Extending Docker with APIs, Drivers, and Plugins

Anusha Ragunathan

Software Engineer, Docker

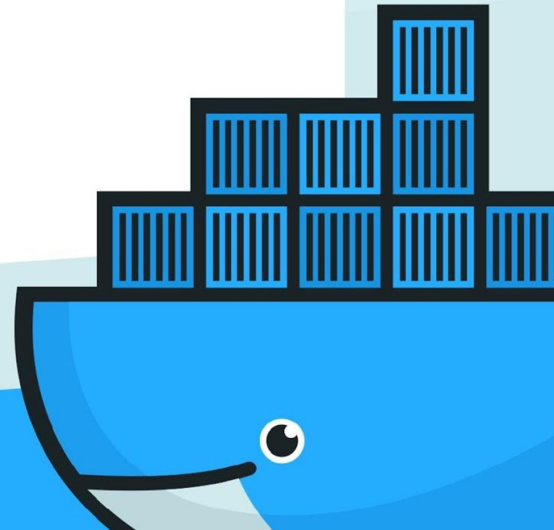
Arnaud Porterie

Sr. Engineering Manager, Docker

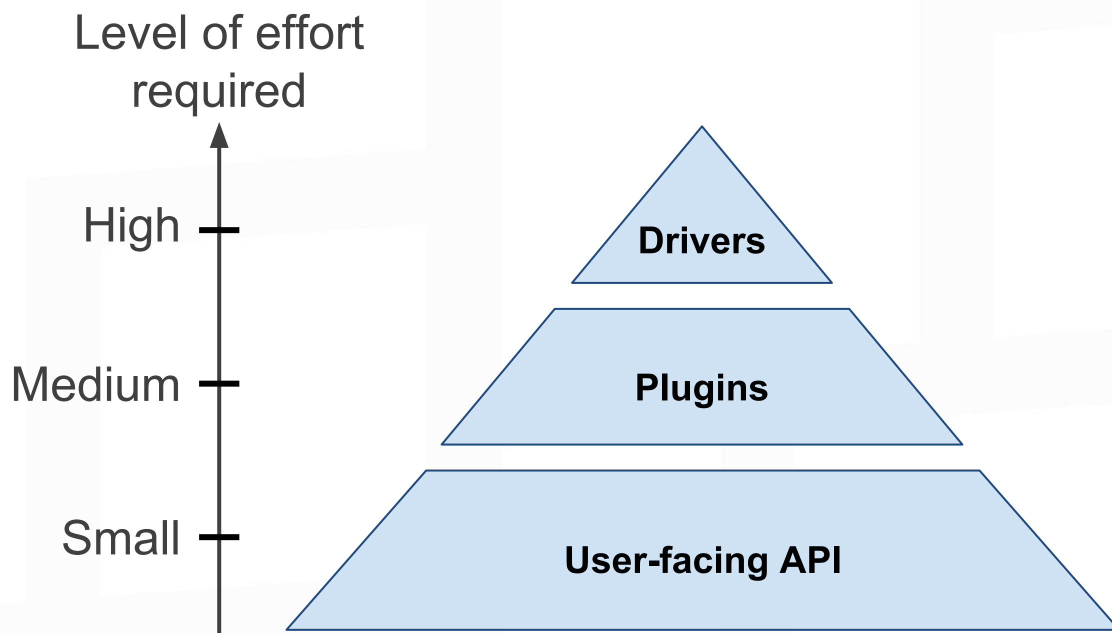
dockercon 16

“Batteries included but swappable”

