Event driven Microservices with VERT.X & Kubernetes

Andy Moncsek
Senior Consultant

Andy.Moncsek@trivadis.com
Twitter: @AndyAHCP

Agenda

1. Why?
2. Event driven Microservices
3. VERT.X & Hazelcast
4. Docker & Kubernetes
5. Hazelcast & Kubernetes
6. Demo

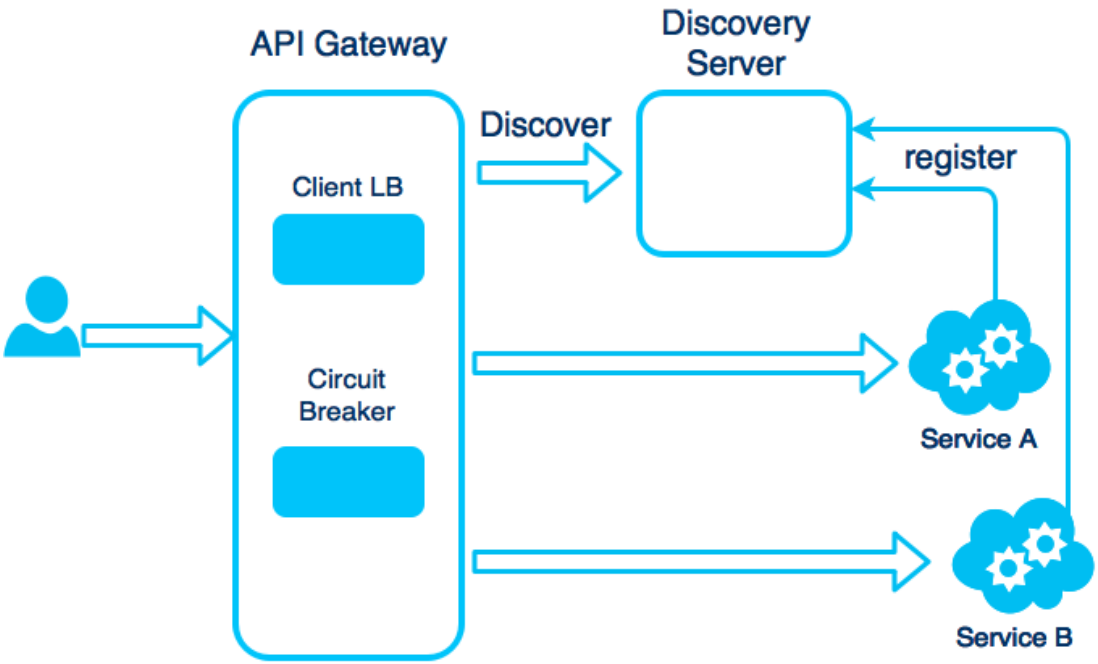
Why?

Why?

- Developing **one** Microservice is **fun**
  - Reduced scope
  - Choose your technology/language
  - Easy to deploy/test

Life could be so easy ;-)

Why?



Typical microservice architecture

Why?

- Developing **many** Microservices can be a **pain**
  - Complex deployment / orchestration / testing
  - Service registry & lookup → latency
  - DNS & Load balancer → complex

Why?

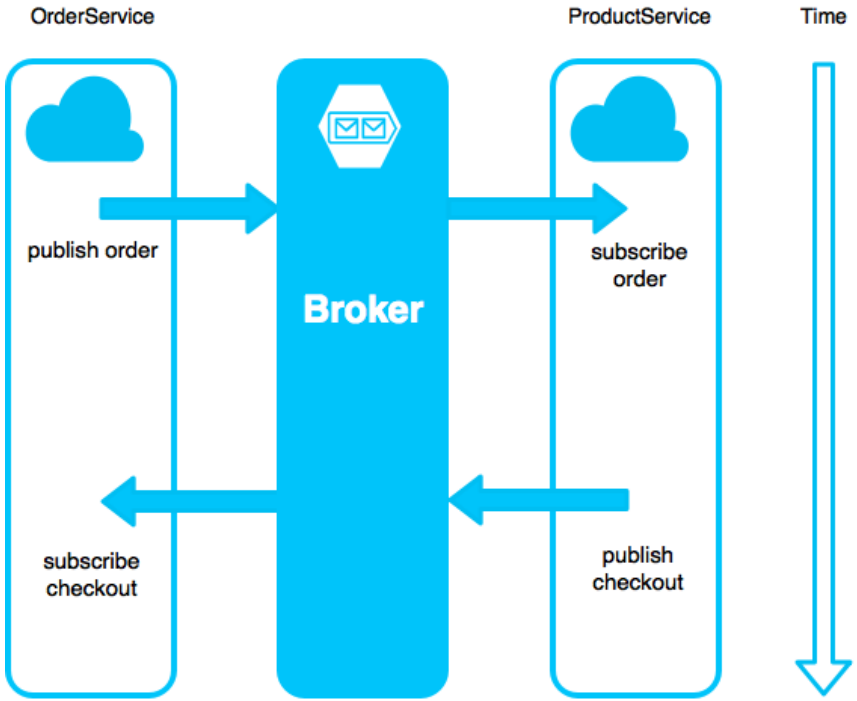
- Idea
  - Avoid Service registry & lookup
  - Decouple services → async + events
  - Fast re-/deploy
  - And more: resilient,…

