



Building Distributed System with Celery on Docker Swarm

Wei Lin

PyCon JP 2016

2016/09/22

About Me

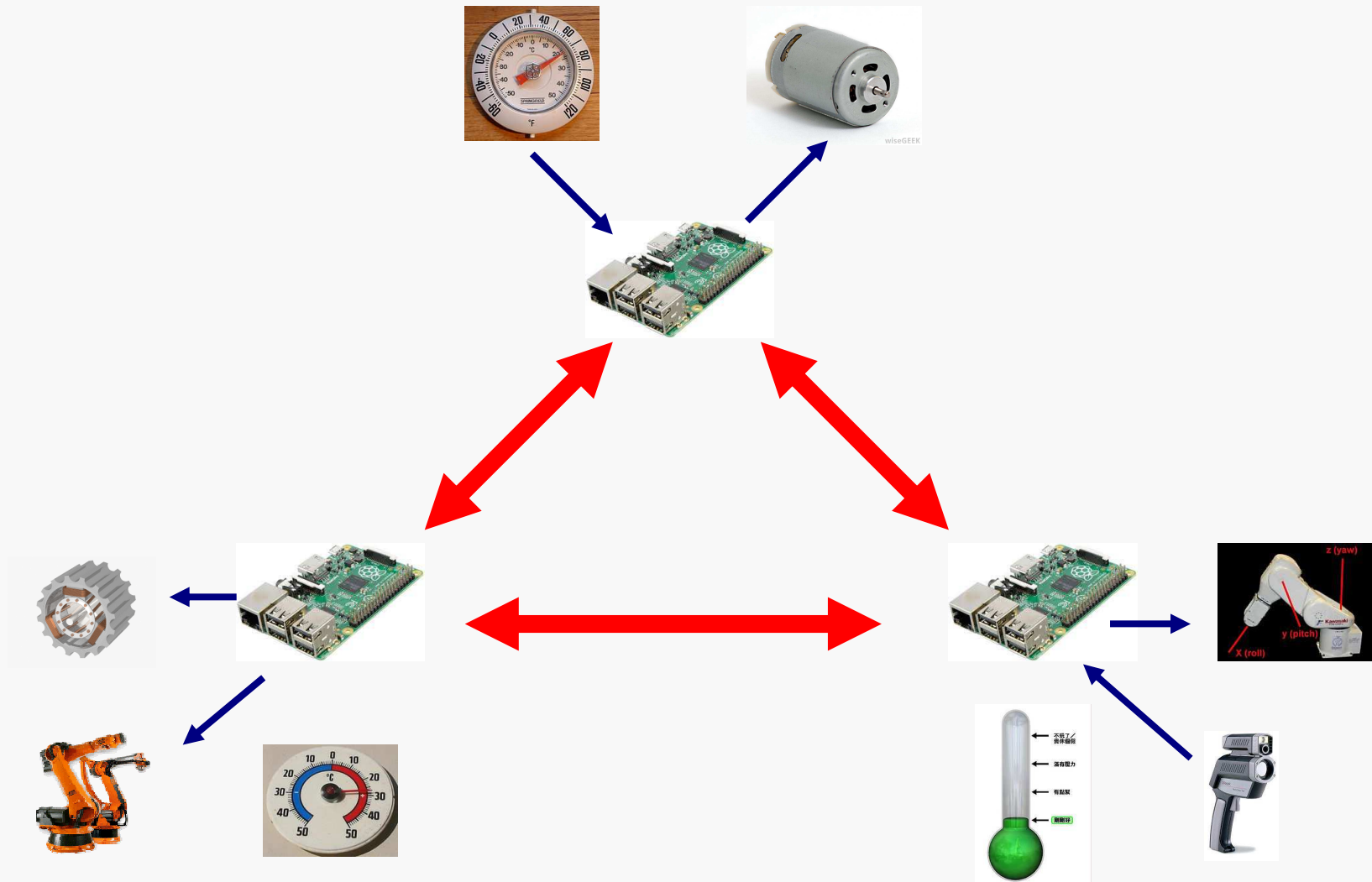
- Wei Lin
@Wei_1144
Wei1234c@gmail.com
<https://github.com/Wei1234c>
- Mostly worked in the fields of:
 - marketing
 - strategy-planning
 - project-management
- Materials of this talk:
<https://github.com/Wei1234c/PyCon-JP-2016-talk>

