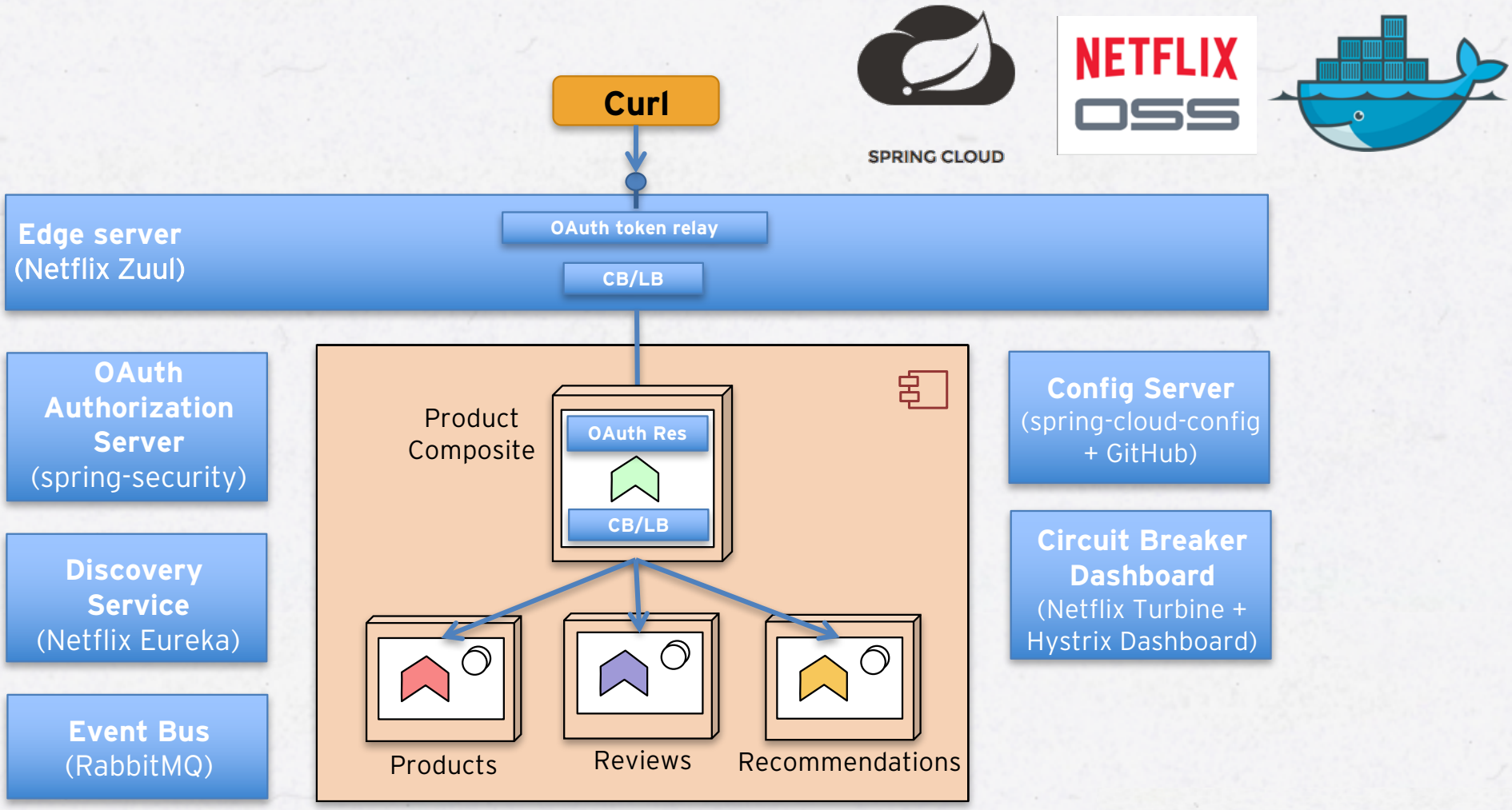
CADEC 2017 - EXPERIENCES FROM USING DISCOVERY SERVICES IN A MICROSERVICE LANDSCAPE

