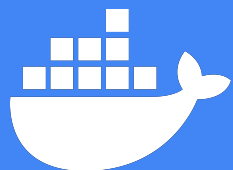


docker
ridgerun 2020



developers love docker



Most **Wanted**
Platform



Linux: 83.1%
Docker: 77.8%
Kubernetes: 76.8%

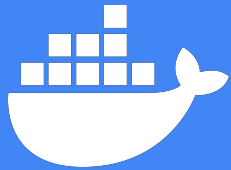
Most **Loved**
Platform







































Docker: 31.5%
Kubernetes: 8.5%

Most **Used**
Platform



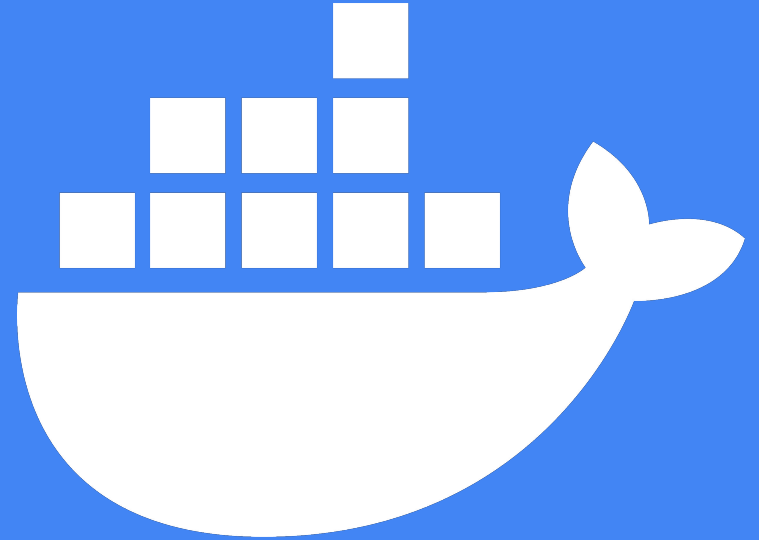


companies love docker

 Oil & Gas / Energy	 Healthcare & Science	 Financial Services	 Tech & Manufacturing	 Insurance	 Public Sector
					
					
					
					
					

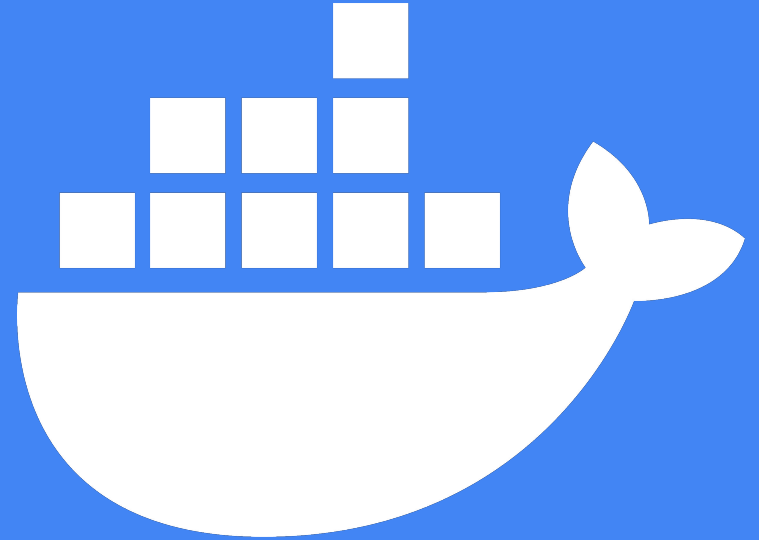
SO...

...what is docker useful for?



you...

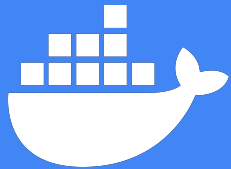
...tell me at the end of the slides.





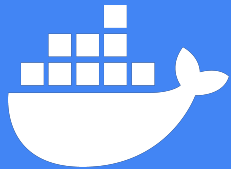


- A technology that enables the management and use of containers.
- A set of tools to architect applications and services.
- An ecosystem to share and collaborate with other developers.



containers are

- A technology that allows you to package and isolate applications with their entire runtime.
- Not virtual machines.
- Built on top of the host operating system and kernel.



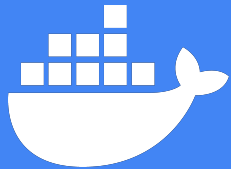
comparison

Containers

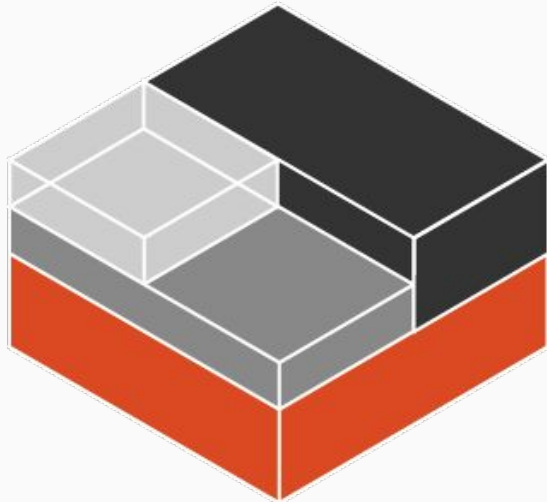
- Less overhead
- Share host OS kernel
- Virtualize using software runtimes
- Only the FS experience

Virtual Machines

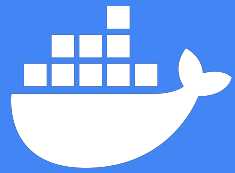
- More overhead
- Provide their own kernel
- May use hardware to virtualize
- Full machine experience