Event driven Microservices

Event driven microservice

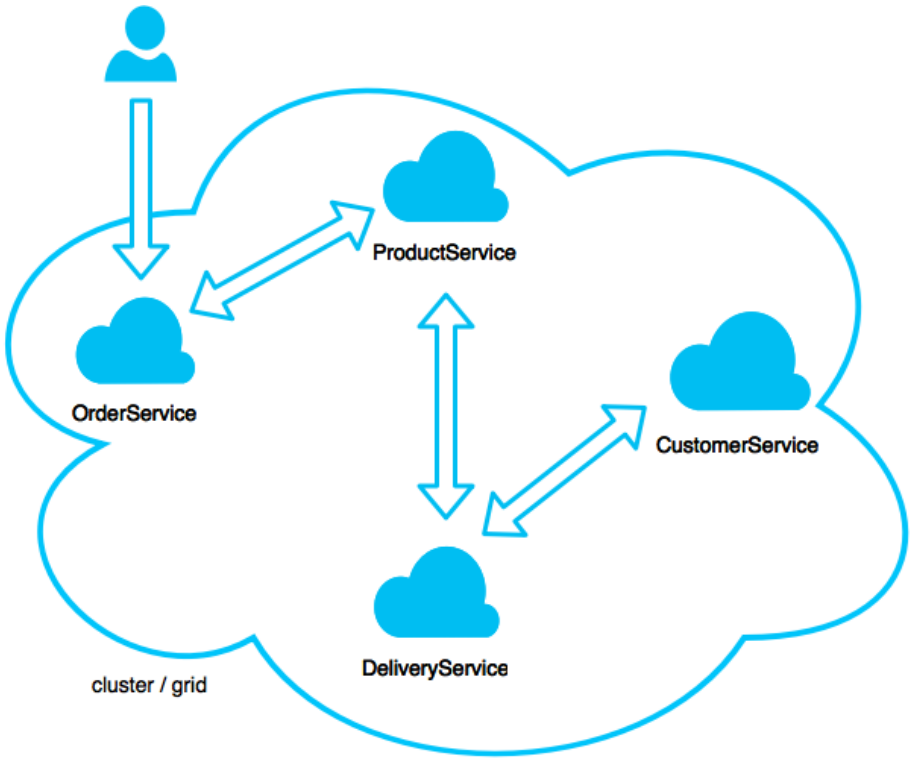
- **Asynchronously**
- **Any number of receivers**
- **Easy Load Balancing**
- **Pub/Sub + p2p**
- **No need for service discovery**

Event-driven (micro-)service



- Traditional way → Message Broker
- No direct request / response
- Broker → single point of failure

Event-driven microservice



- Using an in-memory data grid
- Automatic node lookup (services)
- Failover
- Shared data → replicated in cluster
- Pub / sub
- Request / response
- NO Service discovery

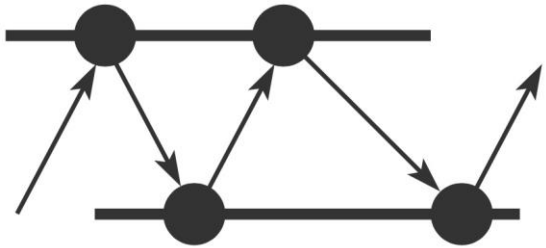
VERT.X & Hazelcast

What Vert.x is

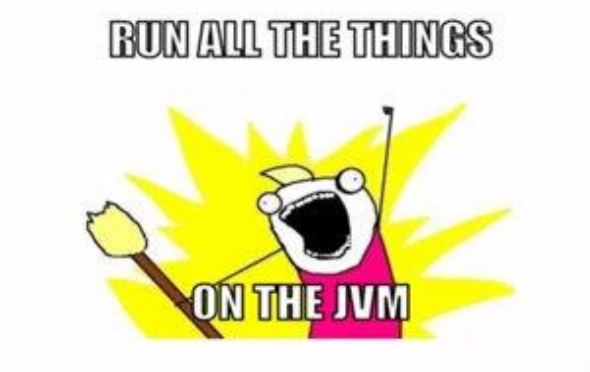
VERT.X is a tool-kit for building **reactive**