— Anonymous



Docker extension points

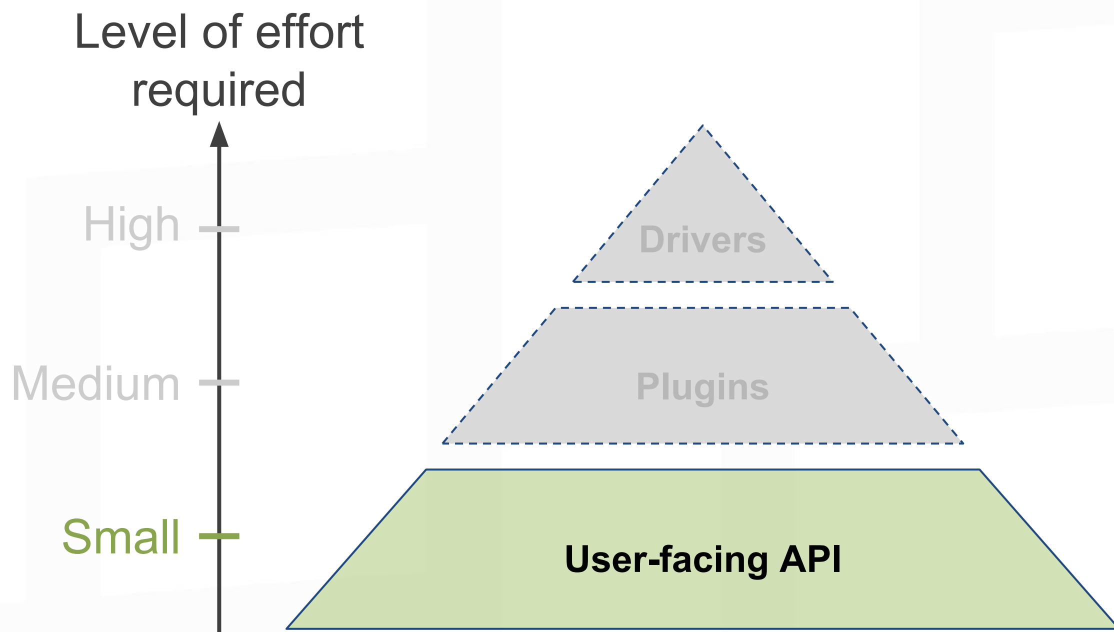


The user-facing API

Extending through observation

dockercon 16

User-facing API



dockercon 16

User-facing API

Introduction

All interactions with Docker go through an HTTP/JSON API

The daemon listens by default on `/var/run/docker.sock`

dockercon 16

User-facing API

Example interaction

```
# Listen on tcp:8080, print to stderr, and write to daemon's default socket.
$> socat -v TCP4-LISTEN:8080,fork UNIX-CLIENT:/var/run/docker.sock

# From another terminal (DOCKER_HOST informs the client where to connect to):
# $> DOCKER_HOST="tcp://localhost:8080" docker version
GET /v1.23/version HTTP/1.1
User-Agent: Docker-Client/1.11.2 (linux)

HTTP/1.1 200 OK
Content-Type: application/json

{"Version":"1.11.2","ApiVersion":"1.23","GitCommit":"b9f10c9", … }
```

dockercon 16

User-facing API

The events endpoint

- The `/events` endpoints is powerful for automation
- Gives live external visibility on **every operation** the daemon is doing
    - Action (e.g., container creation)
    - Context (e.g., image, container ID)

dockercon 16

User-facing API

The events endpoint

```
# Start listening to events (this command doesn't return).
$> docker events

# From another terminal:
# $> docker run --rm --name test busybox true
<timestamp> container create 439c5aa3 (image=busybox, name=test)
<timestamp> container attach 439c5aa3 (image=busybox, name=lonely_chandrasekhar)
<timestamp> network connect 9bddd27d (container=439c5aa3, name=bridge, type=bridge)
<timestamp> container start 439c5aa3 (image=busybox, name=test)
<timestamp> container die 439c5aa3 (exitCode=0, image=busybox, name=test)
<timestamp> network disconnect 9bddd27d (container=439c5aa3, name=bridge, type=bridge)
<timestamp> container destroy 439c5aa3 (image=busybox, name=test)
```

User-facing API

Extending using the events endpoint

- **Example use case**: Interlock (github.com/ehazlett/interlock)
  - Event driven plugin system
  - Routes events to extensions, such as HAProxy