Parallel computing

<https://resin.io/blog/what-would-you-do-with-a-120-raspberry-pi-cluster/>



Distributed System



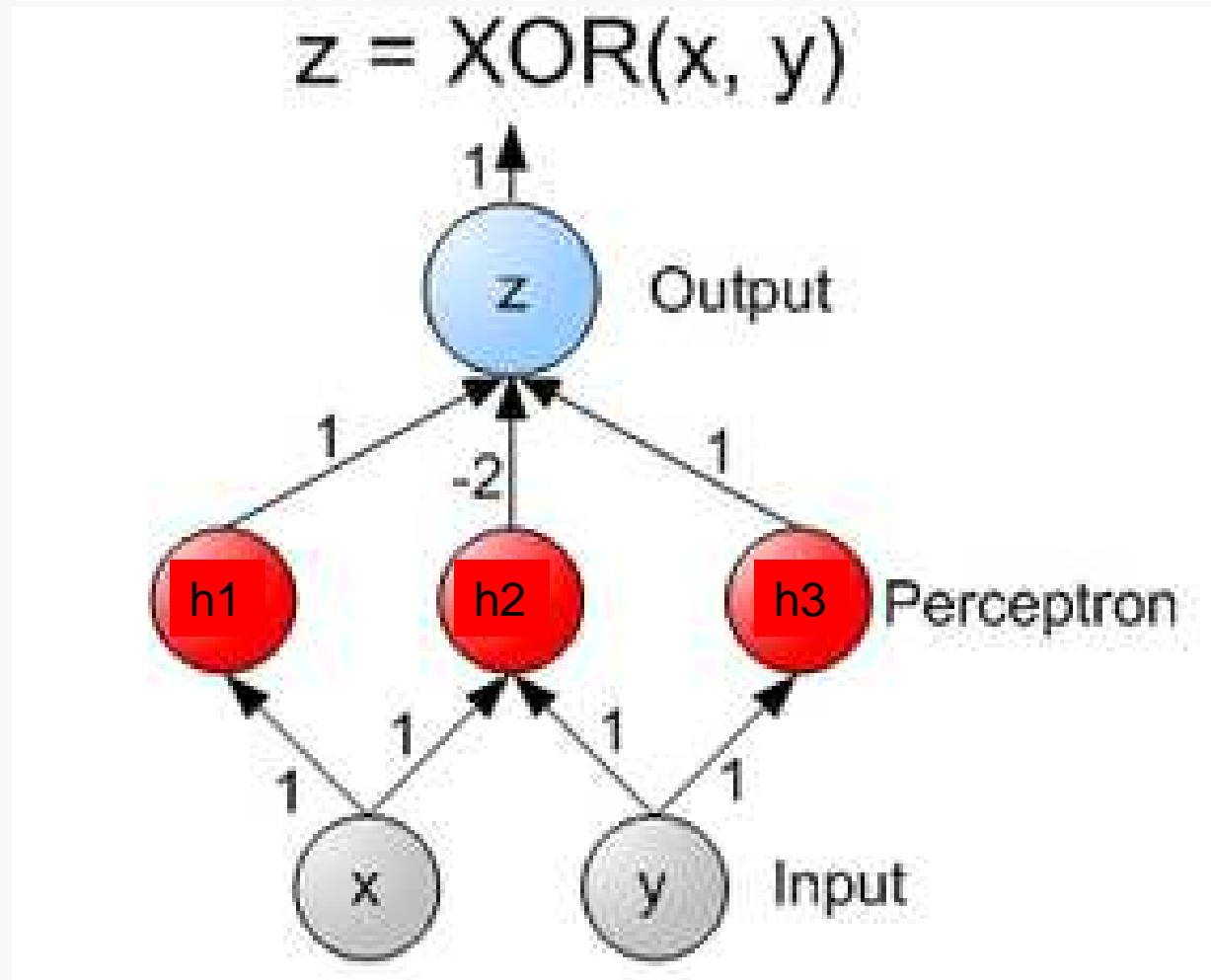
Distributed system



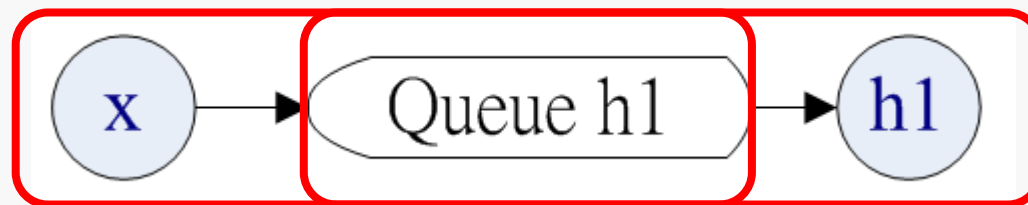
Distributed system (II)



