Docker Swarm

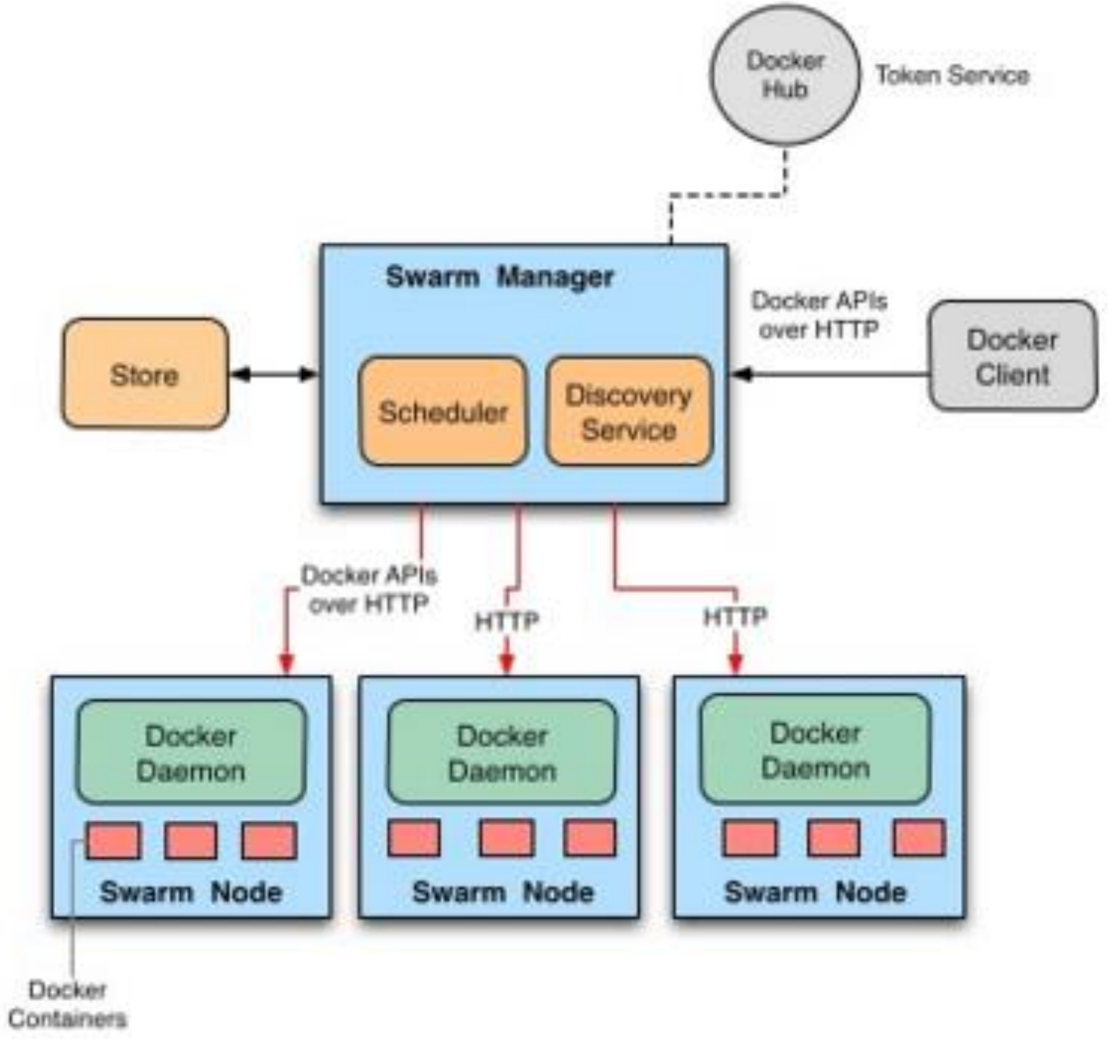
Docker Swarm overview

- Docker Swarm is native clustering for Docker.
- It turns a pool of Docker hosts into a single, virtual Docker host.
- Because Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts.
- Supported tools include, but are not limited to, the following:
    - Dokku
    - Docker Compose
    - Krane
    - Jenkins

Docker Components

- The Key Components of Swarm are:
  - Swarm manager
  - Swarm Node
  - Scheduler
  - Discovery