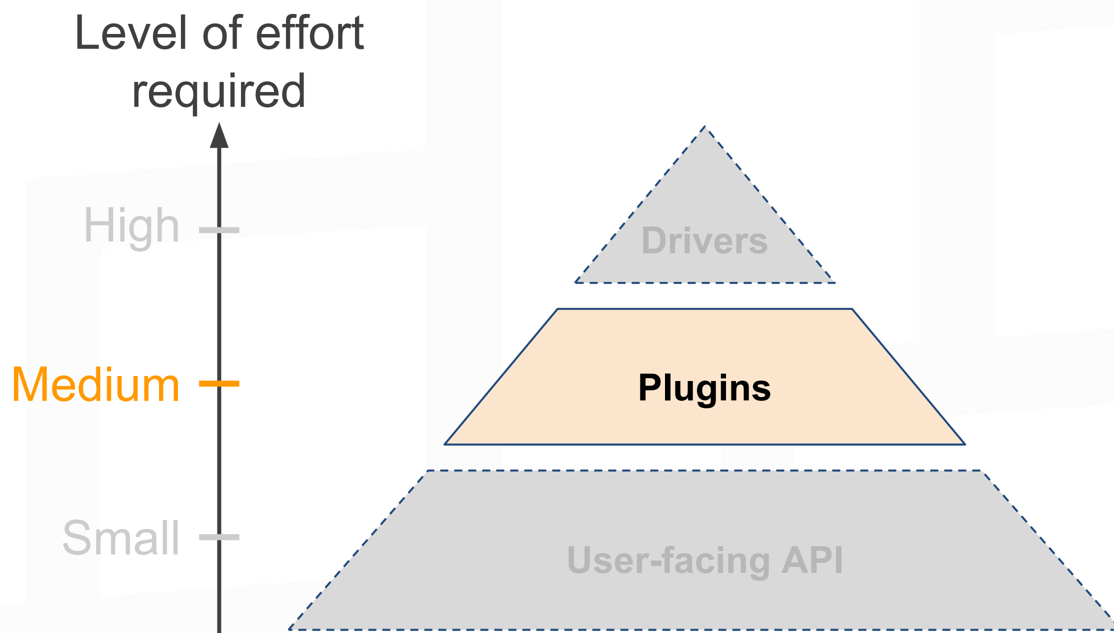
- The first Docker load-balancer
  - Drop-in solution that runs as a container and listens for events
  - Requires absolutely no change to Docker itself

Plugins

Past, present, and future

dockercon 16

Plugins

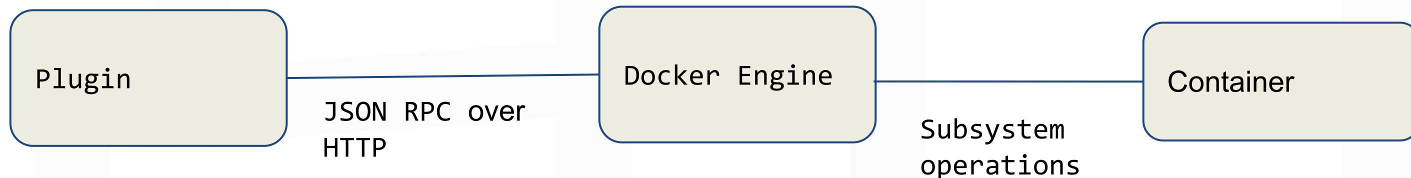


dockercon 16

Plugins

What are plugins?

- A process external to the docker engine that extends functionality of the docker engine.
  - Plugins available for volumes, networks and authorization subsystems.
  - Implements well defined plugin API for the specific subsystem.
  - Extend single node functionality or across the cluster.
  - Introduced in Docker 1.8



Plugins

Characteristics	Challenges in the past
Highly available services	Lack of boot ordering. No standard way to start containers before plugins.
Powerful distribution channels	Lack of streamlined discovery and distribution channels leads to customer confusion.
Defined/predictable runtime behavior	Lack of specification that defines plugin behavior.

Plugins

New plugin infrastructure

- Challenges resolved
    - Plugin distribution via the **new Docker Store!**
    - Plugins start and stop alongside docker engine (highly available)
    - Plugin behavior clearly defined in a plugin manifest specification.
- Experimental support in 1.12 (API, manifest spec subject to change)

All of the above plugin management and more via your favorite docker CLI/API !