MAGNUS LARSSON

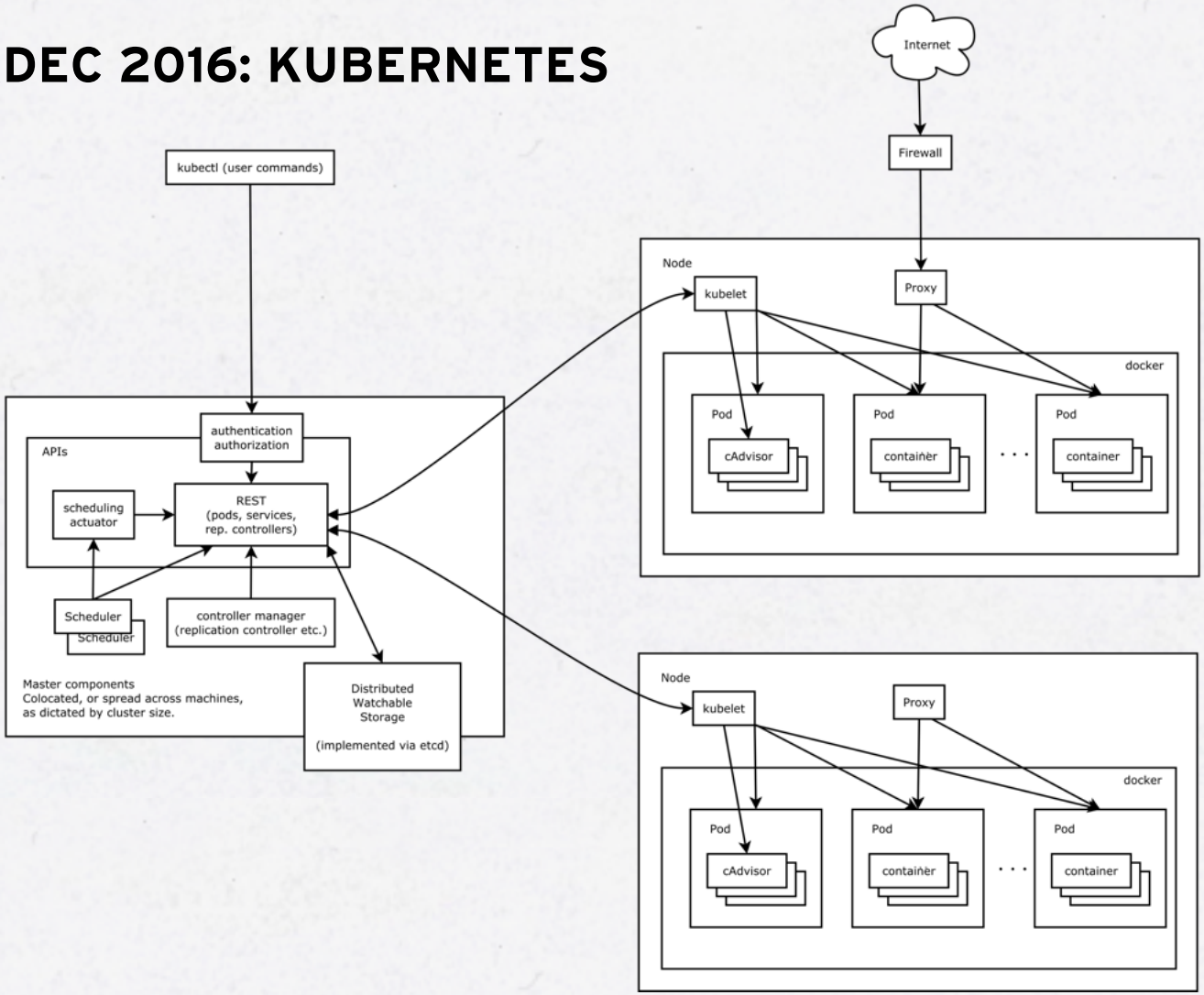
2017-01-25 | CALLISTAENTERPRISE.SE

CALLISTA
— ENTERPRISE —

FROM CADEC 2016: MICROSERVICES WITH SPRING CLOUD AND NETFLIX OSS