Neural network



Communicate via Celery

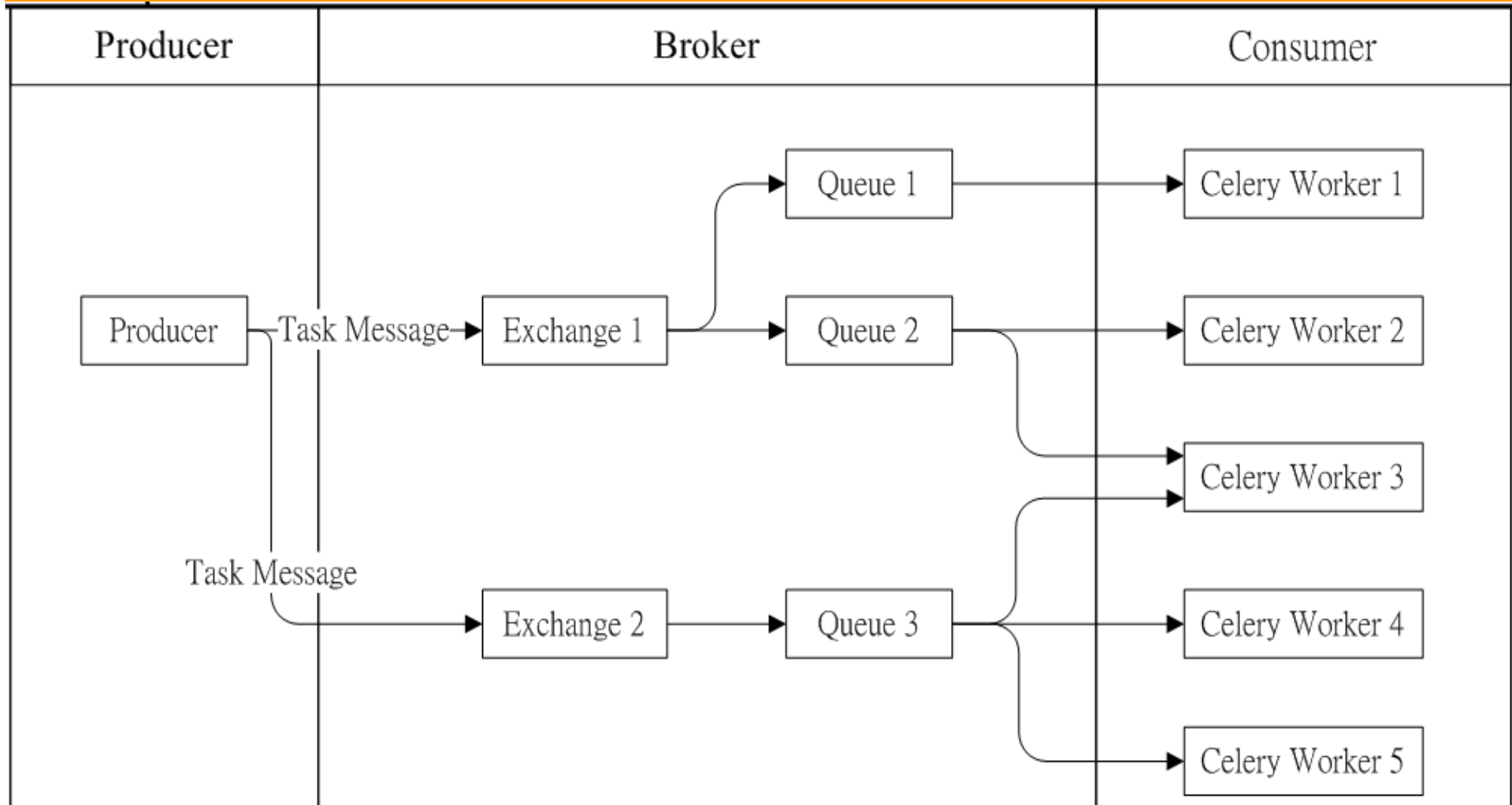




Celery



Celery flowchart



Send message to a specific queue



Send message

<http://docs.celeryproject.org/en/latest/userguide/calling.html>

```
from celery import Celery

app = Celery()
app.config_from_object('celeryconfig')

@app.task
def kick(neuron_id):
    myName = gethostname()
    log('{0} is kicking {1}.'.format(neuron_id, myName))
    ...
```



```
kick.apply_async(['neuron_x'])
```

```
kick.apply_async(['neuron_x'], routing_key = 'neuron_h1')
```





Configuration file

```
# Import Kombu classes
from kombu import Exchange, Queue

# CELERY_TIMEZONE & Misc.
CELERY_TIMEZONE = 'Asia/Taipei'
CELERYD_POOL_RESTARTS = True

# BROKER_URL
BROKER_URL = 'redis://192.168.0.114:6379/0'

# CELERY_RESULT_BACKEND
CELERY_RESULT_BACKEND = 'redis://192.168.0.114:6379/1'

# CELERY_IMPORTS
CELERY_IMPORTS = ('IoT.neuron',)

# CELERY_QUEUES
CELERY_QUEUES = (
    Queue('neuron_x', Exchange('celery', type = 'direct'), routing_key='neuron_x'),
    Queue('neuron_y', Exchange('celery', type = 'direct'), routing_key='neuron_y'),
    Queue('neuron_h1', Exchange('celery', type = 'direct'), routing_key='neuron_h1'),
    Queue('neuron_h2', Exchange('celery', type = 'direct'), routing_key='neuron_h2'),
    Queue('neuron_h3', Exchange('celery', type = 'direct'), routing_key='neuron_h3'),
    Queue('neuron_z', Exchange('celery', type = 'direct'), routing_key='neuron_z'),
)
```





Task Message Example

<http://docs.celeryproject.org/en/latest/internals/protocol.html>

```
{"id": "4cc7438e-afd4-4f8f-a2f3-f46567e7ca77",  
  "task": "celery.task.kick",  
  "args": ['neuron_x'],  
  "kwargs": {},  
  "retries": 0,  
  "routing_key": "neuron_h1"}
```



Routing Tasks

<http://docs.celeryproject.org/en/latest/userguide/routing.html>

```
# _____ CELERY_QUEUES _____  
CELERY_QUEUES = (  
    Queue('neuron_x', Exchange('celery', type = 'direct'), routing_key='neuron_x'),  
    Queue('neuron_y', Exchange('celery', type = 'direct'), routing_key='neuron_y'),  
    Queue('neuron_h1', Exchange('celery', type = 'direct'), routing_key='neuron_h1'),  
    Queue('neuron_h2', Exchange('celery', type = 'direct'), routing_key='neuron_h2'),  
    Queue('neuron_h3', Exchange('celery', type = 'direct'), routing_key='neuron_h3'),  
    Queue('neuron_z', Exchange('celery', type = 'direct'), routing_key='neuron_z'),  
)
```

Assign an unique worker
for a specific queue



Starting the Worker

<http://docs.celeryproject.org/en/latest/userguide/workers.html>
<https://github.com/celery/celery/issues/3065>

```
$ celery -A IoT worker -n neuron_h1 \
-Q neuron_h1 --concurrency=1
```

celery_projects

IoT

```
from celery import Celery
from kombu import Exchange, Queue

app = Celery()
app.config_from_object('celeryconfig')

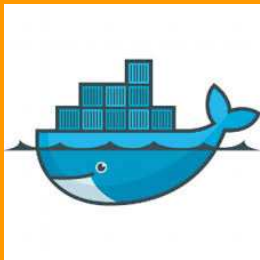
if __name__ == '__main__':
    app.start()
```