Plugins

New plugin infrastructure

- Sample plugin: tiborvass/no-remove
  - Simple extension of **local** volume driver
  - Volume plugin API: Create, **Remove**, Mount, Unmount, List, ...
  - Implements a variation of **Remove**

Plugins

New plugin infrastructure

```
$> docker plugin install tiborvass/no-remove
Plugin “tiborvass/no-remove” is requesting the following privileges:
- network: [host]
- mount: [/data]
- device: [/dev/cpu_dma_latency]
Do you grant the above permissions? [y/N] y

$> docker plugin ls
NAME                    TAG       ACTIVE
tiborvass/no-remove     latest    true

$> docker plugin disable tiborvass/no-remove
NAME                    TAG       ACTIVE
tiborvass/no-remove     latest    false
```

New plugin infrastructure

New plugin infrastructure

```
$> docker plugin inspect tiborvass/no-remove
"Manifest": {
    "Description": "A test plugin for Docker",
    "Documentation": "https://docs.docker.com/engine/extend/plugins/",
    "Interface": {
        "Types": [
            "docker.volumedriver/1.0"
        ],
    },
    "Network": { "Type": "host" },
    "Mounts": [
    {
         "Source": "/data",
         "Destination": "/data",
         "Type": "bind"
    }]
}
```

dockercon 16

Plugins

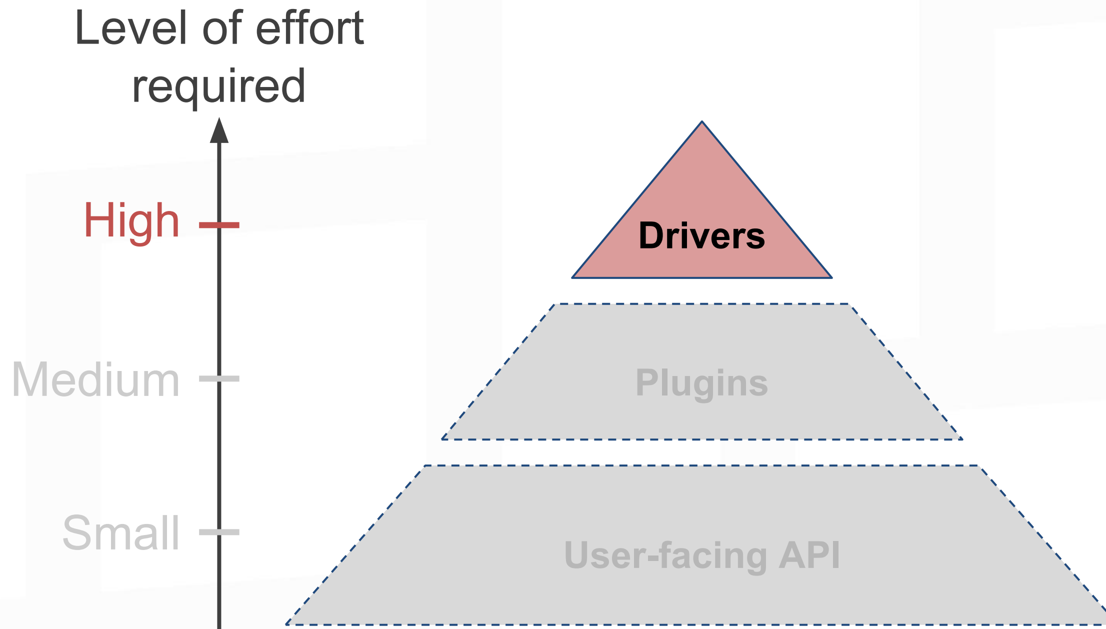
Future improvements

- **Per node plugins**
  - Stable support in 1.13.
- **Swarm-deployed plugins**
  - In 1.13:
    - Plugin deployment will be across the swarm and managed by the orchestrator.
    - Relies on the same plugin infrastructure under the hood.
  - Beyond 1.13, **customizing orchestration through plugins is possible**
    - E.g., placement strategies
    - E.g., scheduling modes