Docker Architecture



Swarm Manager

- The **swarm manager** is responsible for the entire cluster and manages the resources of multiple *Docker hosts* at scale.
- Uses Docker API to access docker daemon running on each node
- Nodes are added to the swarm manager by call back from discovery services fetch() function
- Elements of a Swarm Cluster Manager
    - Event Handler
    - Map of Nodes
    - Store
    - Options

Swarm Node

- Runtime Instance representing a node in the Cluster.
- Talks to the actual Host using Docker Client
- Created from a Discovery Entry fetched from a Discovery Service
- Elements of Swarm
  - Node
  - Id
  - IP Address (of remote host)
  - Map of Containers
  - Map of Images
  - Health state of the node
  - Total CPUs
  - Used CPUs
  - Total Memory
  - Used Memory

Scheduler

- Scheduler
  - Responsible for scheduling a container on a Node
  - Pluggable architecture – Bring your own scheduler
  - Elements of a Scheduler
    - – Placement Strategy Instance
    - – Array of Filters

The Docker Swarm scheduler strategies

- The `Docker Swarm` scheduler comes with multiple strategies
- These strategies are used to rank nodes using a scores computed by the strategy.
- `Docker Swarm` currently supports 2 strategies: –
    - BinPacking Strategy
    - Random Strategy Usage :
    - You can choose the strategy you want to use with the `-- strategy` flag of `swarm manage`

The Docker Swarm scheduler strategies

- BinPacking strategy
  - The BinPacking strategy will rank the nodes using their CPU and RAM available and will return the node the most packed already.
  - This avoid fragmentation, it will leave room for bigger containers on unused machines.
- Random strategy
  - The Random strategy, as it's name says, chooses a random node, it's used mainly for debug

Scheduler Filters

- Scheduler uses the following filters for container placement on a node
- Filters are divided into two categories,
    - node filters and
    - container configuration filters.
- Node filters operate on characteristics of the Docker host or on the configuration of the Docker daemon.
- Container configuration filters operate on characteristics of containers, or on the availability of images on a host.

Scheduler Filters

Node constraints can refer to Docker's default tags or to custom labels. Default tags are sourced from `docker info`. Often, they relate to properties of the Docker host. Currently, the default tags include:

- node to refer to the node by ID or name
- storagedriver
- executiondriver
- kernelversion
- Operatingsystem

$ docker daemon *--label storage=ssd* $

swarm join *--advertise=192.168.0.42:2375 token://XXXXXXXXXXXXXXXXXX*

Scheduler Filters

- The node `health` filter prevents the scheduler form running containers on unhealthy nodes.
- A node is considered unhealthy if the node is down or it can’t communicate with the cluster store.

Scheduler Filters

- Use an `affinity` filter to create “attractions” between containers.
- For example, you can run a container and instruct Swarm to schedule it next to another container based on these affinities:
  - container name or id
  - an image on the host
  - a custom label applied to the container

Scheduler Filters

- Use an `affinity` filter to create “attractions” between containers.
- For example, you can run a container and instruct Swarm to schedule it next to another container based on these affinities:
  - container name or id
  - an image on the host
  - a custom label applied to the container
- A label affinity allows you to filter based on a custom container label. For example, you can run a `nginx` container and apply the`com.example.type=frontend` custom label.

Scheduler Filters

- A container dependency filter co-schedules dependent containers on the same node. Currently, dependencies are declared as follows:
  - `--volumes-from=dependency` (shared volumes)
  - `--link=dependency:alias` (links)
  - `--net=container:dependency` (shared network stacks)

Scheduler Filters

- When the `port` filter is enabled, a container’s port configuration is used as a unique constraint.
- Docker Swarm selects a node where a particular port is available and unoccupied by another container or process.
- Required ports may be specified by mapping a host port, or using the host networking an exposing a port using the container configuration.

Swarm Store

- Stores the state of the Cluster.
- Currently implemented as a JSON file
- State is loaded in memory when the cluster starts
- Lifecycle events of the store
  - Get state for a key
  - Store the state of a container
  - Load all the data stored
  - Replace the state of the key with a new state
  - Delete the state

Discovery Services

- Discovery Service
- Helps Swarm Manager discover nodes
- Three main functions –
  - Register : Registers a new node
  - Watch : Callback method for Swarm Manager when a new Node is added to the Discovery Service
  - Fetch : Fetch the List of Entries