celery.py

neuron.py

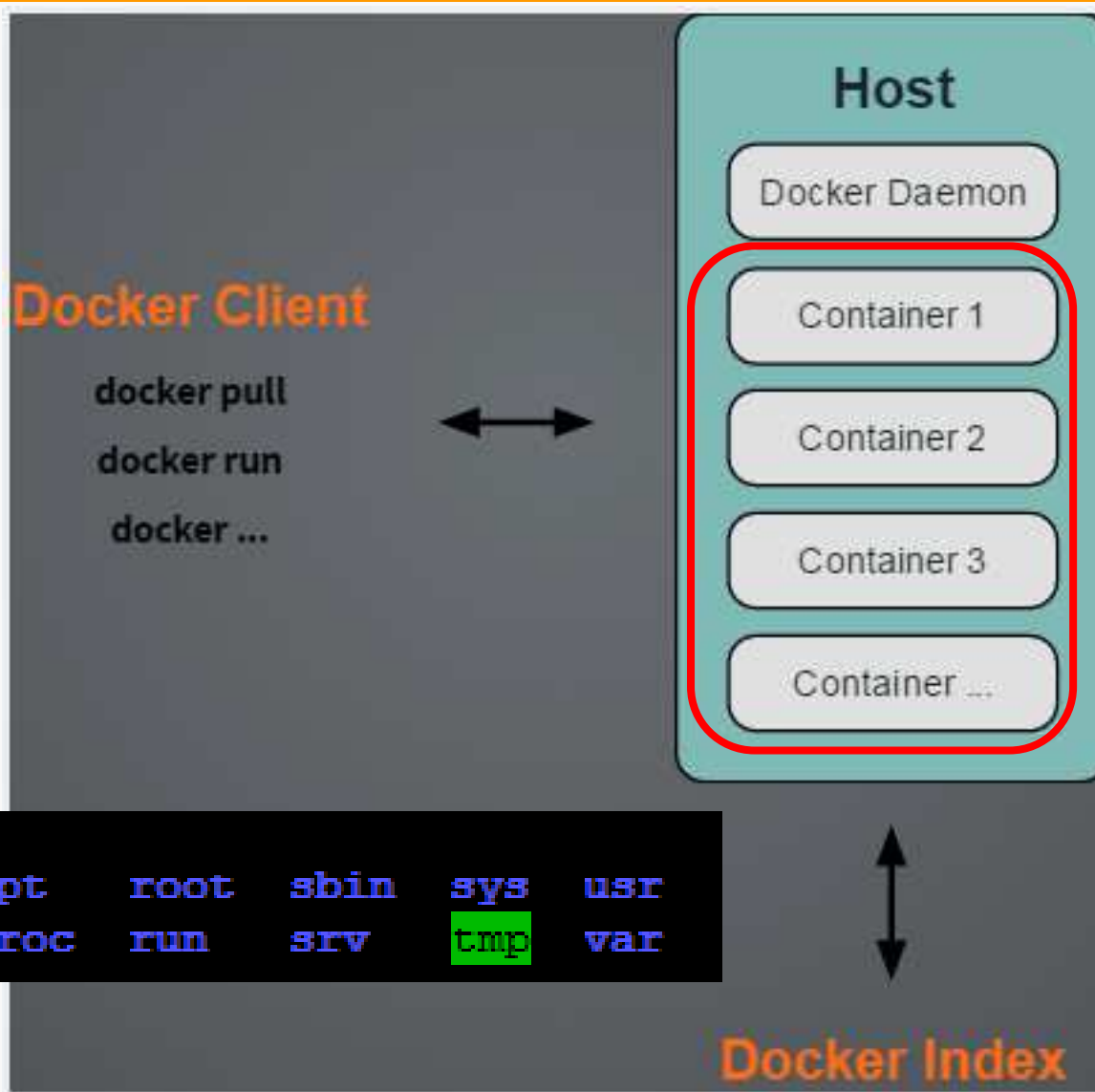


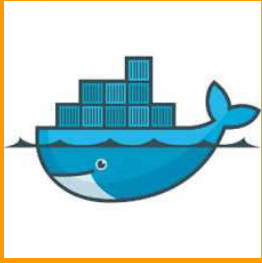
Docker



Host and containers

<https://denibertovic.com/talks/supercharge-development-env-using-docker/#/10>

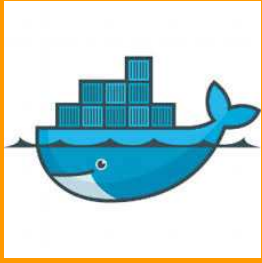




Run worker container

```
docker run -d --name=neuron_x --hostname=neuron_x \  
--volume=/data/celery_projects:/celery_projects \  
weil1234c/celery_armv7 \  
/bin/sh -c \  
"cd /celery_projects && \  
celery -A IoT worker -n %s -Q neuron_x --concurrency=1"
```

```
docker run -d --name=neuron_y --hostname=neuron_y \  
--volume=/data/celery_projects:/celery_projects \  
weil1234c/celery_armv7 \  
/bin/sh -c \  
"cd /celery_projects && \  
celery -A IoT worker -n %h -Q neuron_y --concurrency=1"
```

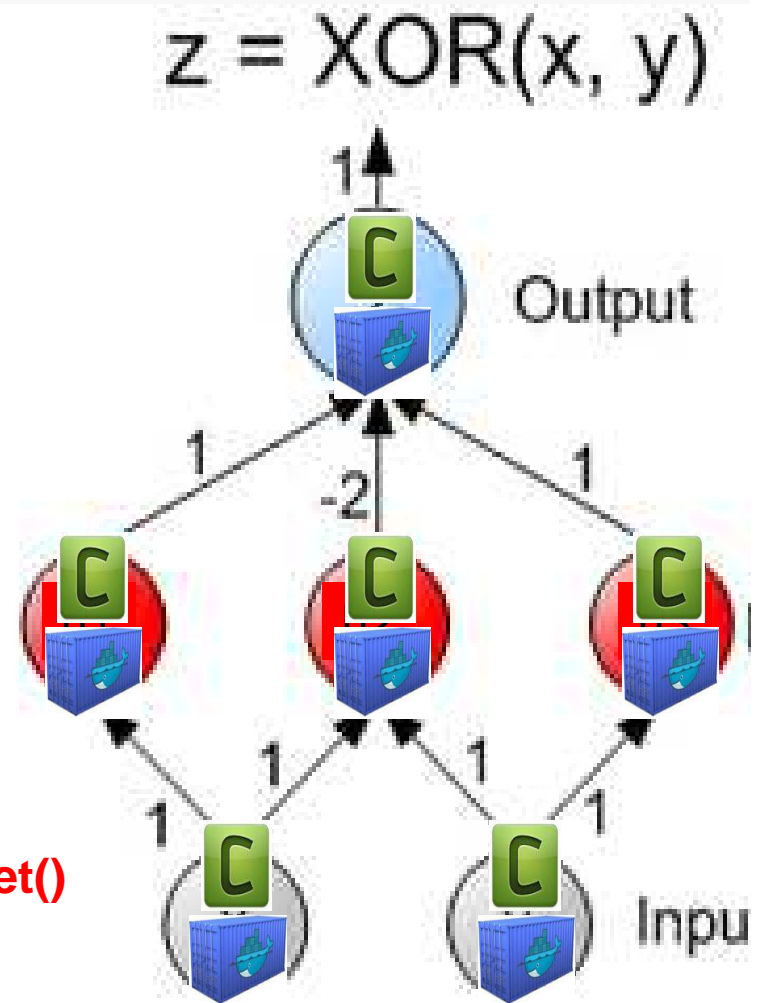


Deploy containers



Celery worker containers as neurons

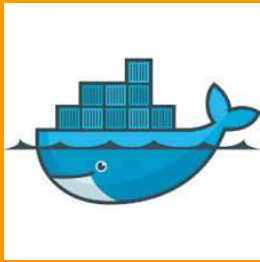
```
kick.apply_async( routing_key = 'neuron_h1')  
getData.apply_async(routing_key = 'neuron_h1') .get()
```



