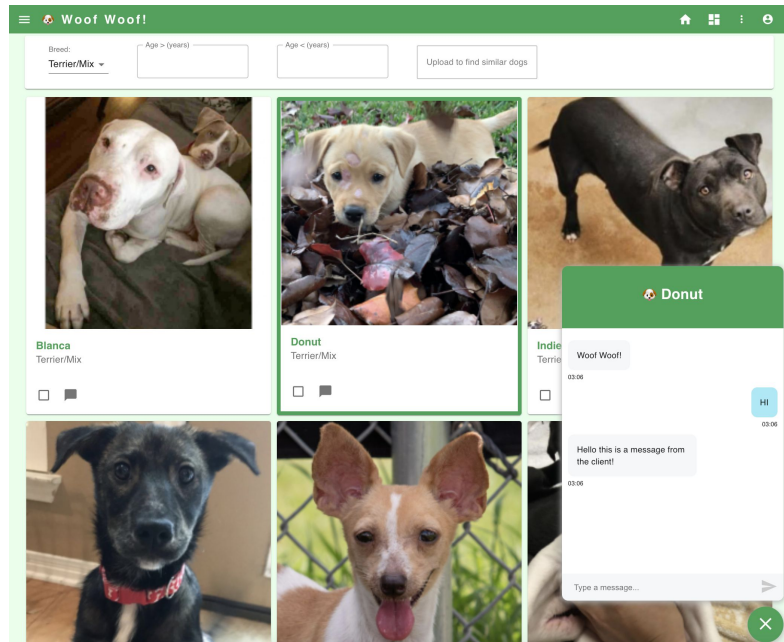
- What if you have a bunch of containers?
 - Either working together or
 - Independent but taking up resources or
 - Both
- K8s does all of the container management for you!
- K8s is an open-source platform for container management developed by Google.
- It allows users to define rules for how container management should occur, and then it handles the rest!

There is so much more...

- We're leaving a lot on the table today
 - Containers for high performance computing: Singularity
 - Amazon Sagemaker : Fully automated machine learning service
 - Much, much more
- Goals for today:
 - Perspective on efficient development practices
 - Best practices for deploying models - containerization

Don't Forget the Big Picture

- Day 1: You built models for transfer learning
 - Find images of similar dogs
- Day 2: You built language models
 - Chat with a dog
- Today: You will put models into containers
 - Serve the embeddings and language models
 - Expose an API
- Day 4: Make it available on the cloud
 - Deploy Day!
 - Release the models
 - Maintain existing models



Workshop Overview for Session 1

Intro Lecture
10:00 - 10:30
David



Code Performance
10:30 - 11:30
Mehul and Connor



Containers
11:30 - 12:30
Connor and Mehul



Lunch
12:30 - 1:30



Build out main application
1:30 - 4:30
Shivas and Rashmi