Vert.x Clustering

Felix Hamann

23. Juni 2016

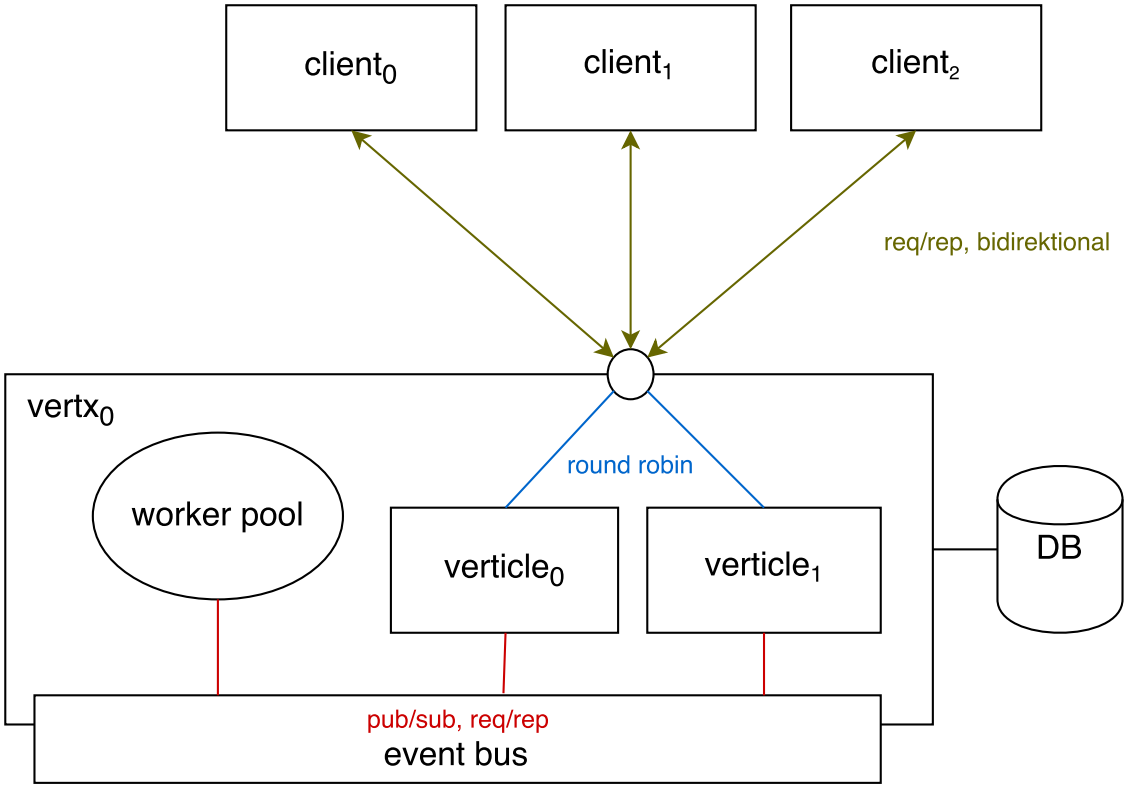
Was ist Clustering?

- Ein Zusammenschluss homogener Computer
- Jeder Computer erfüllt die gleiche Aufgabe
- In der Regel per lokalem Netzwerk verbunden
- Verhält sich wie ein großer Rechner

Warum clustern?

- Ausfallsicherheit erhöhen
- Für Hochverfügbarkeitssysteme
- Um Supercomputer zu bauen
- Variabel auf Lastschwankungen reagieren (IaaS/PaaS)
- Kann kosteneffizient sein