“Celery + Docker-Swarm”

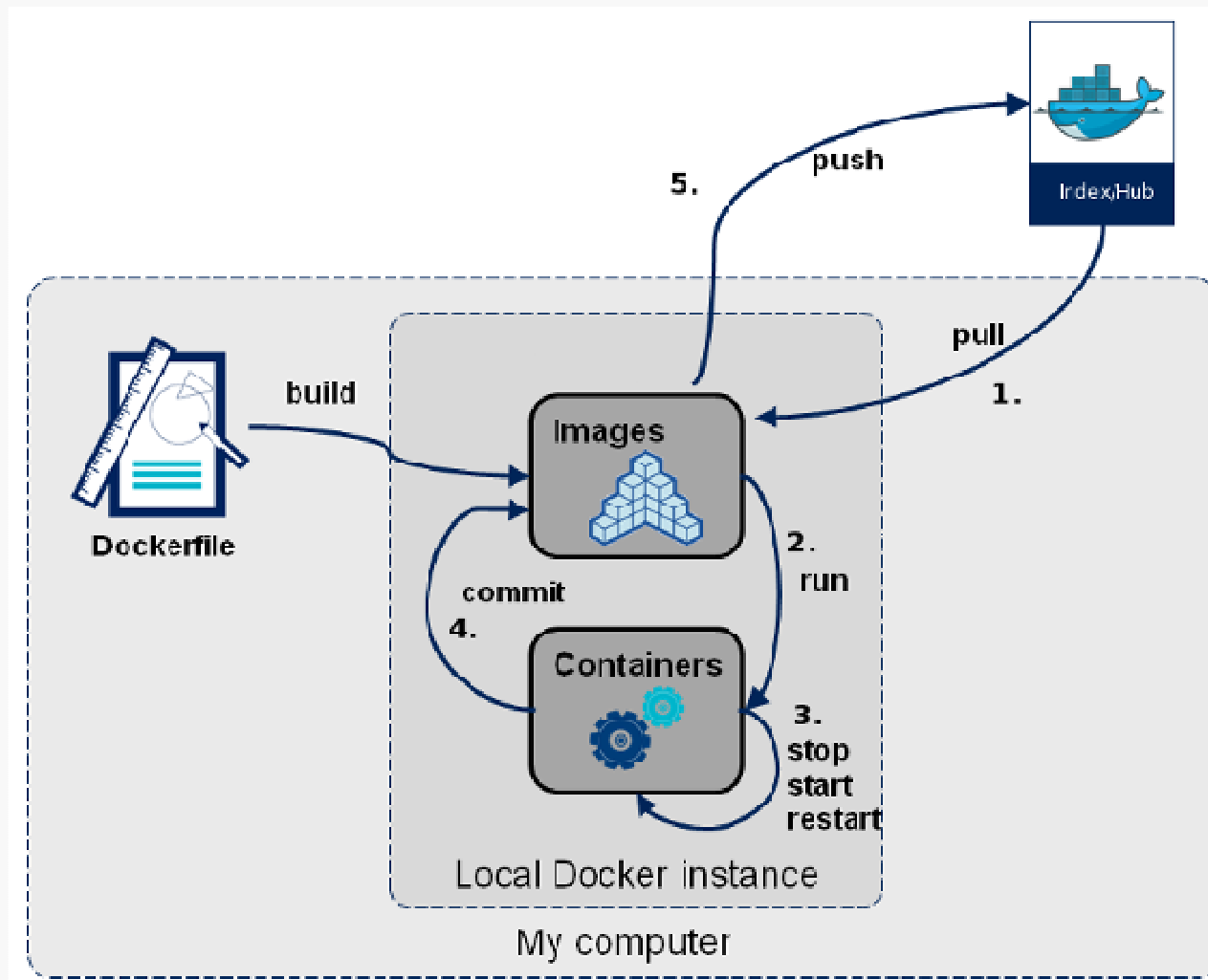
So ?

I think...



Docker file/image/container

<https://denibertovic.com/talks/supercharge-development-env-using-docker/#/11>



Think OO

```
# Celery Worker Dockerfile
# 20151229
```

```
FROM wei1234c/python_armv7
```

```
MAINTAINER Wei Lin
```

```
ENV TERM linux
```

```
# Add user pi
```

```
RUN \
```

```
    useradd -G adm,sudo,users -s /bin/bash -m pi && \
    echo 'pi:raspberry' | chpasswd
```

```
# Install Celery
```

```
RUN \
```

```
    pip3 install -U redis celery flower sqlalchemy pymongo
```

```
WORKDIR /data
```