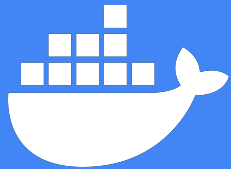
containers already existed



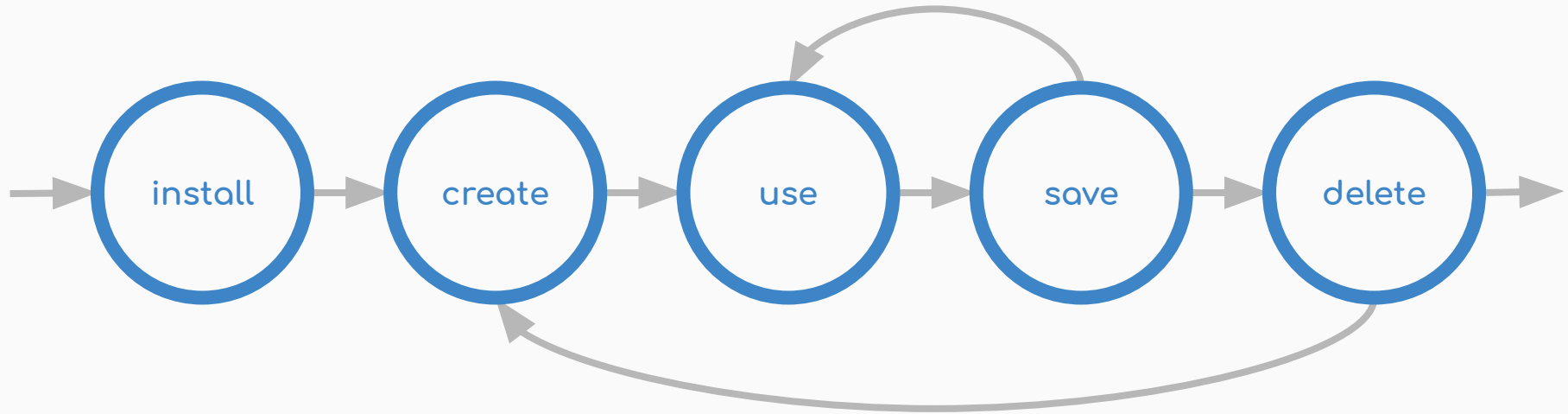
- LXC (Linux Containers) is the most obvious example.
- LXC provide a more OS like experience
- Docker is single-app by default (i.e.: no proper init).
- Docker is much more user friendly.

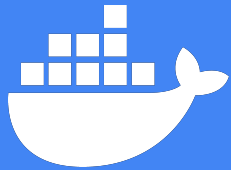


docker lifecycle

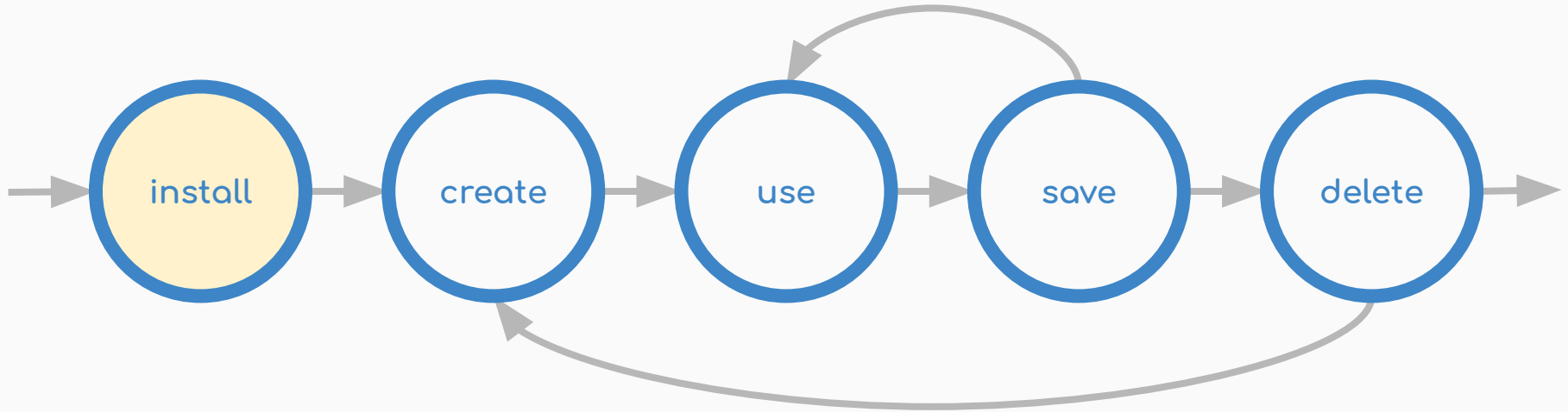


docker lifecycle





docker lifecycle





Full instructions:

<https://docs.docker.com/install/linux/docker-ce/ubuntu/>

Other architectures available!

```
docker

# Uninstall old versions
sudo apt remove docker docker-engine docker.io containerd runc

# Install dependencies
sudo apt update
sudo apt install apt-transport-https ca-certificates curl gnupg-agent
software-properties-common

# Install GPG Key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -

# Add Docker PPA
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

# Finally install
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io
```



Running as non-root user.

Full instructions at:

<https://docs.docker.com/install/linux/linux-postinstall/>

Under your own risk!

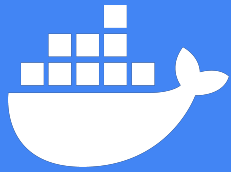
```
docker

# Create the docker group
sudo groupadd docker

# Add yourself to the group
sudo usermod -aG docker $USER

# To avoid re-logging
newgrp docker

# Test proper installation
docker ps
```



install

Other operating systems.

[Docker for Mac](#)

[Docker for Windows](#)



