# Comparing and contrasting Docker Compose and Open Horizon

## Introduction

Both Docker Compose and Open Horizon are tools for managing the deployment lifecycle of containerized applications, but there are significant differences.  This article will attempt to explain service software lifecycle management, and then compare and contrast the approaches used by these two tools.  This article assumes familiarity with containerized software, Dockerfiles, Docker Compose files, and Open Horizon components.

## Service software lifecycle management

For the purposes of this article, we will decompose the concept of service deployment into the following lifecycle: Publishing, Execution, Operation (Updating, Monitoring, Restarting), and Removal.  Publishing is about specifying a container image (e.g. DockerHub, quay.io, IBM Cloud Container Registry), pulling the image, and storing it at the deployment location.  Execution covers optionally deploying secrets and any other configuration and then starting the service within a container engine.  Operation involves inspecting details about a running image, updating the image when new versions become available, and restarting an image if the host restarts or the image terminates unexpectedly.  Removal includes stopping a running image and optionally removing any resources it may be using from the deployment location.

Docker Compose and Open Horizon may then be compared using the above lifecycle stages in a table:

| Stages | Docker Compose | Open Horizon |
|---|---|---|
| Publishing | manually run "docker-compose pull" on destination<br><br>not typically needed since this is included in Execution step below | automatically triggered on destination by Agent when an agreement is formed |
| Execution | manually run "docker-compose up" on destination | automatically run on destination by Agent after publishing completes successfully |
| Operation | manually run "docker-compose ps" on destination<br><br>does not otherwise monitor or alert to runtime failures<br><br>docker-compose up to manually restart if a new service version is published | Agent automatically monitors running services and implements restarts and rollbacks as needed<br><br>If a new service version is published, agreement is terminated and re-negotiated |
| Removal | manually run "docker-compose down" on destination | If an agreement is terminated, Agent will automatically halt and remove running services |

In summary, Docker Compose is a tool for an operator to manually administer the service software lifecycle directly on deployment hosts.  Open Horizon is a tool for an operator to *remotely* specify the conditions under which the service software lifecycle should be autonomously administered on each deployment location by the Open Horizon Agent.

## Operating environment

Docker Compose is designed for Linux Hosts and requires the Docker engine runtime.  It is not compatible with other container runtimes.  It can operate on macOS and Windows hosts using Docker Desktop.  It cannot be used to deploy containers to a Kubernetes cluster.

Open Horizon is designed for both Kubernetes clusters and Linux hosts, and is compatible with both Docker and podman runtimes.  It can deploy to Linux and macOS hosts using the Device Agent and to Kubernetes clusters using the Cluster Agent.  It can both deploy container images to, and bi-directionally synchronize machine learning assets with, the destination device or cluster.

## Dependency management at load-time versus at run-time

Concepts to understand:

**Top-level service vs dependency (or required service)**: A top-level service is the functionality that you intend to deploy.  A dependency or required service is one that is only used because your intended top-level service needs it.

**Stateful vs stateless service**: A service is stateful when it retains or persists information from one invocation to the next.  A service is stateless when each invocation is independently sent all of the information it requires in each request.

**Singleton vs Multiple (sharable property in Service Definition file)**: "The value of this field determines how many instances of the service's containers will be running on a node when the service is deployed more than once to the same node."  You might use the `singleton` value if your environment is resource-constrained and you cannot run more than one instance of a service.  Another reason to use `singleton` is if a service is stateful.  If the service is stateless and you have the available resources to run more than one copy, you should choose `multiple` instead of `singleton`.

**Service Definition and Deployment Policies vs Node (Deployment) Pattern**: Use the former when you have one or more top-level services that have independent lifecycles.  This is the recommended approach.  Use the latter only when you have a single application composed of multiple top-level services that have interdependent lifecycles.

Docker Compose combines the Publishing and Execution phases of the service software lifecycle, collectively described as load-time. It has no awareness of which services are top-level services because it doesn't describe the dependency relationship between services. It *does* explicitly determine the sequence in which services should be started based on the order in which they appear in the Docker Compose file. This means that two top-level services could be started *before* the first top-level service's dependencies if that is the sequence in which they are described in the file.

Open Horizon allows a developer to describe a top-level service and its dependencies in a standalone Service Definition file. A deployer uses policy (or a pattern) to instruct which services should be running on a deployment host. It is not necessary to explicitly publish dependent services, since Open Horizon will do this autonomously. Top level services are started last, after all of their dependencies are started.