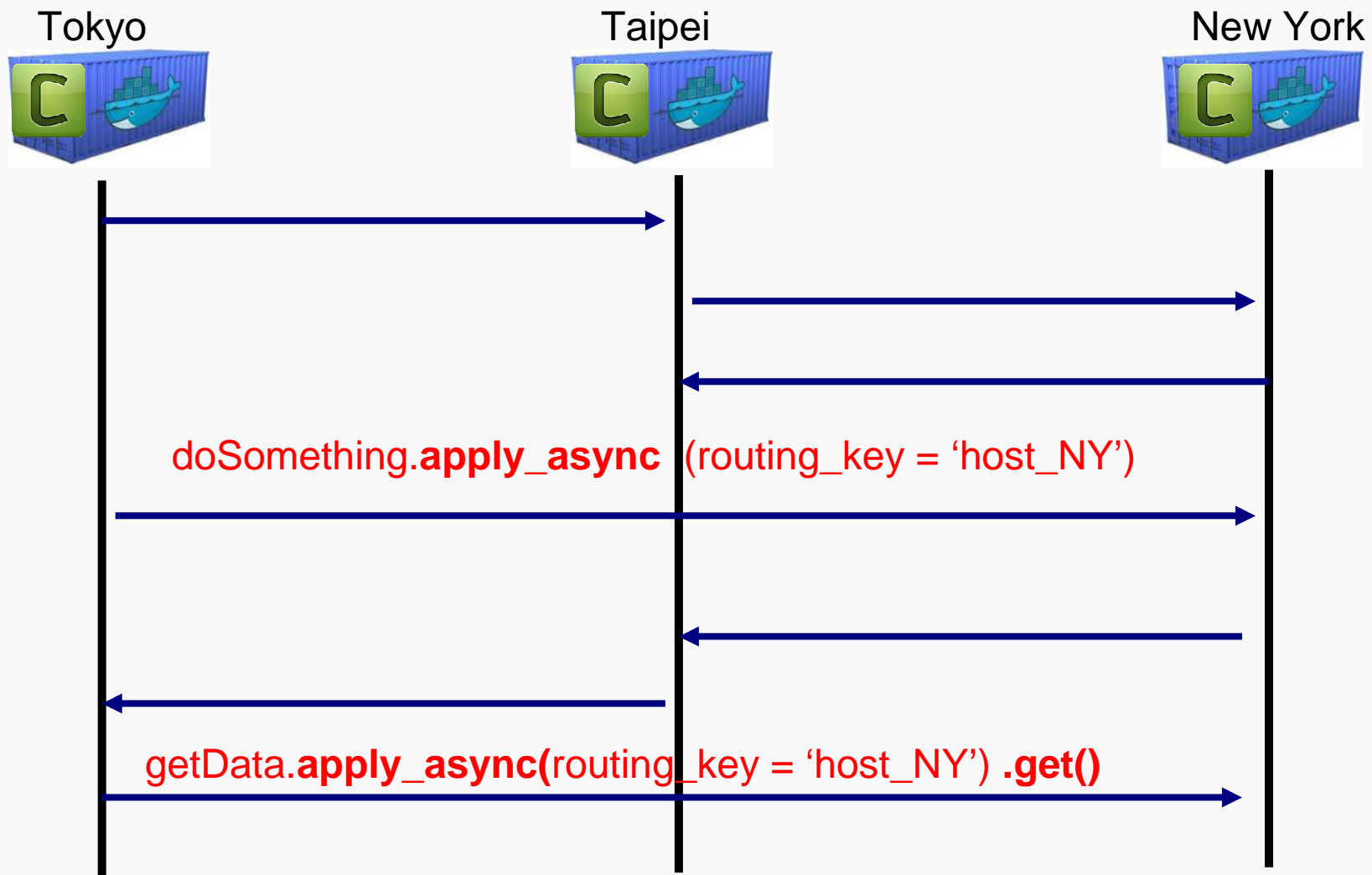
```
USER pi
```

```
EXPOSE 5555
```

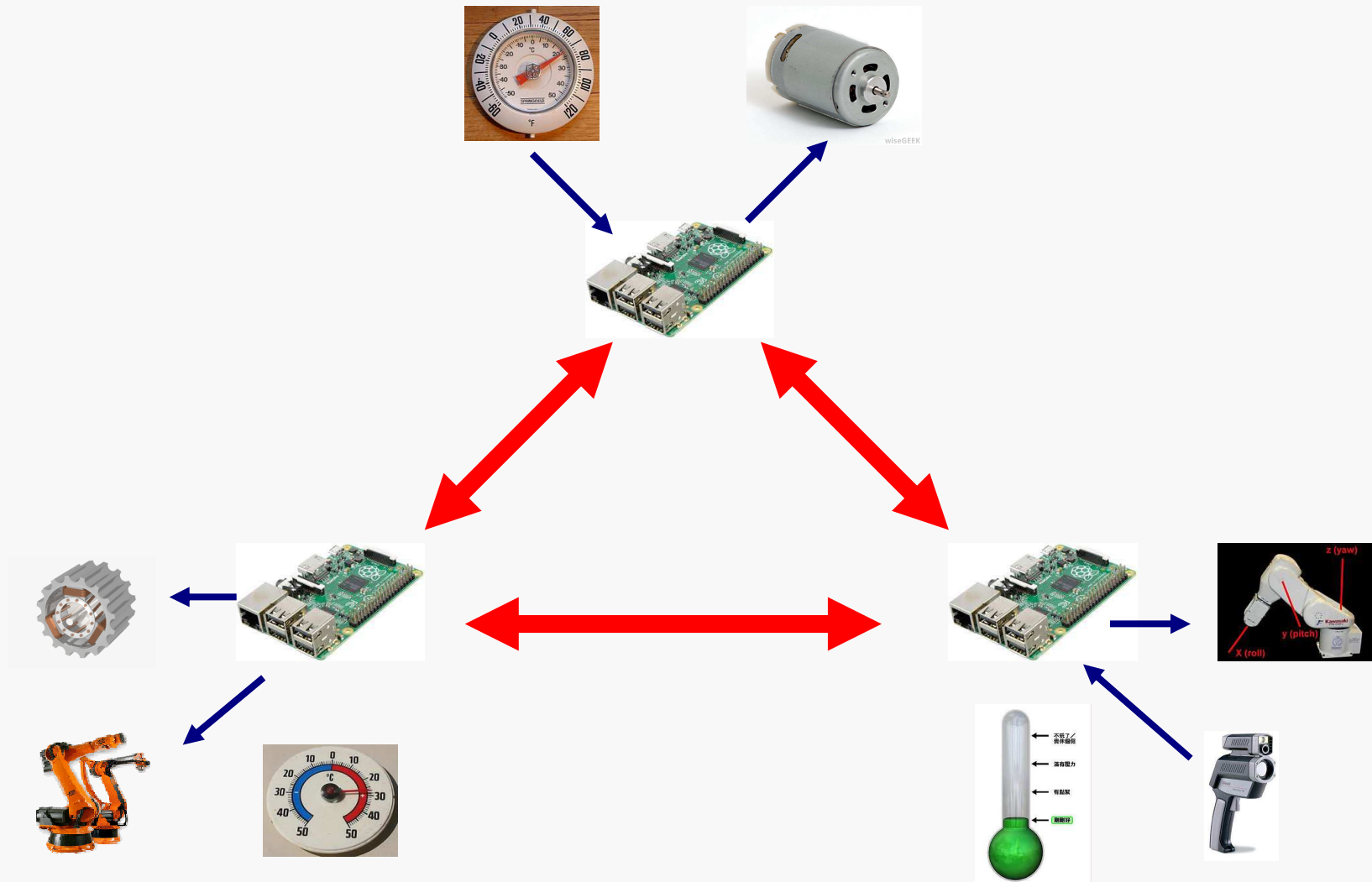
```
CMD ["bash"]
```

- “From” to Inherit
- Dockerfile to Encapsulate
- Docker Image as Class
- Docker Container as Object