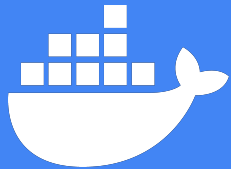
Some useful tips.

```
docker

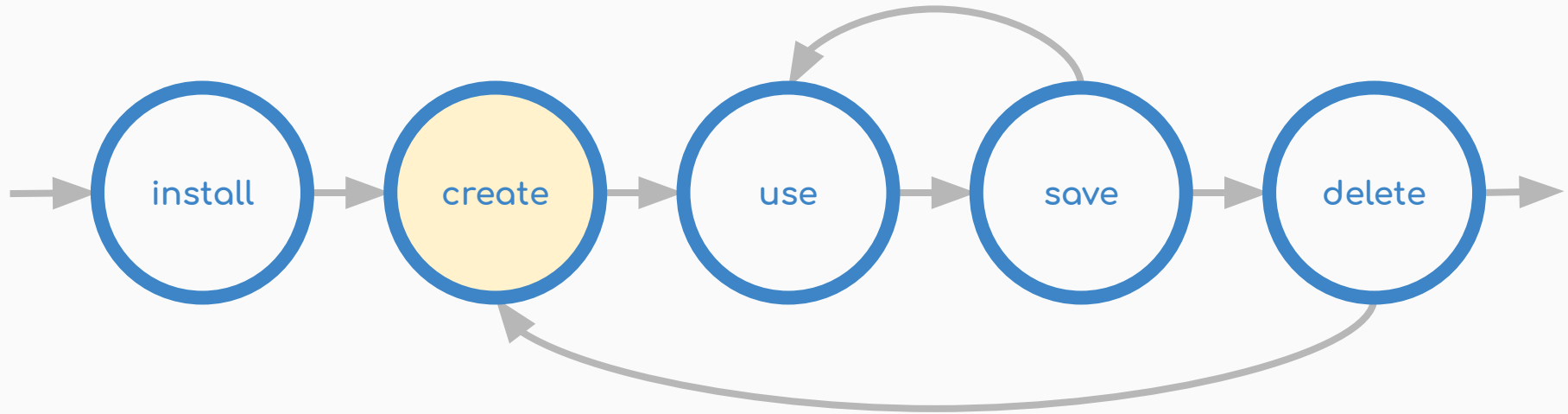
# List commands
docker help

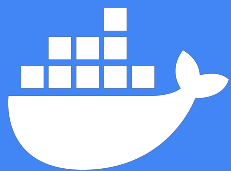
# Specific command help
docker create --help

# Print version
docker version
```



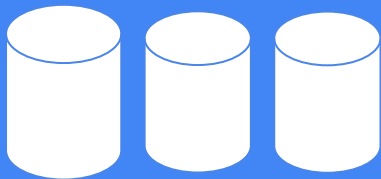
docker lifecycle





create

Containers are created from



images



registry



PULLS

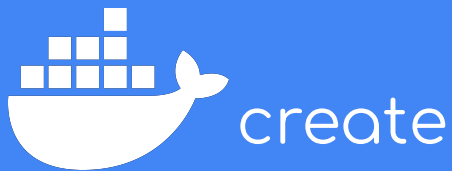
computer



CREATES



container



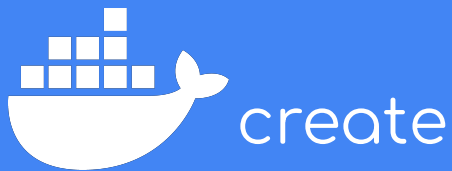
You refer to images by their name.

Image names convey a lot of information.

We use defaults most of the times.

dockerhub.cr.ridgerun.com:5000/ridgerun/ubuntu:16.04

Portion	Description
1	Registry hostname to pull image from. If omitted defaults to the official Docker hub.
2	Account hosting the image. Not needed for official docker images.
3	Image name. The only mandatory parameter.
4	Tag or version. If omitted, the "latest" tag will be used. This is not necessary the most recent!



Create a new container!

You need a base image

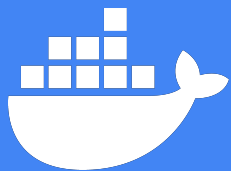
List of public images:

[https://hub.docker.com/search/
?type=image](https://hub.docker.com/search/?type=image)

When do you omit -t and -i?

```
docker
# Create the container
docker create -ti --name mycontainer ubuntu:18.04
```

Parameter	Description
-i	Read STDIN (input)
-t	Attach a pseudo-terminal (interaction)
--name mycontainer	Custom name of the container
ubuntu	Image
18.04	Tag name (version). Defaults to "latest"



create

Share data between host and container

Absolute paths share dirs or files

Relative create new volumes

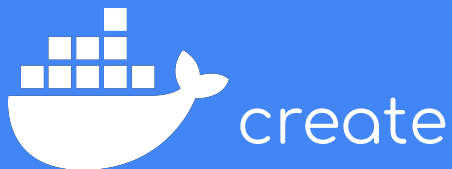
Useful for devices, for example.

```
docker

# Share dir: /home/mgruner (host) in /host (container)
docker create -ti -v /home/mgruner/:/host ubuntu:18.04

# Share file: /home/mgruner/file.txt (host) in /host.txt (container)
docker create -ti -v /home/mgruner/file.txt:/host.txt ubuntu:18.04

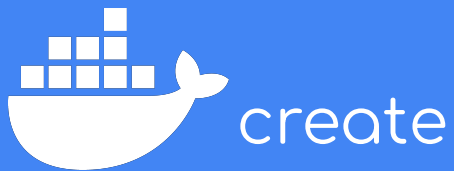
# Create a persistent volume
docker create -ti -v myvolume:/host ubuntu:18.04
```



Other useful options

Make sure you list all options
using *docker create --help*

Parameter	Description
<code>--memory , -m</code>	Memory limit (bytes), i.e: 1024, 1M, 1G
<code>--cpus</code>	CPU percentage to use, i.e: 1, 1.5
<code>--restart</code>	Auto-restart: no, on-failure, always, unless-stopped
<code>--rm</code>	Remove container after stop
<code>--env, -e</code>	Define environment variables, i.e: A=a



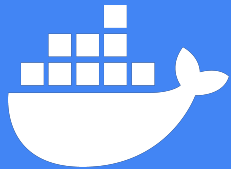
Some useful tips

```
docker

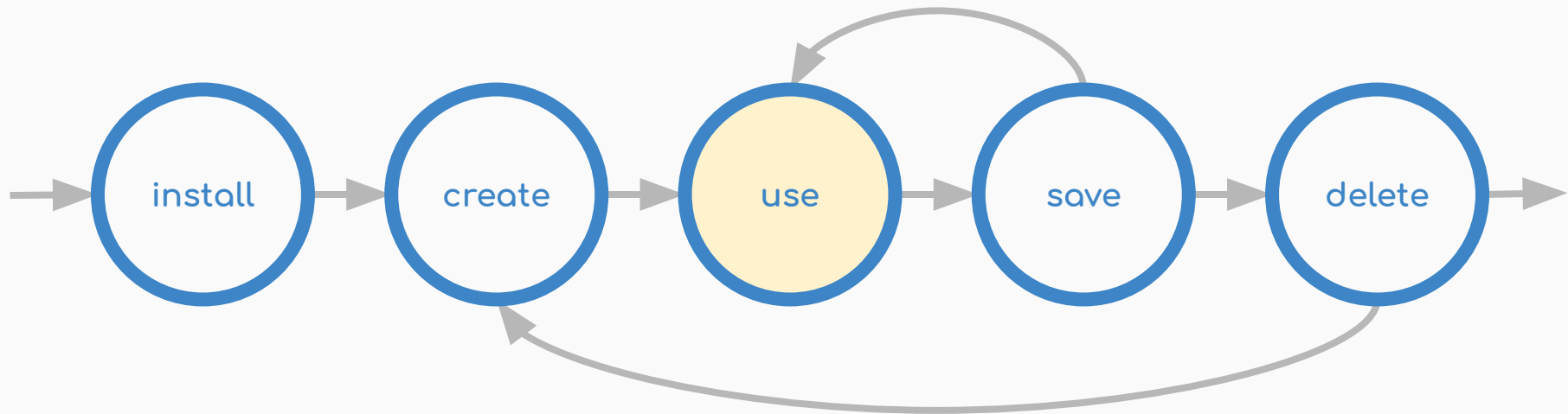
# List existing containers (started)
docker ps

# List existing containers (all)
docker ps -a

# Get container info (IP address, for example)
docker inspect mycontainer
```