appications on the **JVM**

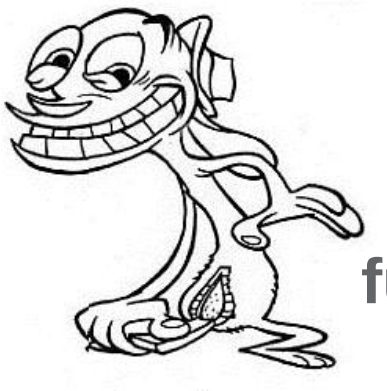
What Vert.x is



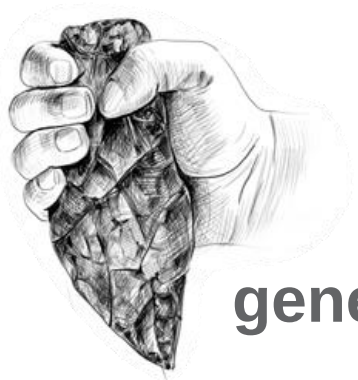
scale



polyglot



fun



general purpose

Verticles

- Smallest deployable unit
- Scalable
- Actor-like
- Access to Vertx instance

```
public class Service extends AbstractVerticle {

    public void start() {
            this.vertx...
    }

    public void stop() {

    }

}
```

Event Loop

- Multi-Reactor - event loop

  (N threads for N verticles on one instance)

- **Don’t block the Event-Loop!**

- Don’t call us, we call you

```
server.requestHandler(request -> {

  request.response().end(„Hello!");

});
```

Event Bus (EB)

- Nervous system of Vert.x
- Communicate
- Distributed p2p
- Pub/sub
- Request-response

```
EventBus eb = vertx.eventBus();

eb.send(“the.addr", “hello", ar -> {
  log.info(ar.result().body());
});

eb.consumer(“the.addr", message -> {
  message.reply(“don‘t know");
});
```

What is Hazelcast ?

is an In-Memory Data Grid

(and the default cluster provider of Vert.X)

Hazelcast

- Open Source & Enterprise Edition
- Features:
  - Distributed Caching & Compute
  - **Clustering**
  - Storage
  - Management & Security

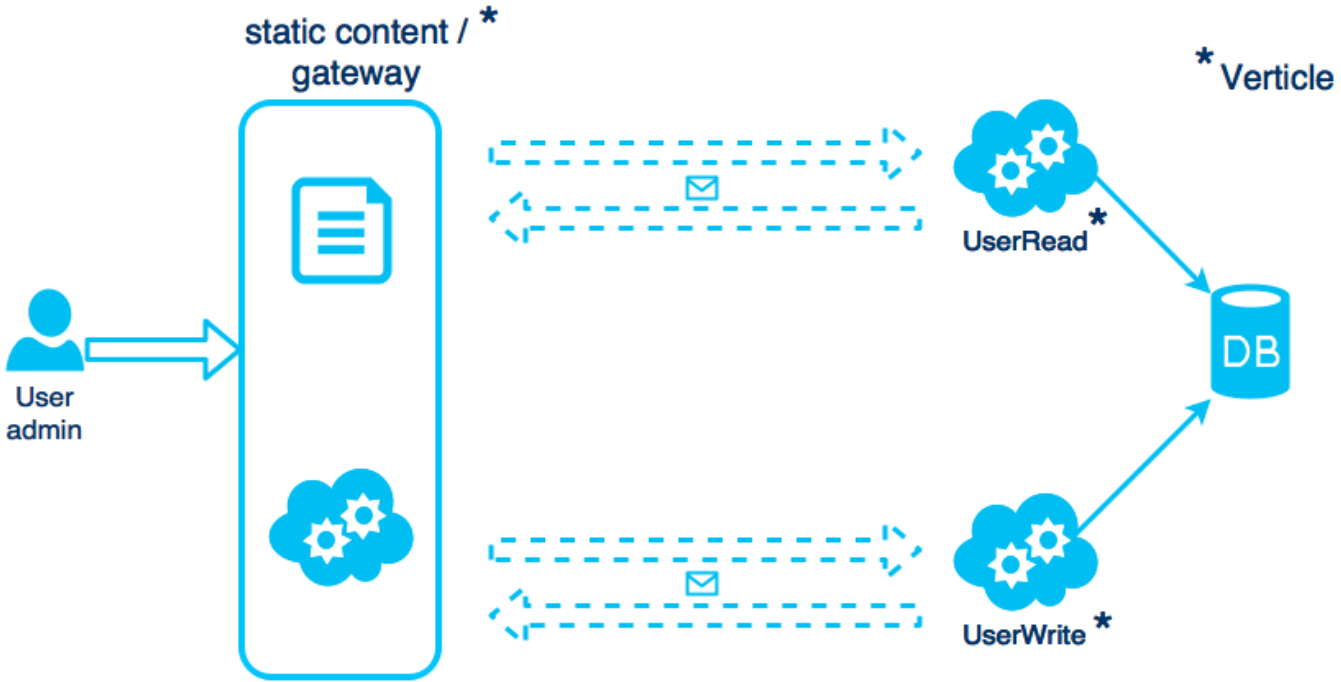
Hazelcast Clustering

- Discover Cluster Members using:
  - Multicast
  - TCP
  - EC2 Cloud
  - jclouds
- Plugins:
  - AWS
  - Azure
  - Kubernetes
  - Etcd
  - …