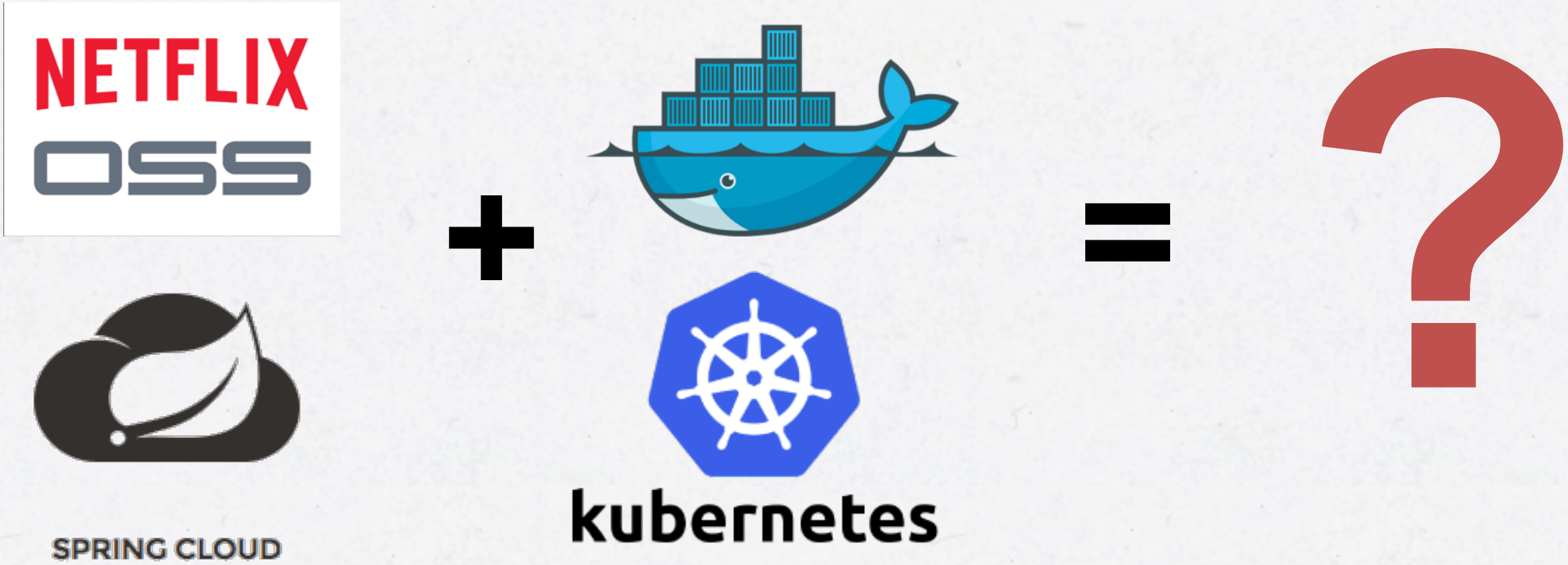
FROM CADEC 2016: KUBERNETES



kubernetes

Source: https://raw.githubusercontent.com/kubernetes/kubernetes/master/docs/design/architecture.png