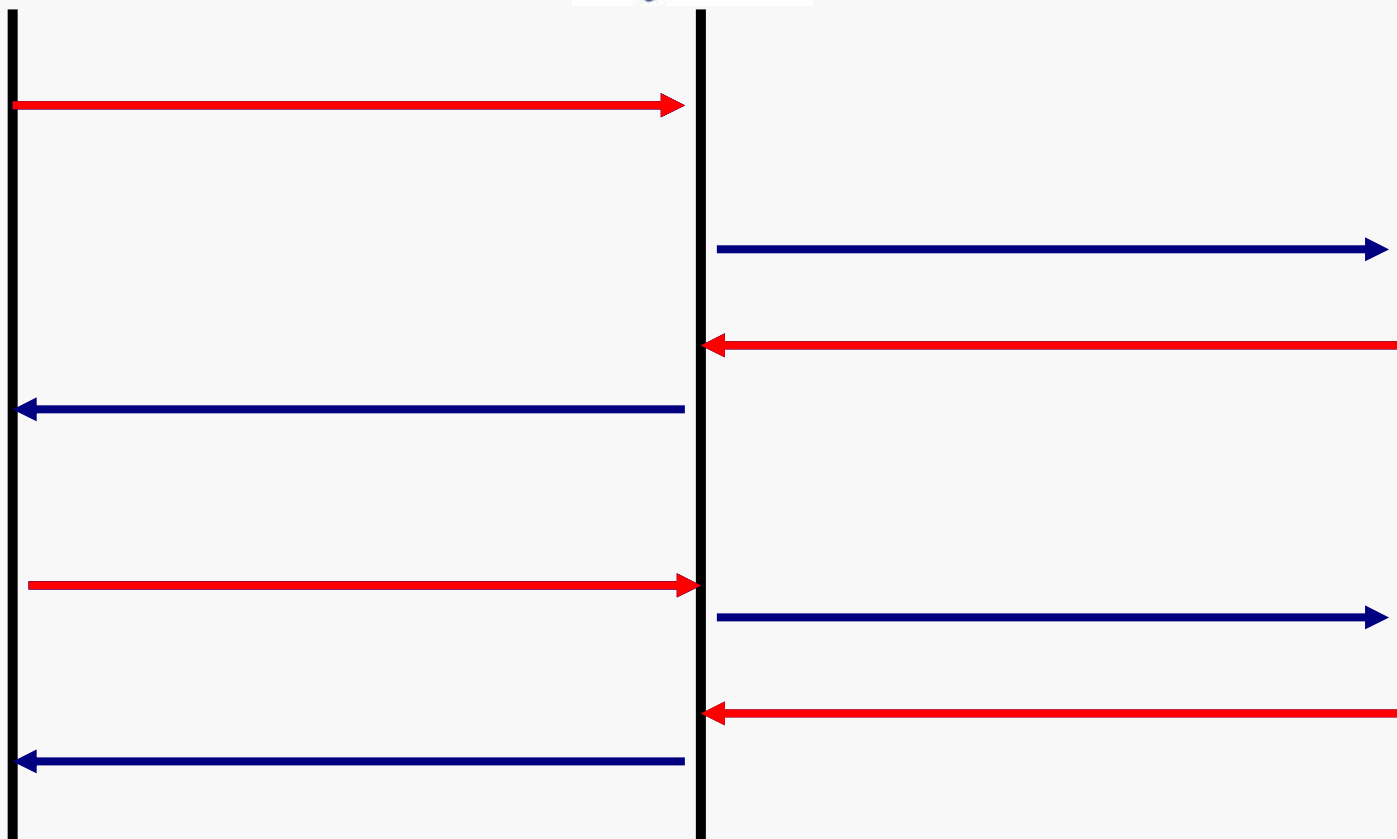
Docker Swarm & Celery

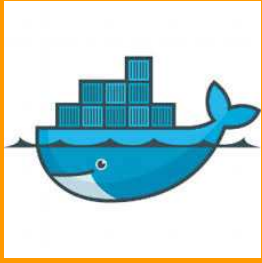


Distributed System



OOAD





Bound services

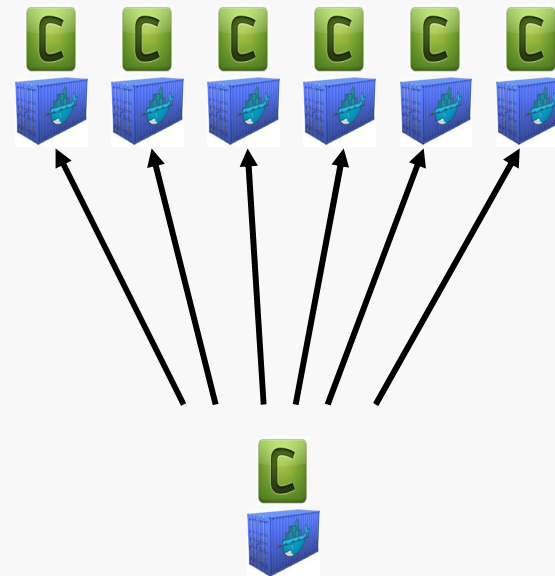
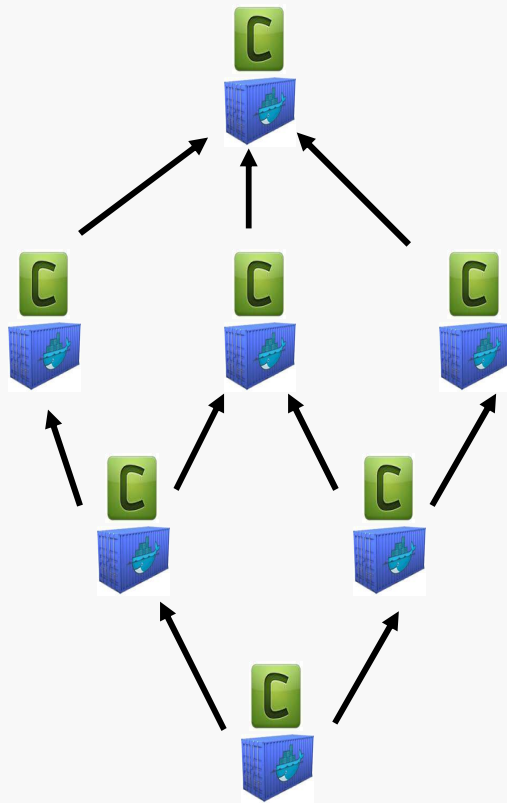


Parallel computing

<https://resin.io/blog/what-would-you-do-with-a-120-raspberry-pi-cluster/>



parallel computing \subseteq distributed computing



How to do Parallel Computing

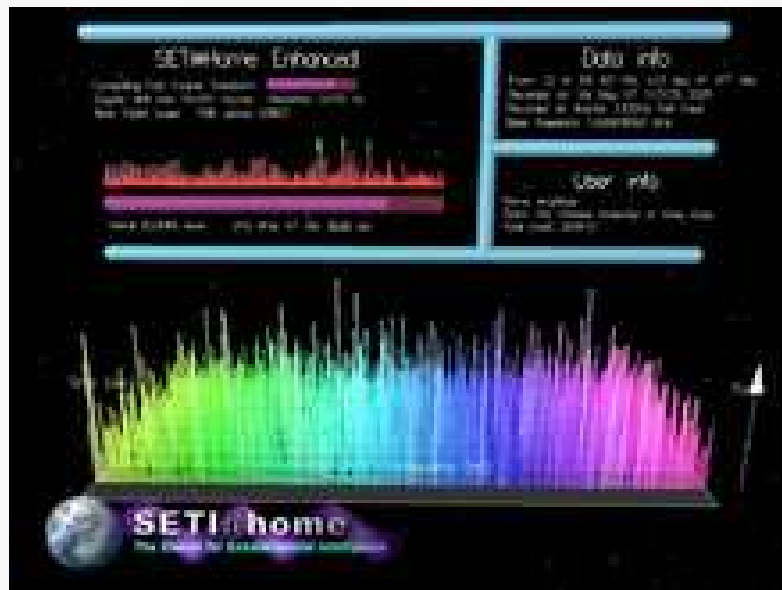
1. Run and deploy Docker containers across Docker Swarm cluster, with a Celery worker process running in each container.
2. When initiate worker processes, **don't assign "Queue" parameter**, so **ALL** the workers will listen to the **default queue** in broker.
3. Sending bunch of messages to broker **without setting routing_key parameter**, the messages will be routed to the **default queue**.
4. In a parallel fashion, all workers will pick up messages from the default queue and do the tasks.

Deploy parallel computing

- Deploy containers over Docker Swarm
 - deploy containers
 - `docker-compose up -d`
 - `docker-compose scale celery= n` (for example)
- Send task messages to worker containers

```
results = [ doSomething.apply_async([data]) for data in dataset ]
```
- But, Docker Swarm needs
root password-less SSH

SETI@home



One for all, All for one

https://github.com/Wei1234c/OneForAll_AllForOne/blob/master/celery_projects/OneForAll_AllForOne.md

- run a Docker image to join distributed parallel computing cluster, that's all.
`docker run -d wei1234c/musketeer` (for example)

Summary

- Easy to use
 - `doSomething.apply_async ([xxx], routing_key = 'host_NY')`