Run Vert.x inclustered mode

- Vert.x default config: multicast
- „**java –jar service.jar –cluster**“

Demo



Demo

https://github.com/amoAHCP/kube_vertx_demo

Docker & Kubernetes

Docker

isolation + automation + inherence + repository + versioning

Docker alone doesn‘t make you happy ;-)

Kubernetes

Scale

HA distributed containers

Google starts

>20billion apps/week

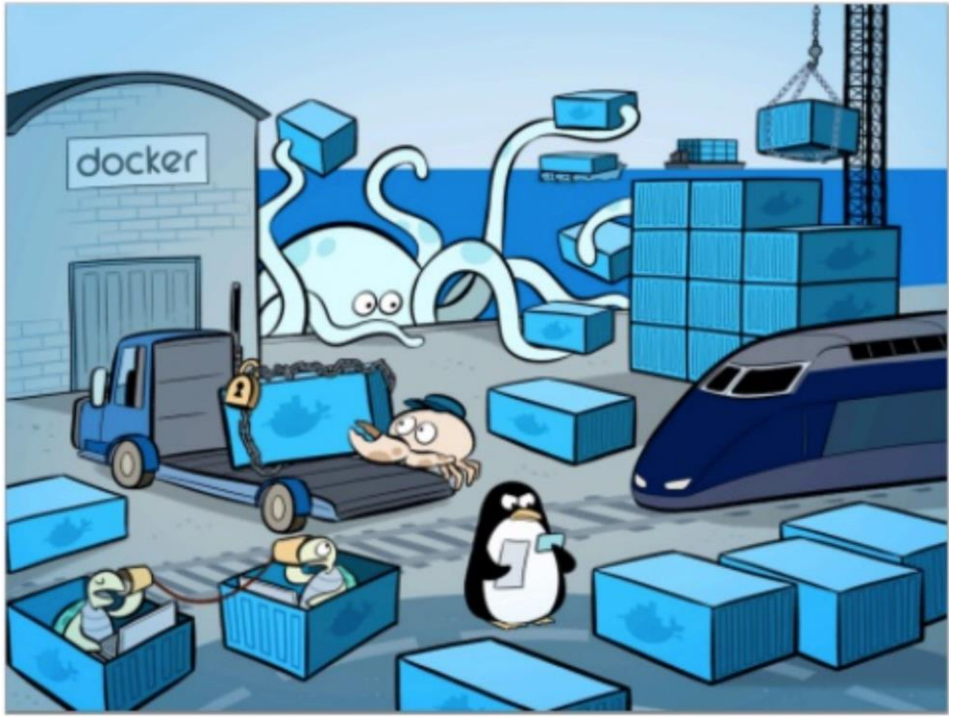
Gmail, Web Search, Maps…

Kubernetes

But

It is not trivial

Kubernetes



Service oriented

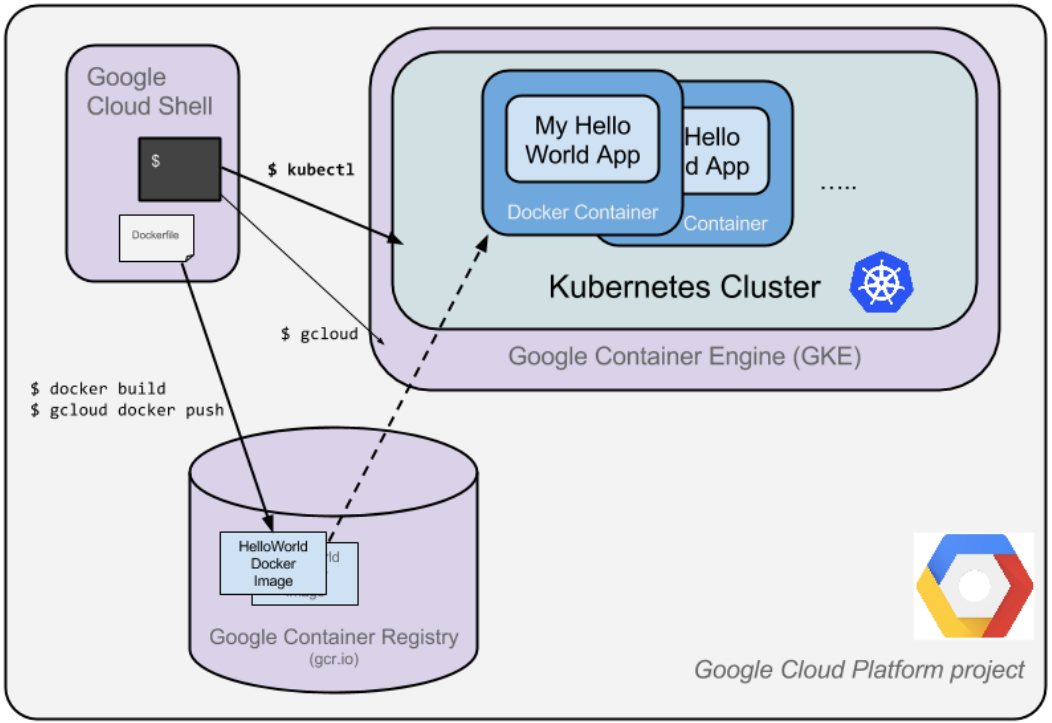
Services -> machines

“Let users manage **applications**, not machines”