docker lifecycle





Time to execute and use the container.

Containers need to be started.

Bash is a usual command for interaction.

You may use any command.

Stop your container afterwards.

Nothing will be erased.

```
docker

# Start the container
docker start mycontainer

# Get into the container
docker exec -ti mycontainer bash

# Stop the container
docker stop mycontainer
```



Usual initial needed work

Note that you are root!

You have absolutely nothing in your container.

```
docker

# Get into the container
docker start mycontainer
docker exec -ti mycontainer bash
```

```
container

# Load up the apt caches
apt update

# Install common packages
apt install git build-essential autoconf libtool autotools
```



SSH is a very common use case

SSH needs to be manually started

I recommend creating a user

Practice connecting to the container!

How can you get the IP?

```
container

# Install sshd and sudo for your new user
apt install sudo openssh-server

# Create new user
adduser ridgerun

# Give it root access
adduser ridgerun sudo

# Manually start ssh server (every time container is started)
service ssh start

# Exit the container
exit
```



What about long running processes?

You may detach the container.

Processes will continue to run.

Reattach later!

```
docker

# Option 1: detach from execution
docker exec -d -ti mycontainer /run_long_process.sh
```

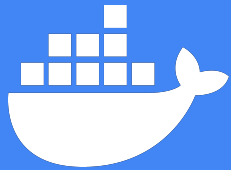
```
container

# Option 2: detach from container
./run_long_process.sh

# Now press ctrl+p ctrl+q to detach
```

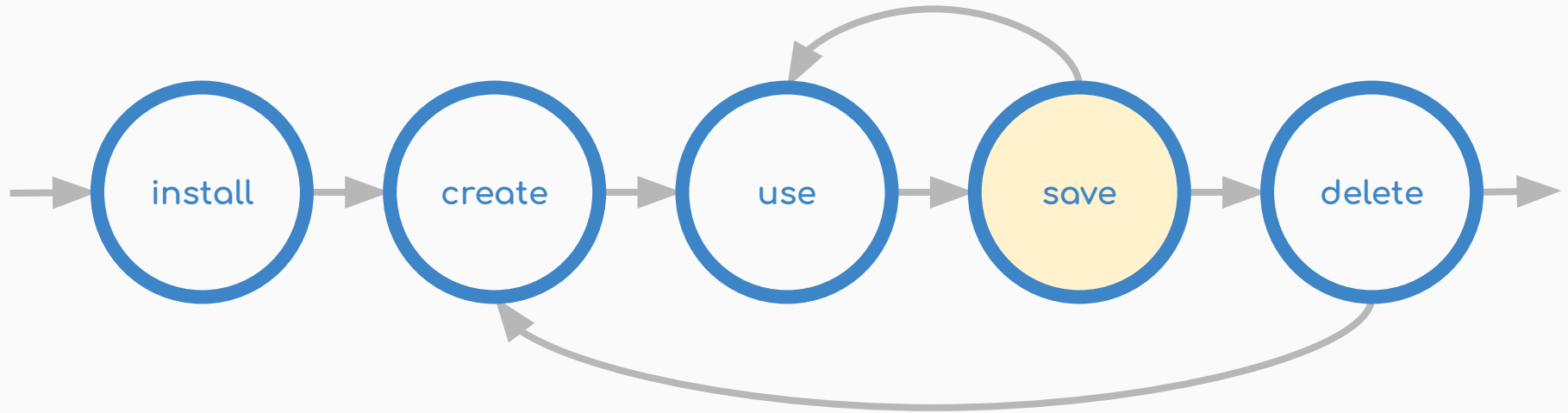
```
docker

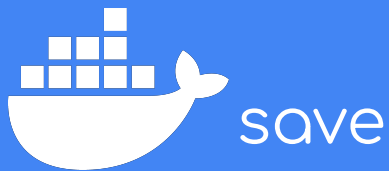
# Reattach to the container
docker attach mycontainer
```



docker lifecycle

30





You just spent hours in a perfect container!

You can save the current state into an image.

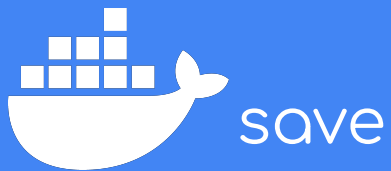
You can create new containers from this image.

The image is a snapshot of the container.

```
docker

# Save an image based on a container
docker commit \
  -a "Michael Gruner < michael.gruner@ridgerun.com >" \
  -m "Example container" mycontainer \
  myimage:mytag

# Create a new container from you newly created image!
docker create -ti --name newcontainer myimage:mytag
```



Tags serve as an alias to an image.

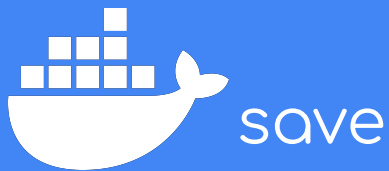
You can rename images using aliases.

```
docker

# Save an image based on a container
docker tag myimage:mytag newimage:newtag

# List existing images
docker images

# Alternative way to list images
docker image ls
```

Here's how you interact with images.