POST CADEC 2016 QUESTION IN MY HEAD

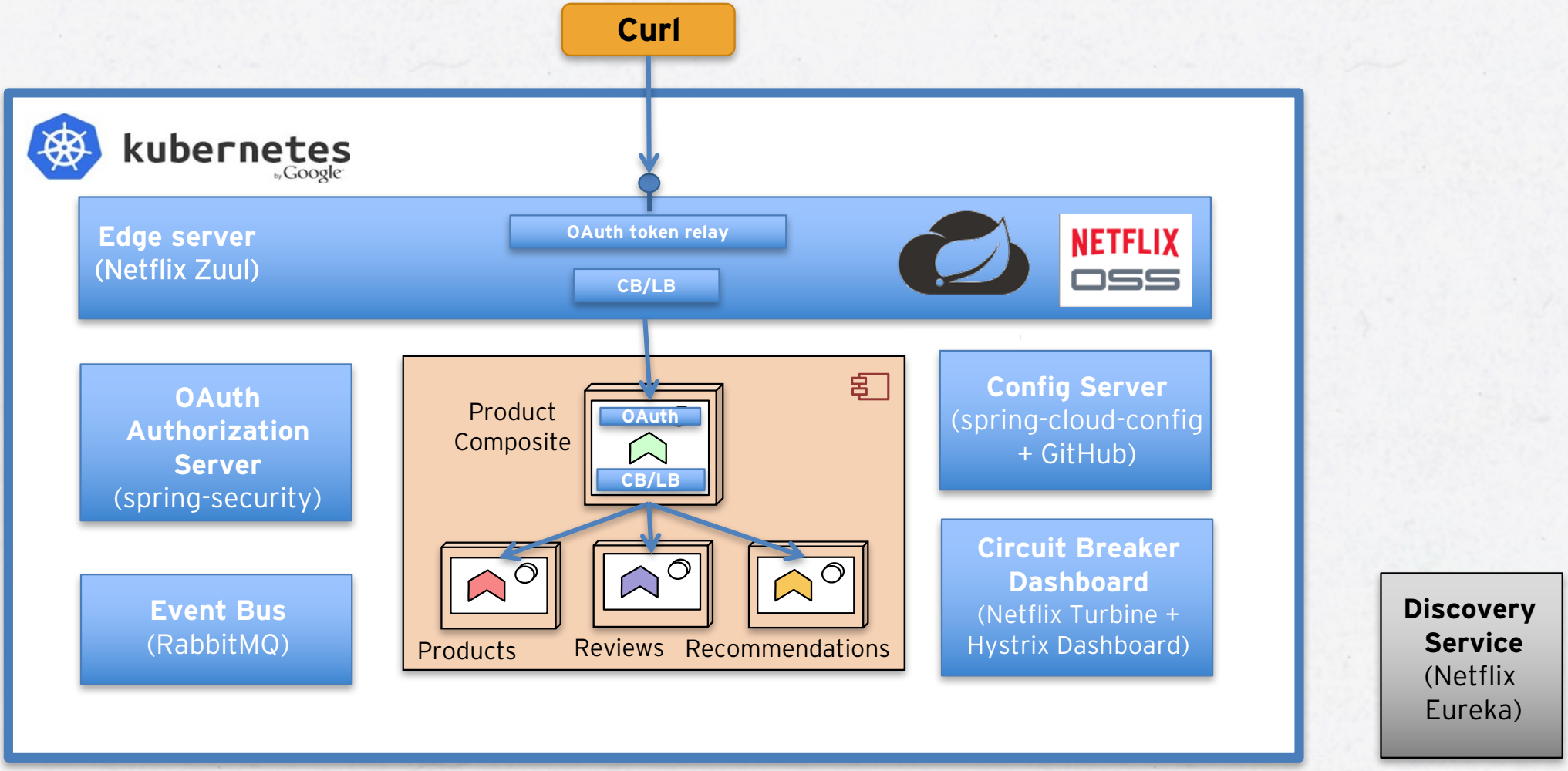


POST CADEC 2016 QUESTION IN MY HEAD

- Spring Cloud and Netflix OSS provides
  - Service discovery
  - Centralized configuration
  - Edge server
  - Circuit-breaker
  - OAuth based API security
  - Distributed tracing
  - Event-bus
  - …

- Kubernetes provides container orchestration capabilities
  - Cluster management
  - Declare a desired state
  - Service discovery
  - Self healing
  - Rolling upgrades
  - Automated scaling
  - …

SPRING CLOUD, NETFLIX OSS AND KUBERNETES



WHAT ABOUT OTHER CONTAINER ORCHESTRATION TOOLS?

- Docker Swarm mode (since v1.12)
- Amazon EC2 Container Services (ECS)
- Apache Mesos (DC/OS)
- Hashicorp Nomad

WHAT ABOUT MICROSERVICES WITHOUT CONTAINERS?