Kubernetes

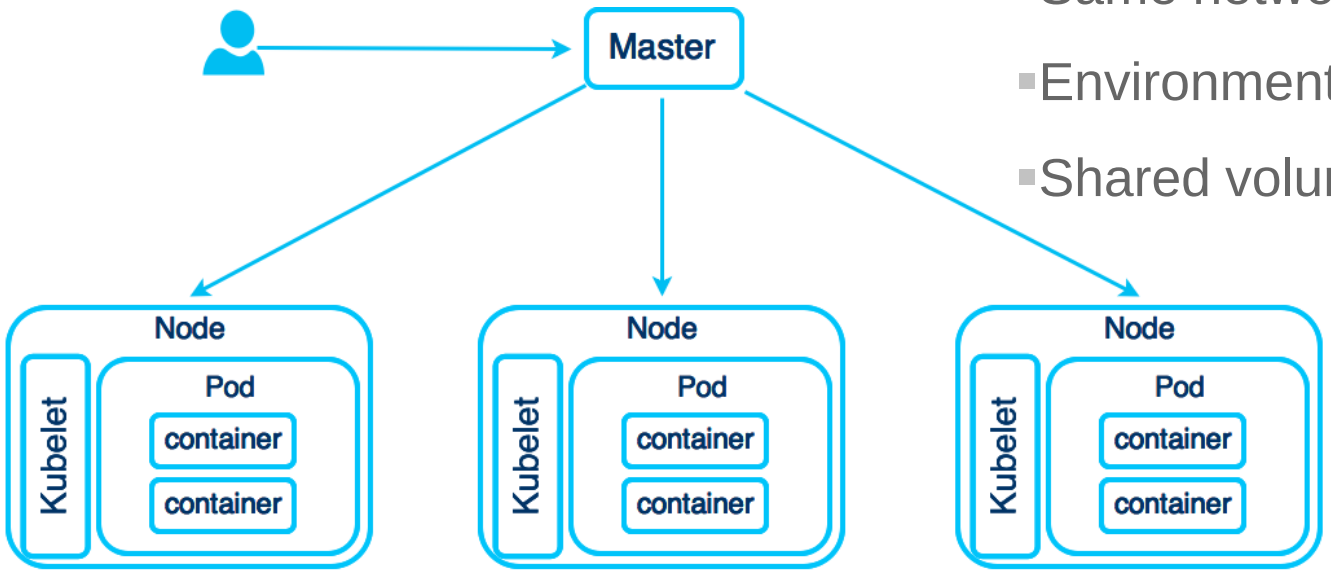
- Platform for hosting containers in clusteres
- Docker containers across multiple hosts
- Provides: monitoring, grouping, load balancing, auto-healing, scaling
- Contributors: Google, Redhat, Microsoft, ...