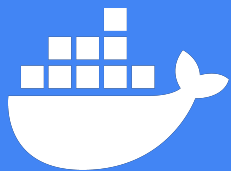
```
docker

# Remove an existing image
docker image rm newimage:newtag

# Remove all unused images
docker image prune

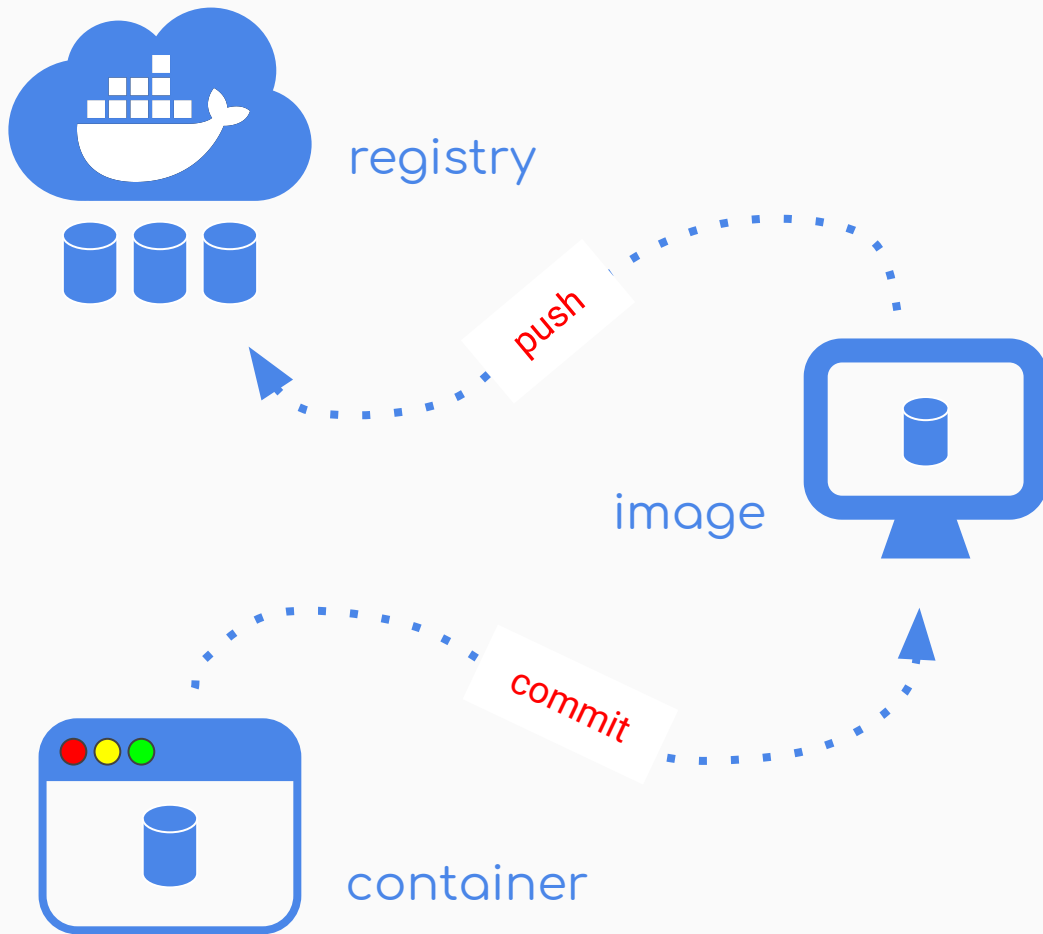
# Save an image to share it
docker save myimage:mytag | gzip > myimage_mytag.tar.gz

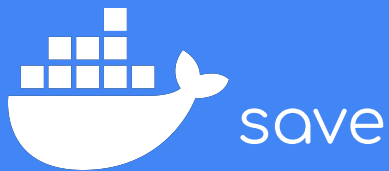
# Load a shared image
docker load < myimage_mytag.tar.gz
```



save

Images may be pushed to registries.





Push an image to a docker registry.

Use tag to point to the appropriate registry.

Remember that no hostname default to DockerHub.

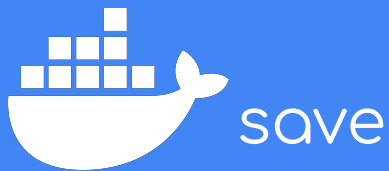
```
docker

# Point to custom registry
docker tag newimage:newtag
dockerhub.cr.ridgerun.com:5000/ridgerun/myimage:mytag

# Push image
docker push
dockerhub.cr.ridgerun.com:5000/ridgerun/myimage:mytag

# Point do ridgerun repository at Docker Hub
docker tag myimage:mytag ridgerun/myimage:mytag

# Push image
docker push ridgerun/myimage:mytag
```



Dockerfiles are an alternative way to create an image.

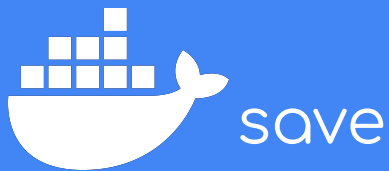
Dockerfile:

```
FROM ubuntu:18.04
RUN apt-get update && \
    apt-get install -y openssh-server sudo && \
    useradd -p $(openssl passwd -1 mgruner) mgruner && \
    adduser mgruner sudo

ENTRYPOINT service ssh restart && bash
```

```
# Build an image from a dockerfile
docker build -t imagename:imagetag .

# List images
docker images
```



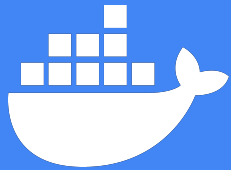
Dockerfile vs push?

The golden image is a mythical creature.