Vert.x Architektur

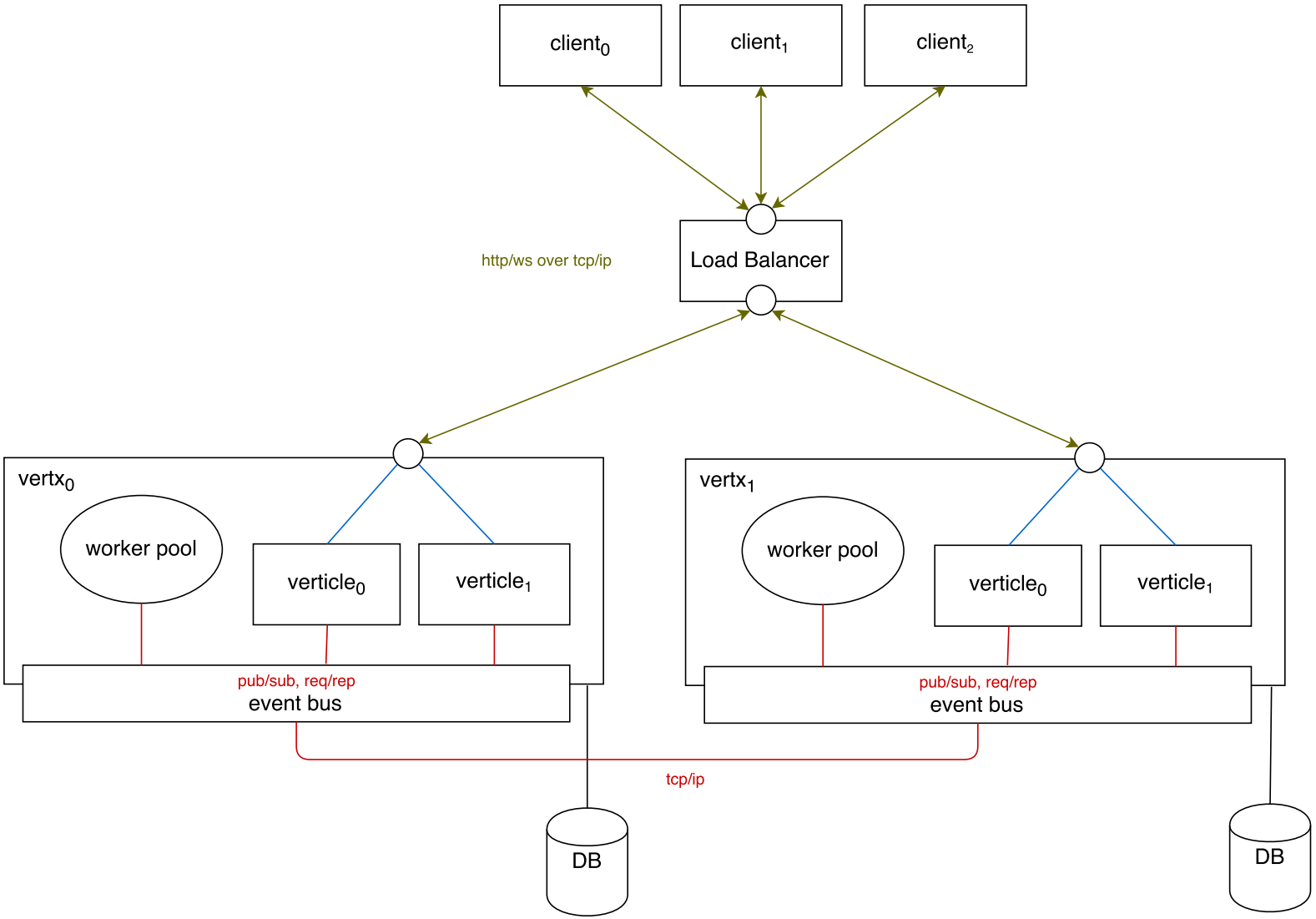


Clustering bei Vert.x

1. Optionen setzen
2. Vert.x asynchron starten
3. Profit

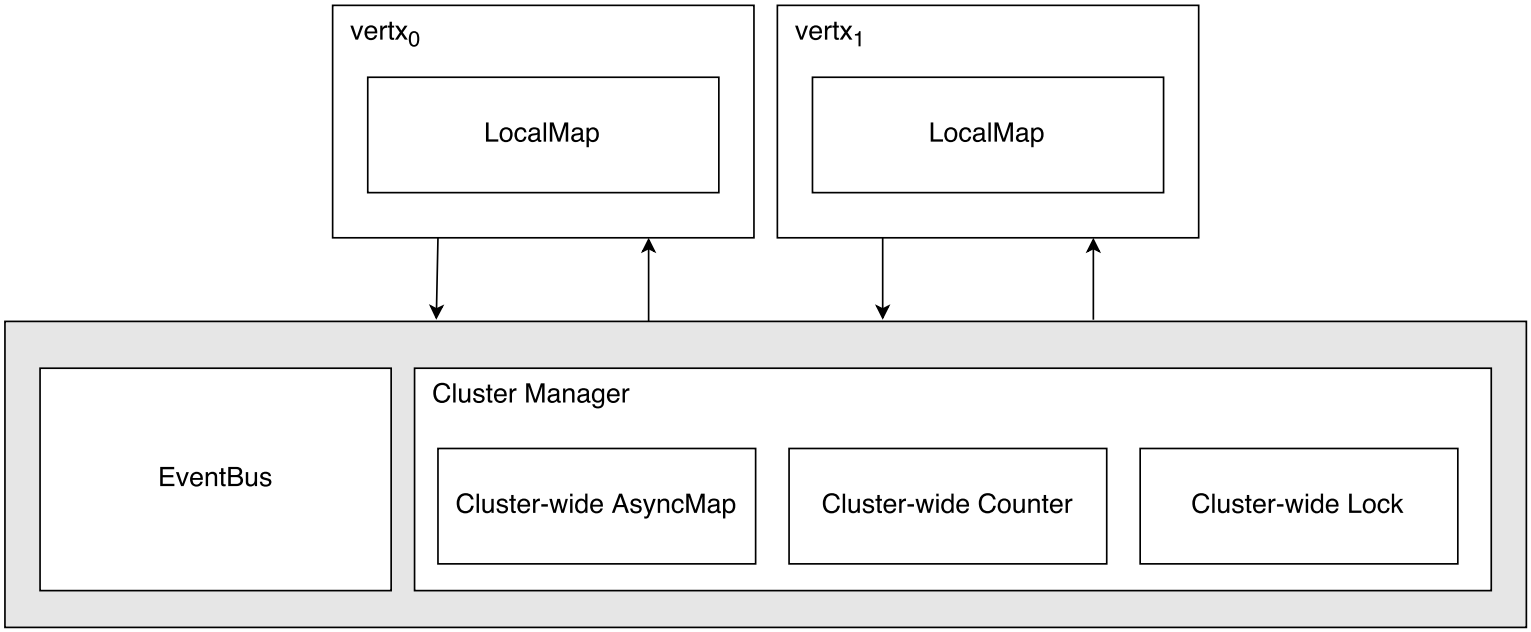
```
VertxOptions opts = new VertxOptions();
opts.setClusterPort(CLUSTER_PORT);
opts.setClusterHost(CLUSTER_HOST);

Vertx.clusteredVertx(opts, ar -> {
    final Vertx vertx = ar.result()
    // deploy verticles
});
```

Vert.x Cluster Beispiel-Architektur



Shared Data

1. LocalMap pro Instanz sichtbar
2. AsyncMap pro Cluster
3. Atomare Counter
4. Verteilte Locks



Local Shared Map

Synchrones Interface zum Austausch von unveränderlichen Daten zwischen verschiedenen Verticle-Instanzen

---

```
final SharedData sd = this.vertx.SharedData();
LocalMap<String, String> local = sd.getLocalMap("mapname");

local.put("foo", "bar");
local.get("foo"); // "bar"
```