Kubernetes – Key concepts



1. Push Docker image to Registry
2. Create service
3. Create Controller
4. (Scale Pods)

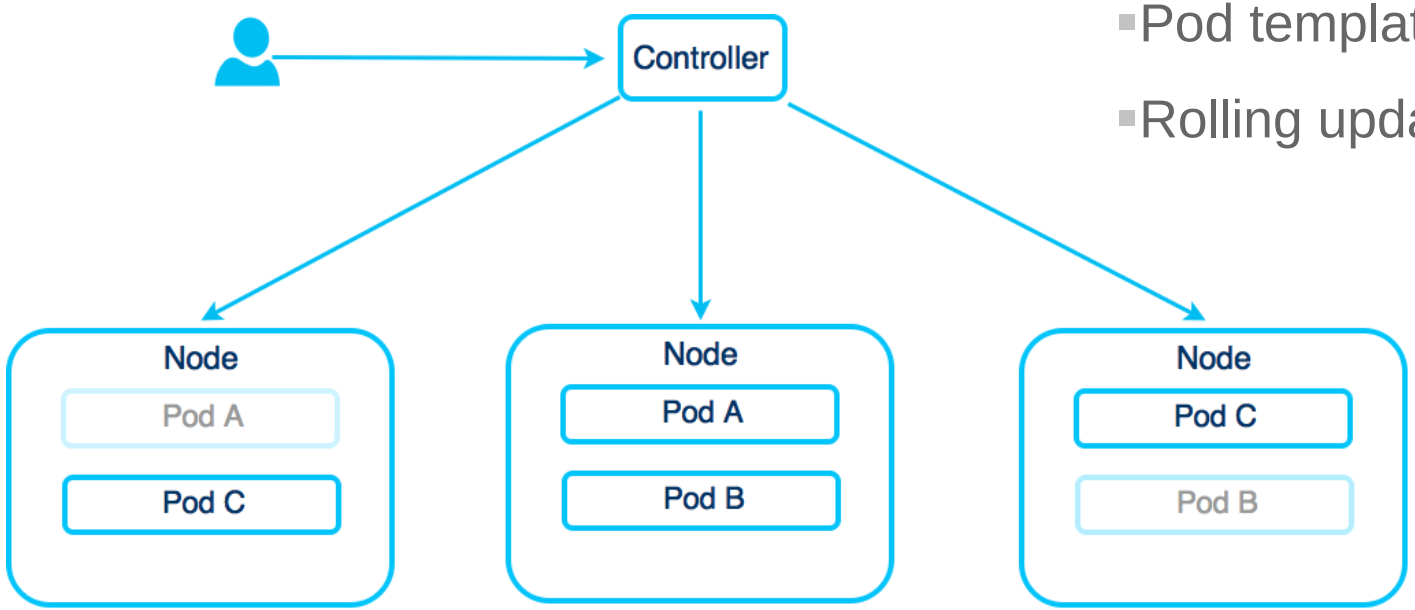
Kubernetes Pods



- Group of containers
- Same network / namespace / IP
- Environmental variables
- Shared volumes

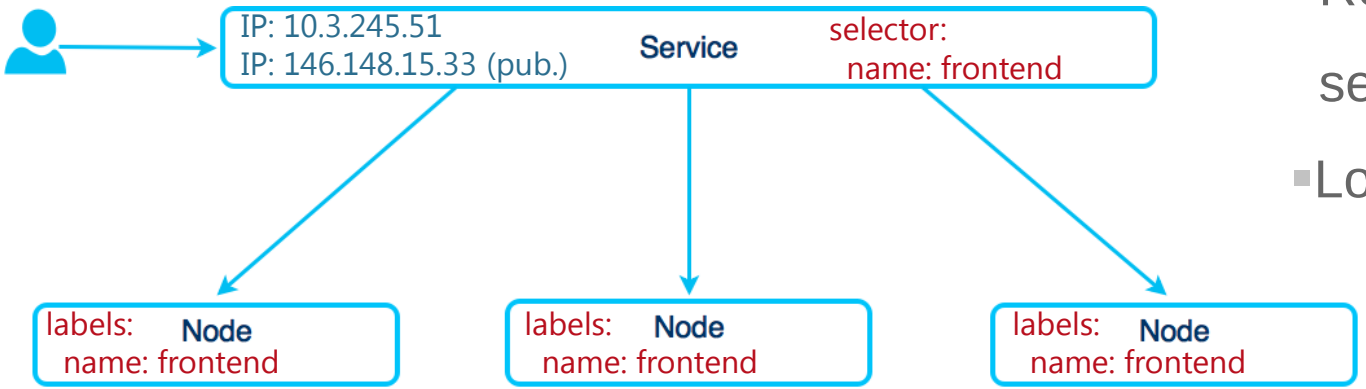
Kubernetes Controller

- Ensure X pods are running
- Pod templates
- Rolling update



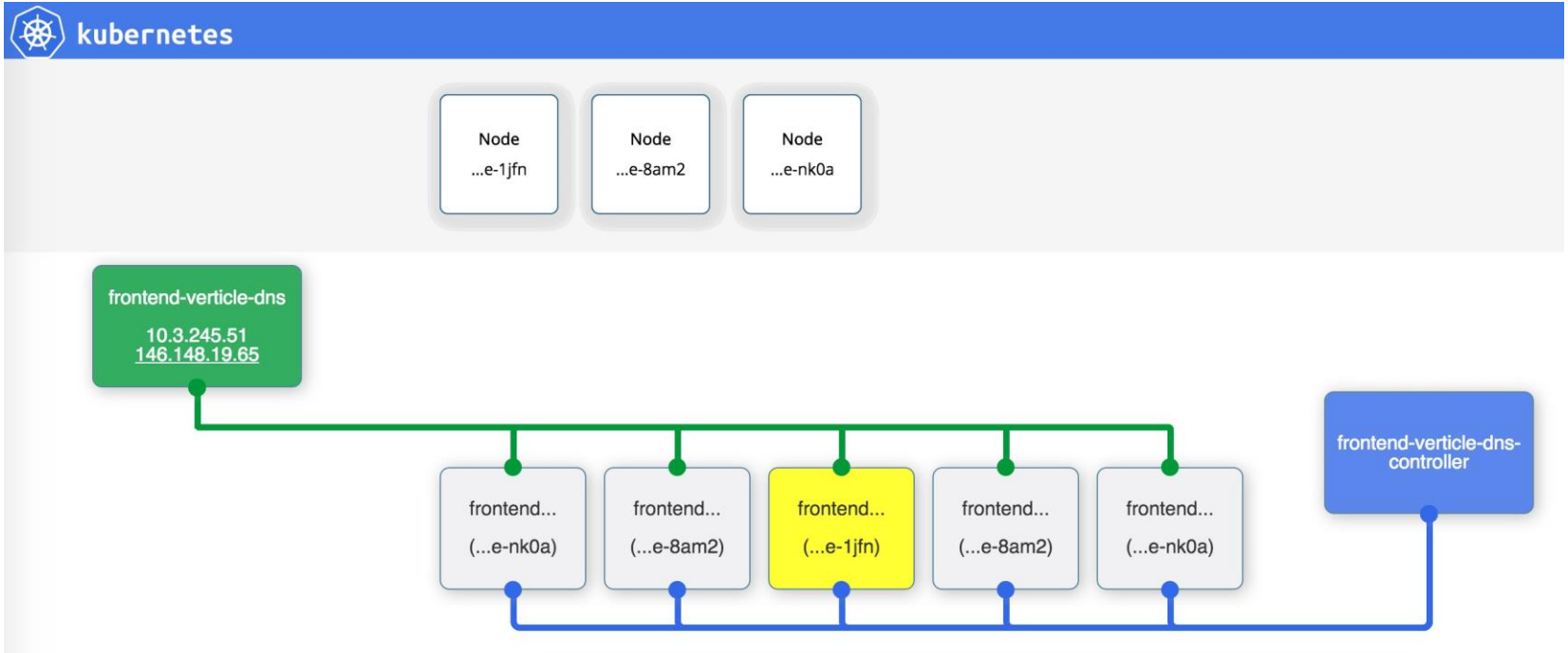
Kubernetes Services

```
ports:
  -port: 80
    targetPort: 8181
```