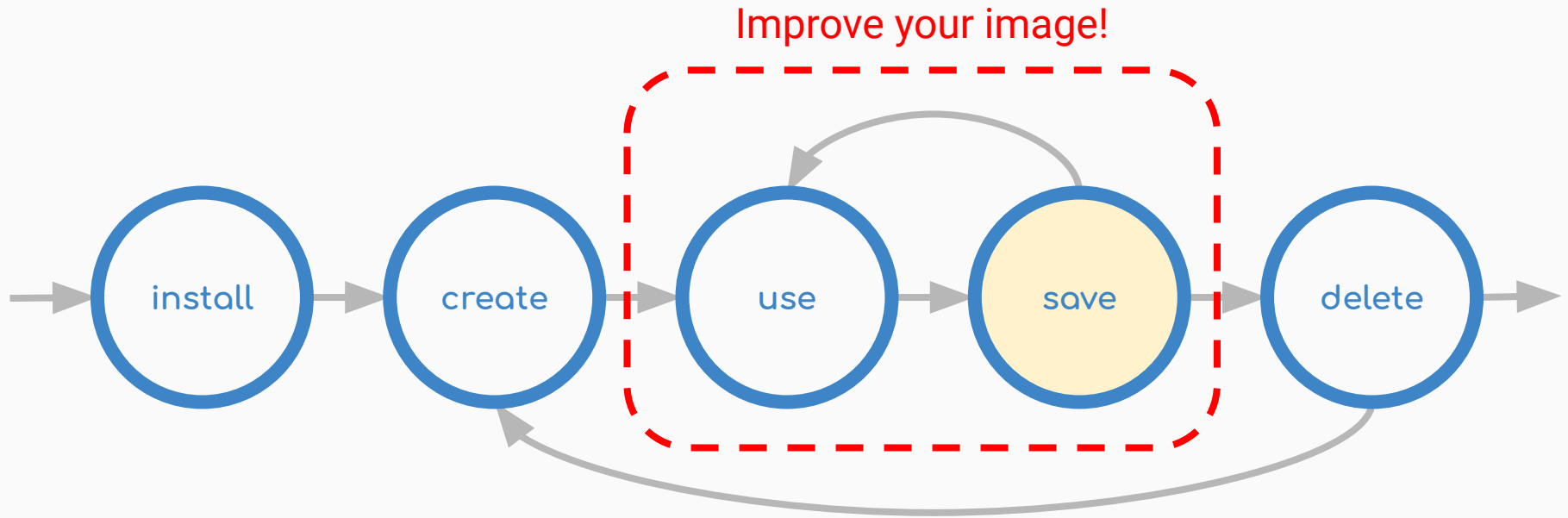
Nobody knows how the golden image was built.

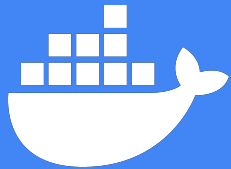
Avoid the golden image.

ツ (ツ) ツ
THE GOLDEN
IMAGE

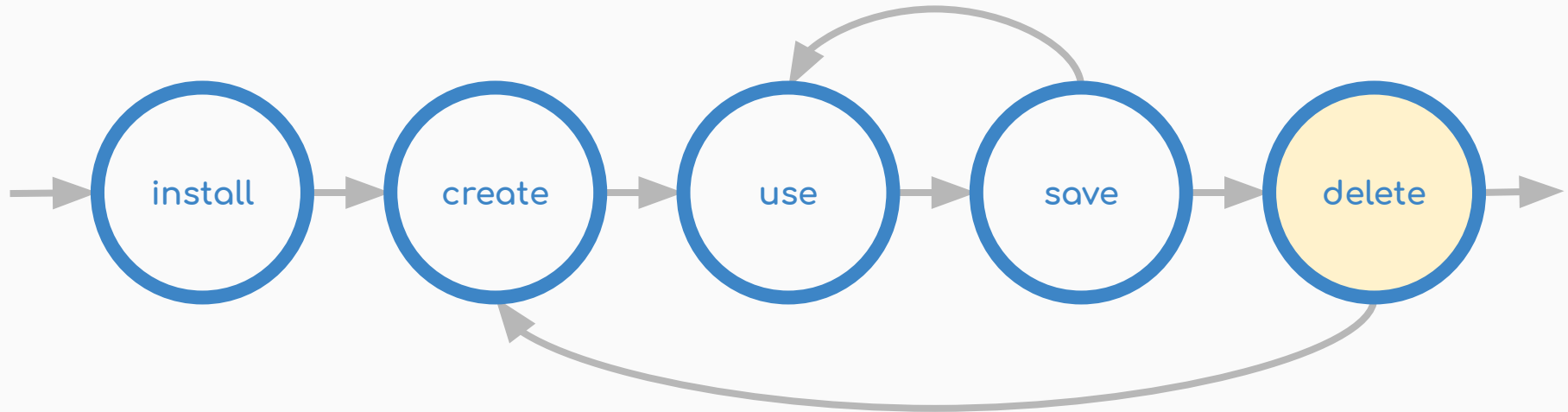


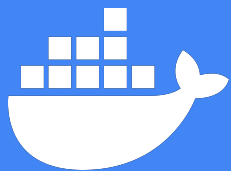
docker lifecycle





docker lifecycle

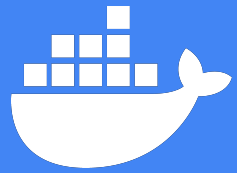




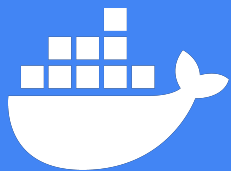
delete

Finally delete the containers you no longer need.

```
docker  
  
# Stop the container first  
docker stop mycontainer  
  
# Remove the container  
docker rm mycontainer
```

docker pro tips



portainer

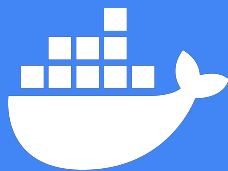
Manage containers through a web server.

Can you figure out the syntax?

```
docker

# Create a persistent memory volume
docker volume create portainer_data

# Run portainer forever!
docker run -d -p 8000:8000 -p 9000:9000 --name=portainer
--restart=always -v /var/run/docker.sock:/var/run/docker.sock -v
portainer_data:/data portainer/portainer
```



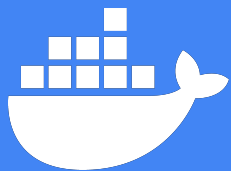
portainer

Use your browser to access portainer.