Drivers

Execution backend drivers

Drivers



Execution backend

OCI compatible runtimes

- OCI Runtime specification
  - Currently in 1.0.0-RC1
  - Defines an **industry standard** interface for runtimes

Execution backend

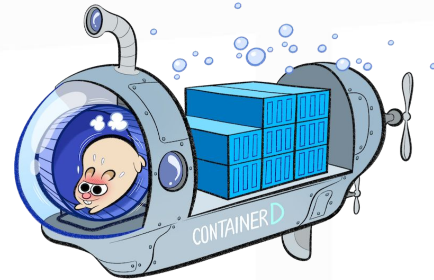
Introducing containerd and runC

runC (runc.io)

A tool for running containers according to the OCI specification

containerd (containerd.tools)

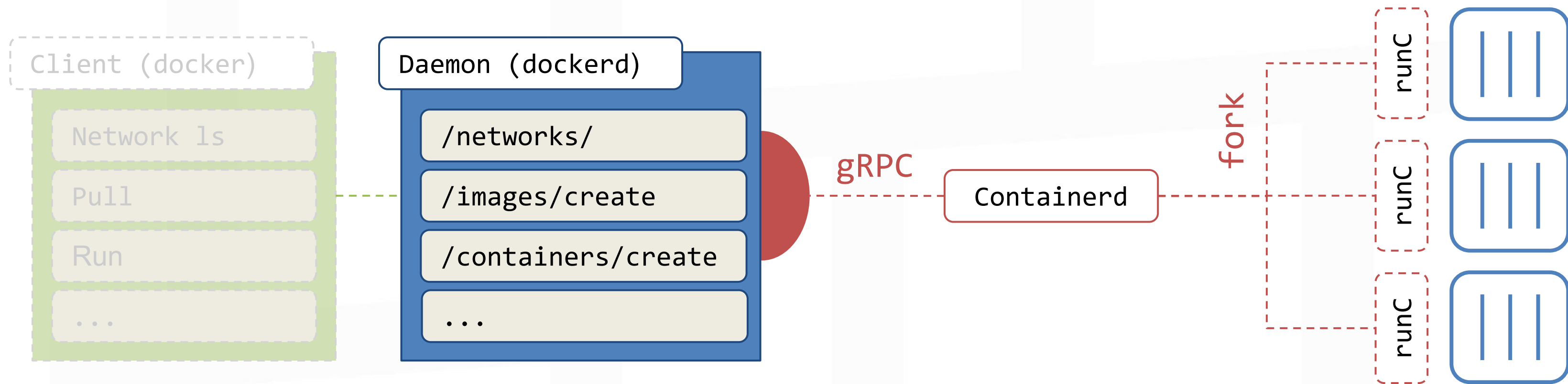
A daemon to control and supervise OCI compatible runtimes



Execution backend

Overview

As of Docker 1.11, the Engine relies on containerd and runC to run containers



dockercon 16

Execution backend

Example interaction

```
# An arbitrary collection of runtimes can be specified to the daemon.
$> dockerd --add-runtime custom=/bin/my_runtime

$> docker info | grep Runtimes
Runtimes: default custom

# All Docker installations have a system-specific default runtime (runC on Linux).
# Docker can be instructed to use a different runtime on a per-container basis.
$> docker run --runtime=default busybox true
$> docker run --runtime=custom busybox true

# The default runtime can be replaced at the daemon level.
$> dockerd --add-runtime custom=/bin/my_runtime --default-runtime=custom
```

Execution backend

Use cases for customizing the execution backend

- Platform specific runtimes
  - Solaris runZ
- Different workloads, different performance/security tradeoffs
  - Intel Clear Containers
  - Hyper_ runV (hypervisor-based runtime)
- What will the community invent next?

Key takeaways

Extension point	Level of effort	Key takeaways
User-facing API	Small	Learn what the Docker API offers Automate and extend by hooking into the API
Plugin infrastructure	Medium	Try the new plugin infrastructure in Docker 1.12 Build and distribute your own plugins
Drivers	High	Explore how to implement an execution backend in your favorite platform

Thank you!