- Pod discovery
- IP per service
- Route to pods selected with labels
- Load balancer

Kubernetes – all together



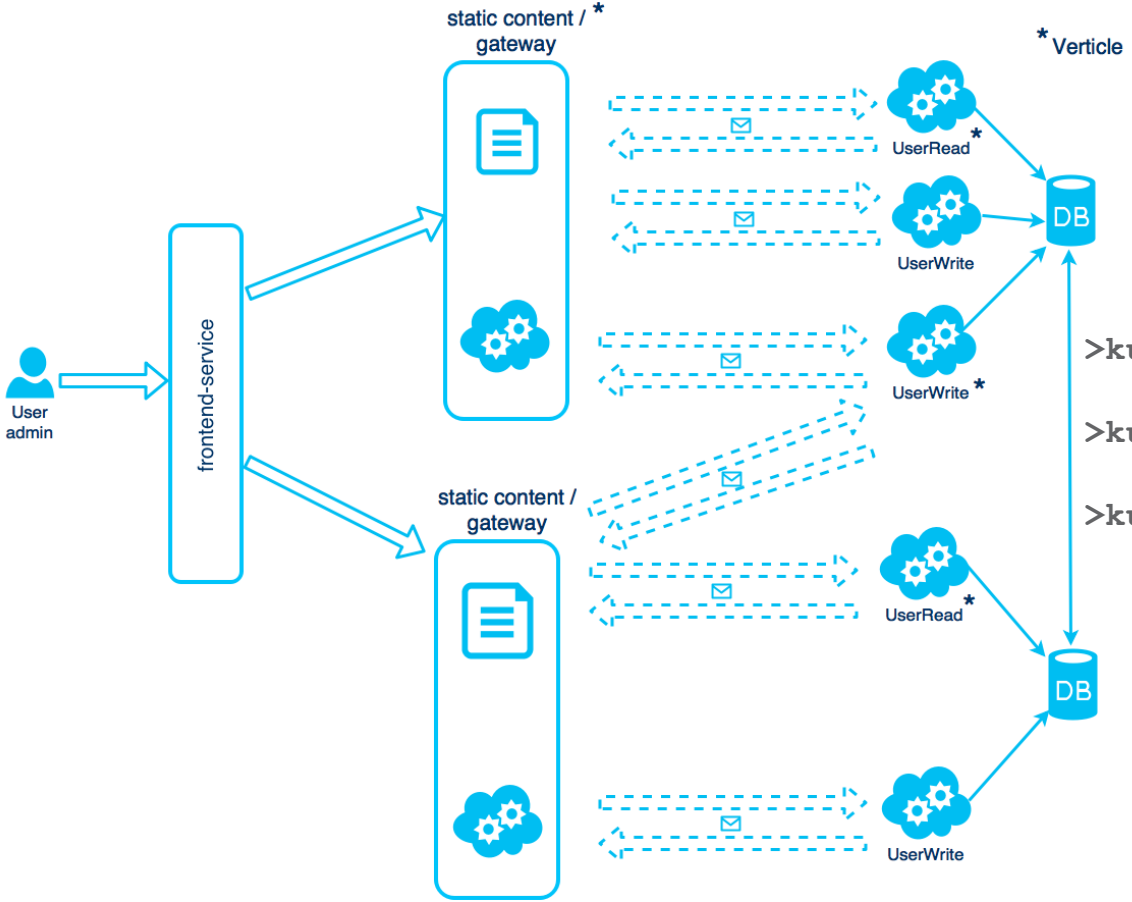
https://github.com/brendandburns/gcp-live-k8s-visualizer