<http://localhost:9000>

The screenshot displays the Portainer web interface. On the left is a dark sidebar with navigation links: Home, SETTINGS, Users, Endpoints, Registries, and settings. The main content area is titled 'Home' and 'Endpoints'. It features a search bar and a list of endpoints. Each endpoint card shows a whale icon, a name, a status indicator (green), a timestamp, and a summary of resources: stacks, services, containers (with a heart icon), volumes, and images. The endpoints listed are:

- Local Swarm Cluster** (2018-10-21 16:23:04): 2 stacks, 5 services, 9 containers, 21 volumes, 33 images. Group: Unassigned. Swarm 18.03.0-aa - 5 Agent.
- Cloud Swarm Cluster** (2018-10-21 16:23:04): 1 stacks, 1 services, 17 containers, 14 volumes, 30 images. Group: Unassigned. Swarm 17.06.0-ca - 5 Agent.
- Windows1607 Swarm** (2018-10-21 16:23:04): 0 stacks, 0 services, 1 containers, 1 volumes, 1 images. Group: Unassigned. Swarm 18.03.1-aa-3.
- Standalone Node** (2018-10-21 16:23:04): 1 stacks, 5 containers, 4 volumes, 9 images. Group: Unassigned. Standalone 1.1.1 - 5 Agent.
- Storage Cluster** (2018-10-21 16:23:04): 0 stacks, 0 services, 1 containers, 0 volumes, 4 images. Group: Unassigned. Swarm 17.12.1-ca - 5 Agent.
- Windows1803 Swarm** (2018-10-21 16:23:04): 0 stacks, 0 services, 1 containers, 1 volumes, 2 images. Group: Unassigned. Swarm 18.03.1-aa-3 - 5 Agent.



user iFace

Having the graphical portion of the OS is very useful.

We've found that X2Go is the easiest way.

XFCE is a lightweight desktop environment.

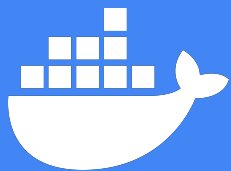
```
docker
# Install x2go client
sudo apt install x2goclient
```

```
container
# Install x2go server, desktop environment and sudo
apt install x2goserver xfce4 sudo

# Create a user
adduser ridgerun

# Give the new user sudo access
adduser ridgerun sudo

# Start ssh server
service ssh start
```

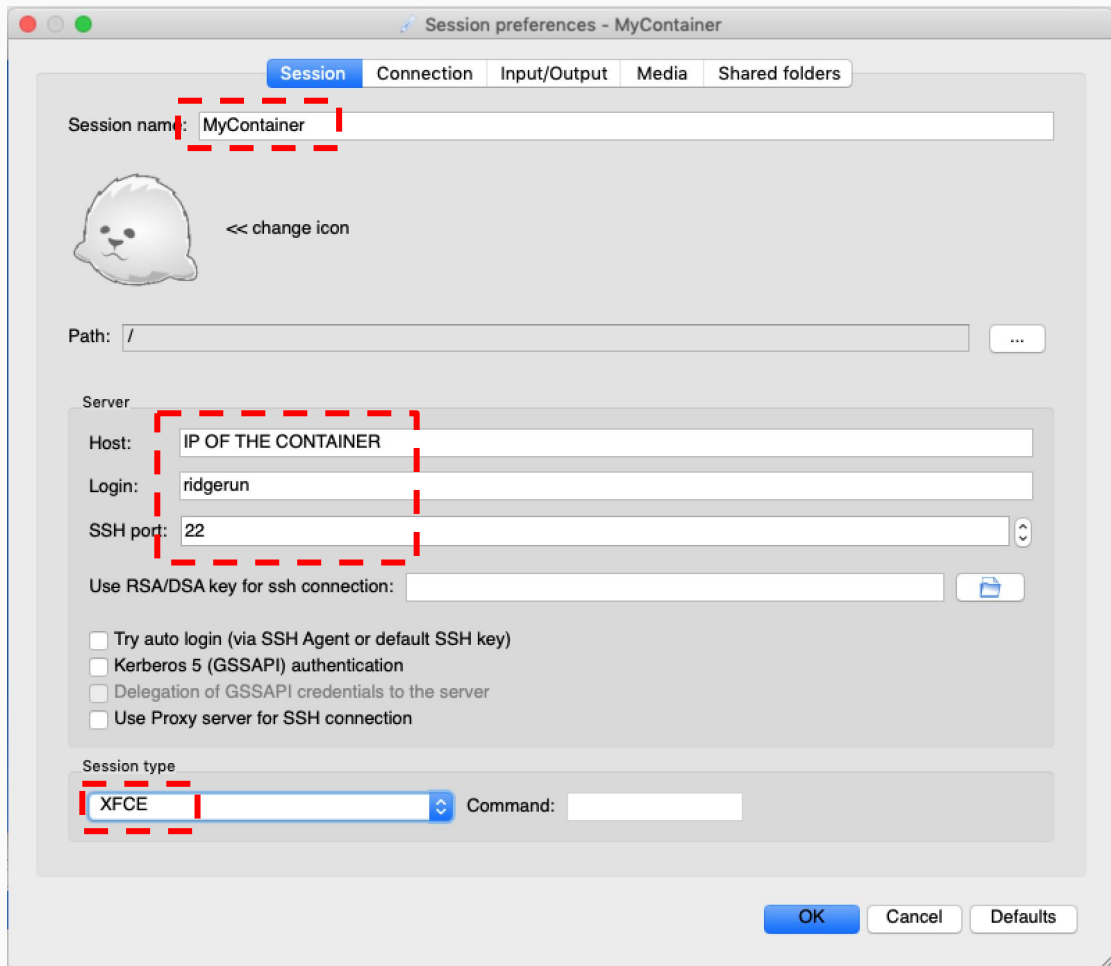


user iface

Launch X2Go Client.

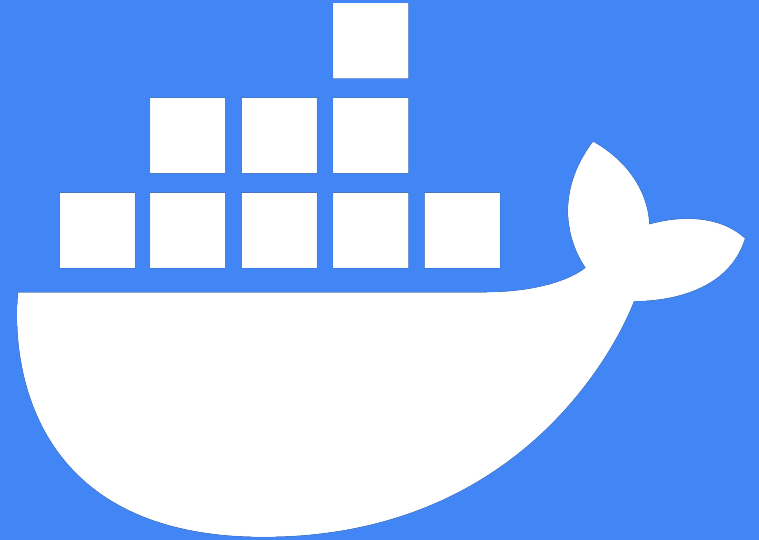
Configure a session as the following.

