# Autonomous Drone Engineer

## D3 – Docker Containers

Paul.Guermonprez@intel.com
*Autonomous Drone Solutions Architect*

# Software Partitioning

# Production / Prototyping OSes

Intel Aero is shipped with a **Yocto Linux** build: Yocto is great as an embedded build system for professionals, including in the aeronautics sector where it is widely used.

But it is not very friendly for **rapid prototyping:** Some may prefer an OS like Ubuntu Linux https://www.ubuntu.com *(© 2017 Canonical) or Debian.*

**How do we get the best of both worlds?**

# Client / Operator containment

Drone developers create **flight stacks** to operate the drone.
It has to be validated, sometimes certified to make sure it's safe to fly.

Drone clients may want to add their code to the drone. Their **software payload** can perform various tasks like image analysis during the flight and internet access. It does not have the same limitations as the flight code.

To speed up development and keep the drone safe to fly,
it is better to keep the two software stacks **separated**.

**We need a separation mechanism**
**between the flight stack and the software payload**

# Docker on Intel Aero

There’s several possible mechanisms to create this separation and choice of OS for deployment. On Intel Aero we support **containers**.

A container is a set of resources (cpu, memory …) allocated to a set of processes. It’s a lot like a virtual machine but much lighter and efficient as it works at the process level and not the OS level.

It is very popular in the server world, where the toolbox **Docker** is allowing you to create development environments on your station (Linux, Mac, Windows) and migrate them to servers seamlessly.

**To summarize, containers behave like a virtual machine but also like a way to package and deploy apps.**

# Working with containers

# Typical usage

You choose an **existing base image** *Ex: Ubuntu:16.04*
Docker will download seamlessly if now available locally already

You launch, or **instantiate** this container *Ex: docker run*

You **connect** to the instance *Ex: docker exec*

You **install** packages, make changes to the OS …

You **commit** the changes to create a new image *Ex: docker commit*

Next time, you can **instantiate** this custom image
Or you can move it to another machine for **deployment**

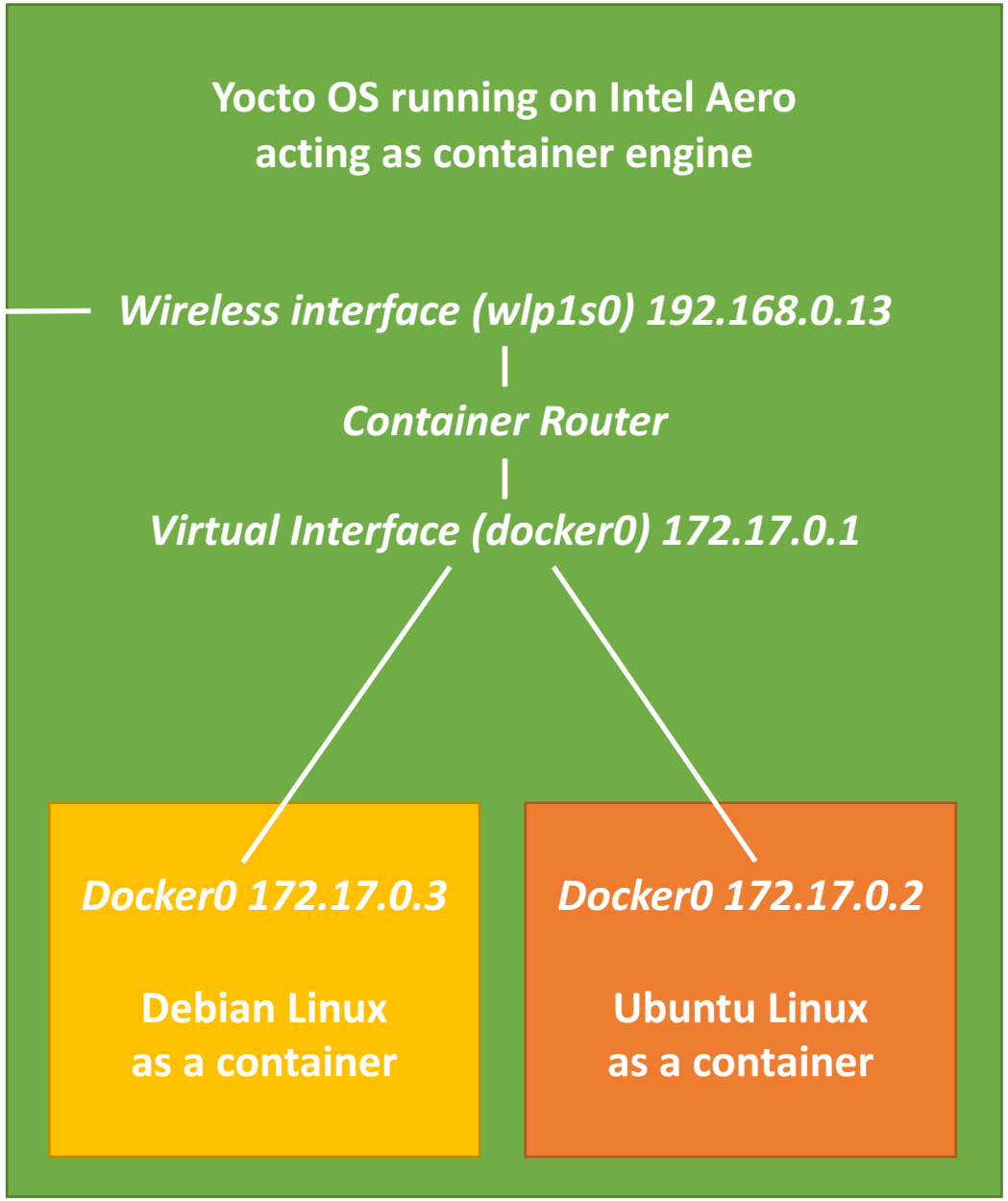
**Containers are not Virtual Machines.**

# Network Architecture

The container engine has direct **network access** and is acting as a **router** and DHCP server.

Containers have network access, but need to ask the container engine if they want to host servers. (it's exactly like opening a port in your home router).

*Ex: -p 2222:22*
*will forward port 2222 of the engine to port 22 of the container.*



# Example

# Thanks

Paul.Guermonprez@intel.com