Hazelcast & Kubernetes

Hazelcast & Kubernetes → What we want

- **Case 1:** Lookup all nodes (Verticles) managed by controller
  - 1 – N pods
    - 1 – X container
- **Case 2:** Lookup all nodes in a cluster
  - N services per cluster
    - Routing to X pods

Hazelcast & Kubernetes → What we get




```
>kubectl scale rc frontend --replicas=5

>kubectl scale rc read --replicas=10

>kubectl scale rc write --replicas=2
```

Hazelcast & Kubernetes → What we get

- Easy scale- up/down of service instances
- Failover provided by Kubernetes and Hazelcast
- Distributed Event-Bus → each Verticle instance
- Replicated shared data

Hazelcast node discovery in Kubernetes

- Discover Cluster Members using:
  - ~~Multicast~~
  - TCP
  - ~~EC2 Cloud~~
  - ~~jclouds~~
- Plugins:
  - ~~AWS~~
  - ~~Azure~~
  - Kubernetes
  - ~~Etcd~~
  - …

Hazelcast node discovery in Kubernetes

- Hazelcast discovery plugin for Kubernetes
  - Option 1: Kubernetes **REST API**
    - Allows discovery by **service name**
  - Option 2: **DNS Lookup**
  - Enable custom cluster.xml in Vert.X → put to classpath
  - https://github.com/noctarius/hazelcast-kubernetes-discovery

Hazelcast node discovery in Kubernetes (2)

- Vertx-service-discovery project (similar to Hazelcast Option 1)
  - Option: Kubernetes **REST API**
    - Allows discovery by **label** OR by **namespace**
  - Enable custom cluster.xml in Vert.X → put to classpath
  - https://github.com/vert-x3/vertx-service-discovery

Hazelcast node discovery in Kubernetes

```
<hazelcast>
  <properties>
    <!-- at the moment the discovery needs to be activated explicitly -->
    <property name="hazelcast.rest.enabled">true</property>
  </properties>
...
  <discovery-strategy enabled="true"
                      class="....HazelcastKubernetesDiscoveryStrategy">

    <properties>
      <property name="service-name">frontend-verticle</property>
      <property name="namespace">default</property>
    </properties>
  </discovery-strategy>

</hazelcast>
```

Hazelcast.xml / Cluster.xml

- REST API

Missing in official Doc. !

- your Service name
  - master Service!
  - needs a Kubernetes Service!

Vert.x / Hazelcast node discovery in Kubernetes

```
<hazelcast>
  <properties>
    <!-- at the moment the discovery needs to be activated explicitly -->
    <property name="hazelcast.rest.enabled">true</property>
  </properties>
...
  <discovery-strategy enabled="true"
                      class="....HazelcastKubernetesDiscoveryStrategy">

    <properties>
      <property name="service-label-name">cluster1</property>
      <property name="service-label-value">true</property>
      <property name="namespace">default</property>
    </properties>
  </discovery-strategy>

</hazelcast>
```

- REST API

Missing in official Doc. !

- Partition your cluster with labels
- OR find all in namespace

Hazelcast.xml / Cluster.xml

Hazelcast node discovery in Kubernetes

```
<hazelcast>
   <properties>
      <!-- at the moment the discovery needs to be activated explicitly -->
      <property name="hazelcast.discovery.enabled">true</property>
   </properties>
...
   <discovery-strategy enabled="true"
                       class="....HazelcastKubernetesDiscoveryStrategy">

   <properties>
      <property name="service-dns">cluster.local</property>
   </properties>
   </discovery-strategy>

</hazelcast>
```

Hazelcast.xml / Cluster.xml

- DNS Lookup
- DNS must be enabled in Kubernetes (kube2sky)!

your Service name
OR
DNS_DOMAIN name
(cluster.local → find all containers)

Hazelcast node discovery in Kubernetes

- Status of Vert.x / Hazelcast discovery plugins for Kubernetes
  - ✓ Vert.x / Hazelcast REST API implementation is more flexible
    - ✓ Available since Vert.x 3.3.0
    - ✓ Code can be (technically) merged to Hazelcast plugin
  - ✓ Hazelcast DNS API has a bug in PORT discovery
    - ✓ workaround: https://github.com/amoAHCP/hazelcast-kubernetes-discovery

Demo

https://github.com/amoAHCP/kube_vertx_demo/tree/dns-resolving

Try this at home

- Test Kubernetes in Docker
  - https://github.com/vyshane/kid
  - Many other Docker-compose based solutions available
  - Limitations: no public IP & access to public service
  - Workaround:
    - `kubectl get -o yaml service/myService | grep nodePort`
    - `wget http://$(docker-machine ip):nodePort`

Conclusion

- Vert.x is lightweight, fast and easy to use
  - typical web-app, microservices, aggregator / bridge, security, integration, stream-/event-processing

- Kubernetes :
  - Functional improvement over clustered infrastructure
  - easy deploy, update and scale of applications

Thank you

Andy Moncsek
Senior Consultant

Andy.Moncsek@trivadis.com
Twitter: @AndyAHCP

trivadis
makes IT easier.