 - `getData.apply_async(routing_key = 'host_NY') .get()`
- Advantages of a distributed system
 - Decoupling, DI/IC (dependency injection / inversion of control)
 - Distributed = Shared
 - Load-Balancing
- Caution:
 - See ref: [Distributed systems theory for the distributed systems engineer](#)
- Take a look of [Canvas](#) in Celery
 - Chain
 - Group

Source Code

- Celery on Docker Swarm
 - using “Word Count” as an example of parallel computing.
 - https://github.com/Wei1234c/CeleryOnDockerSwarm/tree/master/celery_projects
- IoT as Brain
 - simulate an artificial neural-network of 6 neurons.
 - https://github.com/Wei1234c/IOTasBrain/tree/master/celery_projects
- One for all, all for one
 - follow the SETI@home paradigm.
 - on each machine, running a single Docker container to join the cluster.
 - https://github.com/Wei1234c/OneForAll_AllForOne/tree/master/celery_projects

References

References :

- **Distributed systems theory for the distributed systems engineer**
 - <http://the-paper-trail.org/blog/distributed-systems-theory-for-the-distributed-systems-engineer/>
- **Celery user guide**
 - <http://docs.celeryproject.org/en/latest/userguide/>
- **Docker document**
 - <https://docs.docker.com/>
- **MQTT Message Type and Flows**
 - <http://blog.maxkit.com.tw/2014/02/mqttmessage-type-and-flows.html>
- **Celery on Docker Swarm**
 - <https://github.com/Wei1234c/PyCon-JP-2016-talk/blob/master/Celery%20on%20Docker%20Swarm%20-%20PyCon%20JP%202016.md>
- **IoT as Brain**
 - <https://github.com/Wei1234c/PyCon-JP-2016-talk/blob/master/IoT%20as%20Brain%20-%20PyCon%20JP%202016.md>