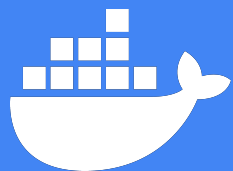
Sessions are persistent!



SO...

...what is docker useful for?

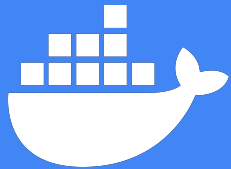




reproducible environments

47

(ツ)/\
IT WORKS
ON MY MACHINE



continuous integration

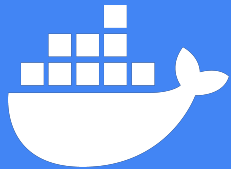
The TensorFlow project strives to abide by generally accepted best practices in open

ci best practices passing Contributor Covenant v1.4 adopted

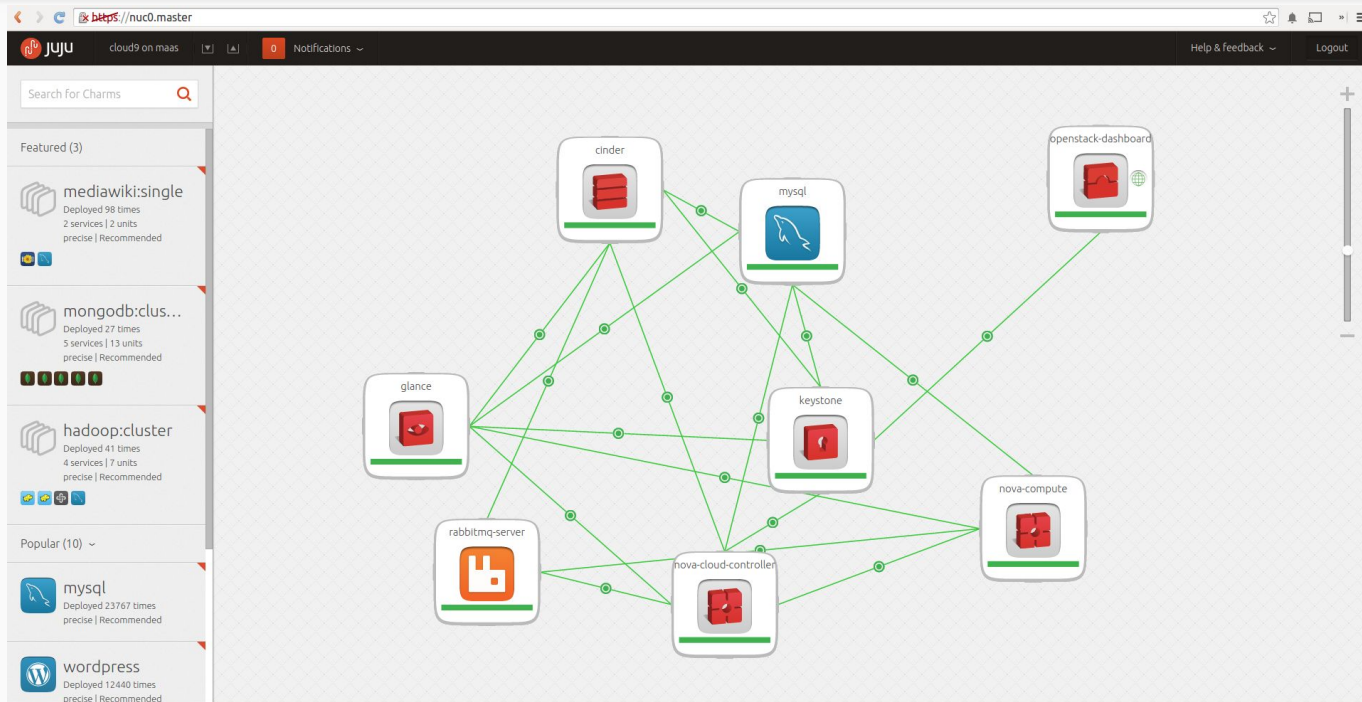
Continuous build status

Official Builds

Build Type	Status	Artifacts
Linux CPU	Ubuntu CC passing	PyPI
Linux GPU	Ubuntu GPU PY3 passing	PyPI
Linux XLA	Ubuntu XLA failing	TBA
macOS	MacOS PY2 CC passing	PyPI
Windows CPU	Windows CPU passing	PyPI
Windows GPU	Windows GPU passing	PyPI
Android	Android passing	Download 1.15.0
Raspberry Pi 0 and 1	Rpi01 py2 failing Rpi01 py3 failing	Py2 Py3
Raspberry Pi 2 and 3	Rpi23 py2 failing Rpi23 py3 passing	Py2 Py3



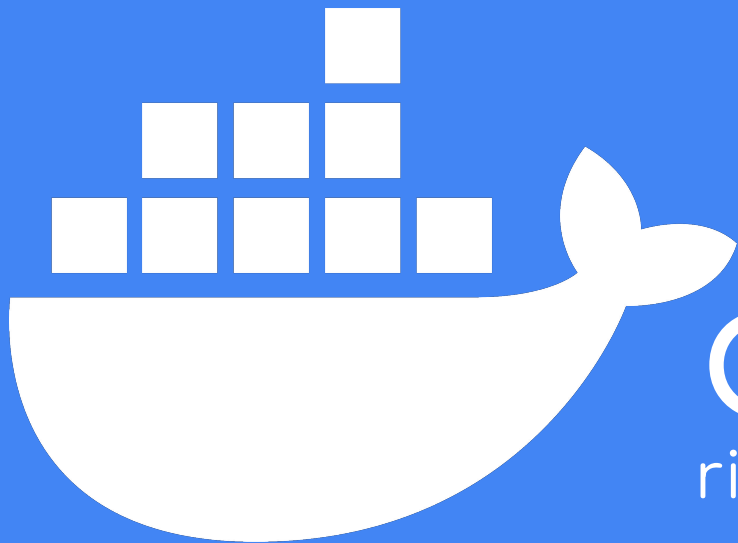
microservices architecture





“In software development perfect is not
an adjective, but a verb.”

- Uncle Bob



docker
ridgerun 2020