Cluster Wide Map

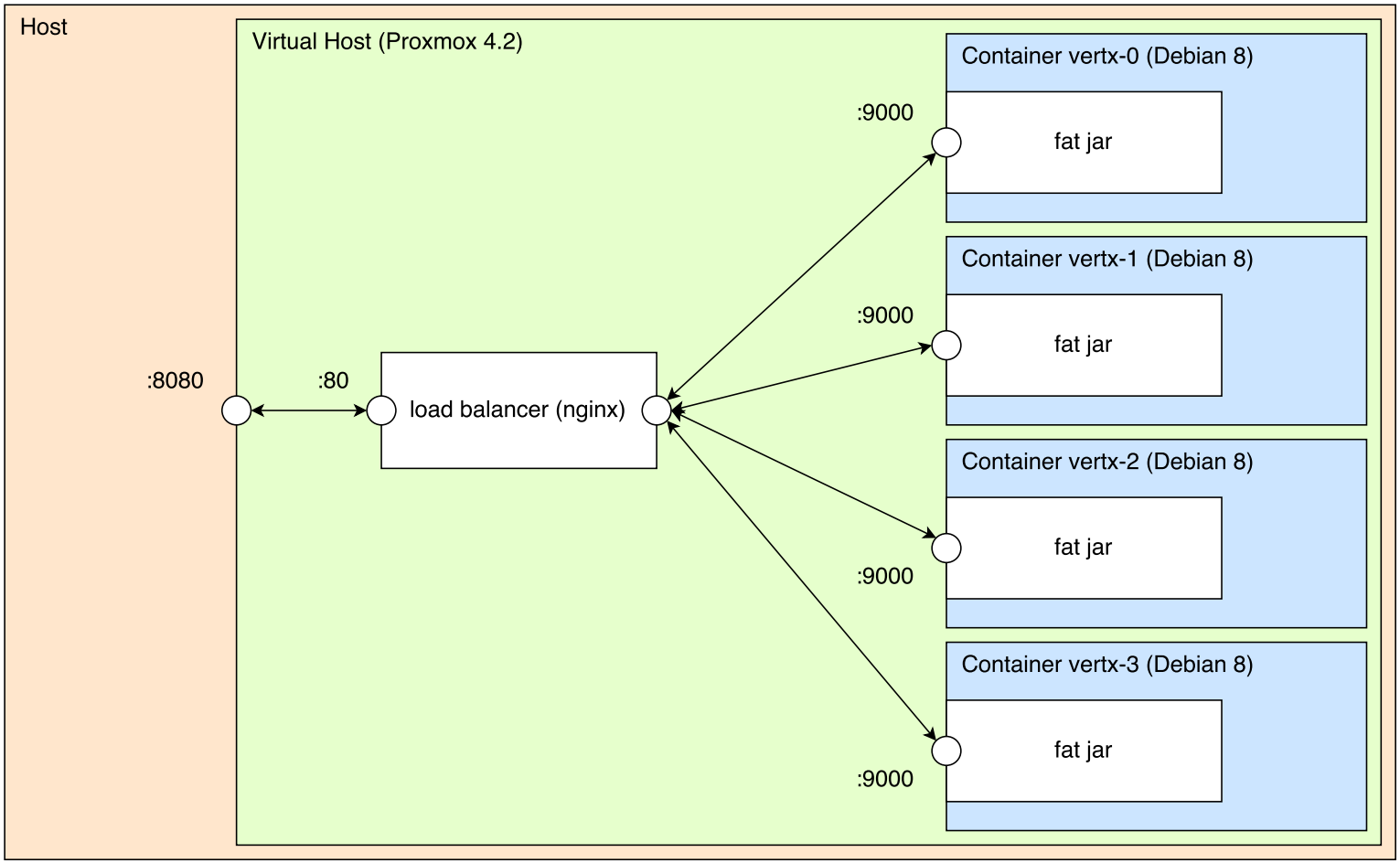
Asynchrones Interface um Daten quer über den gesamten Cluster auszutauschen. Kann zB. verwendet werden um Nutzersessions zu verwalten.

```
final SharedData sd = this.vertx.SharedData();
sd.<String, String>getClusterWideMap("name", ar -> {

    final AsyncMap<String, String> map = ar.result();
    map.put("foo", "bar", putResult -> {
        map.get("foo", getResult -> {
            String val = getResult.result(); // "bar" ?
        });
    });

});
```

Lock und Counter

```
final SharedData sd = this.vertx.sharedData();

sd.getCounter("name", ar -> {
    Counter counter = ar.result();
    long val = 1L;
    long exp = 4L;

    counter.get(cr -> {});
    counter.addAndGet(val, cr -> {});
    counter.getAndAdd(val, cr -> {});
    counter.incrementAndGet(cr -> {});
    counter.getAndIncrement(cr -> {});
    counter.decrementAndGet(cr -> {});
    counter.compareAndSet(exp, val, cr -> {});
});

// sd.getLockWithTimeout
sd.getLock("name", ar -> {
    Lock lock = ar.result();
    // do stuff
    lock.release();
});
```

Demo


Host
Virtual Host (Proxmox 4.2)
Container vertx-0 (Debian 8)
:9000
fat jar
Container vertx-1 (Debian 8)
:9000
fat jar
:8080
:80
load balancer (nginx)
Container vertx-2 (Debian 8)
fat jar
:9000
Container vertx-3 (Debian 8)
fat jar
:9000

Fazit

- Ob Clustering nötig ist, ist abhängig von der individuellen Architektur (SOA vielleicht, Microservices nein)
- Notwendig für Hochverfügbarkeit (wenn man das denn braucht)
- Einfaches Mittel zum ad-hoc skalieren, aber der Preis ist die Komplexität eines verteilten Systems
- Bei I/O lastigen Systemen ist Vert.x wohl selten der Flaschenhals
- Allgemein: Je weniger State im Service, umso weniger Kopfschmerz