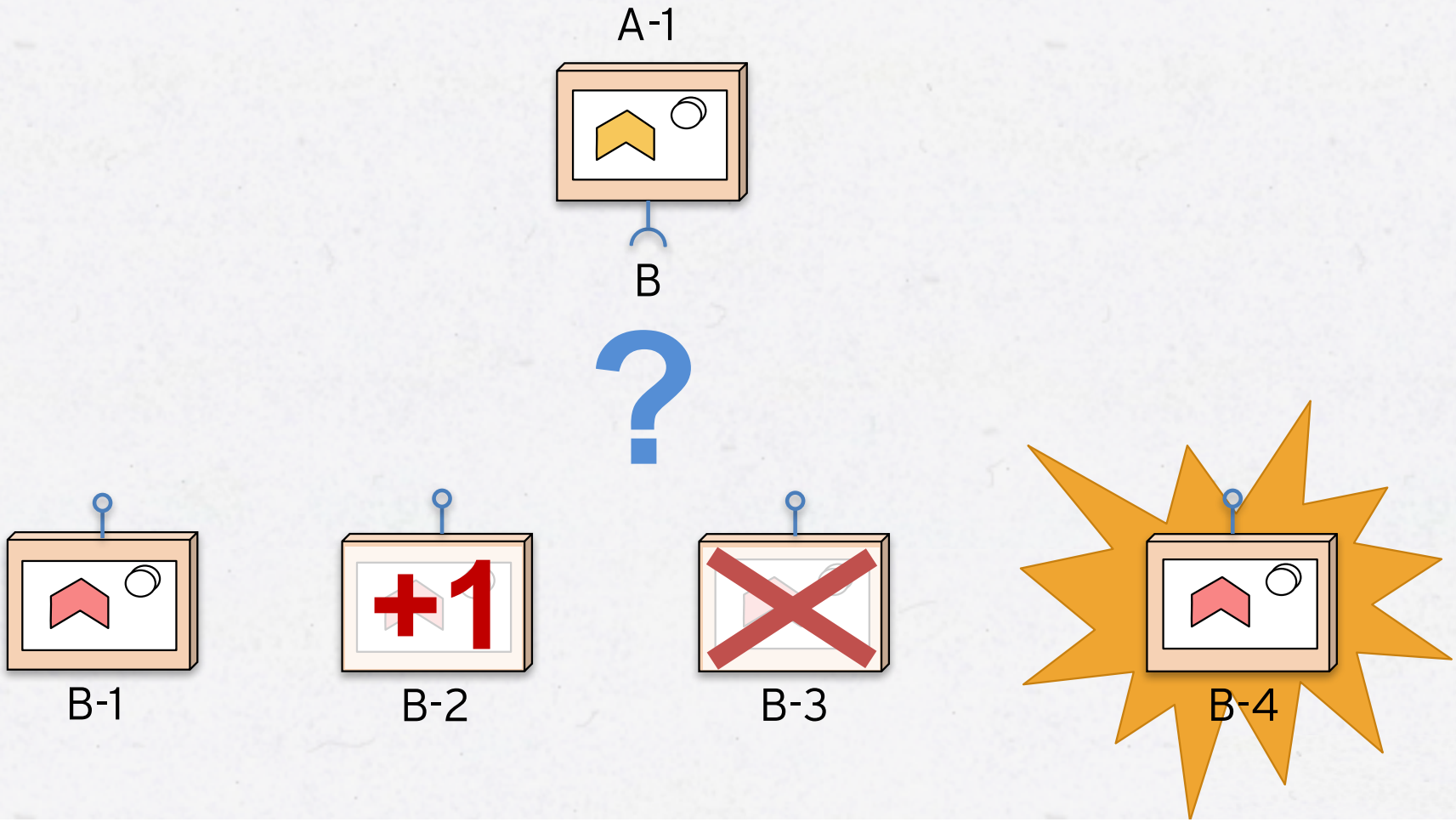
- Can we use Netflix Eureka as Service Discovery when containers are not used?

MY PERSONAL QUEST FOR 2016

- Write once, deploy “everywhere”?
  - Only allow configuration changes!
- Selected platforms (based on customer projects):
  - Without containers using ***Netflix Eureka*** as Service Discovery
  - Container orchestration tools (built in Service Discovery):
    - » *Kubernetes*
    - » *Docker Swarm mode*
    - » *Amazon ECS*
- **Main challenge**
  - How to write code that works with the various Service Discovery implementations?

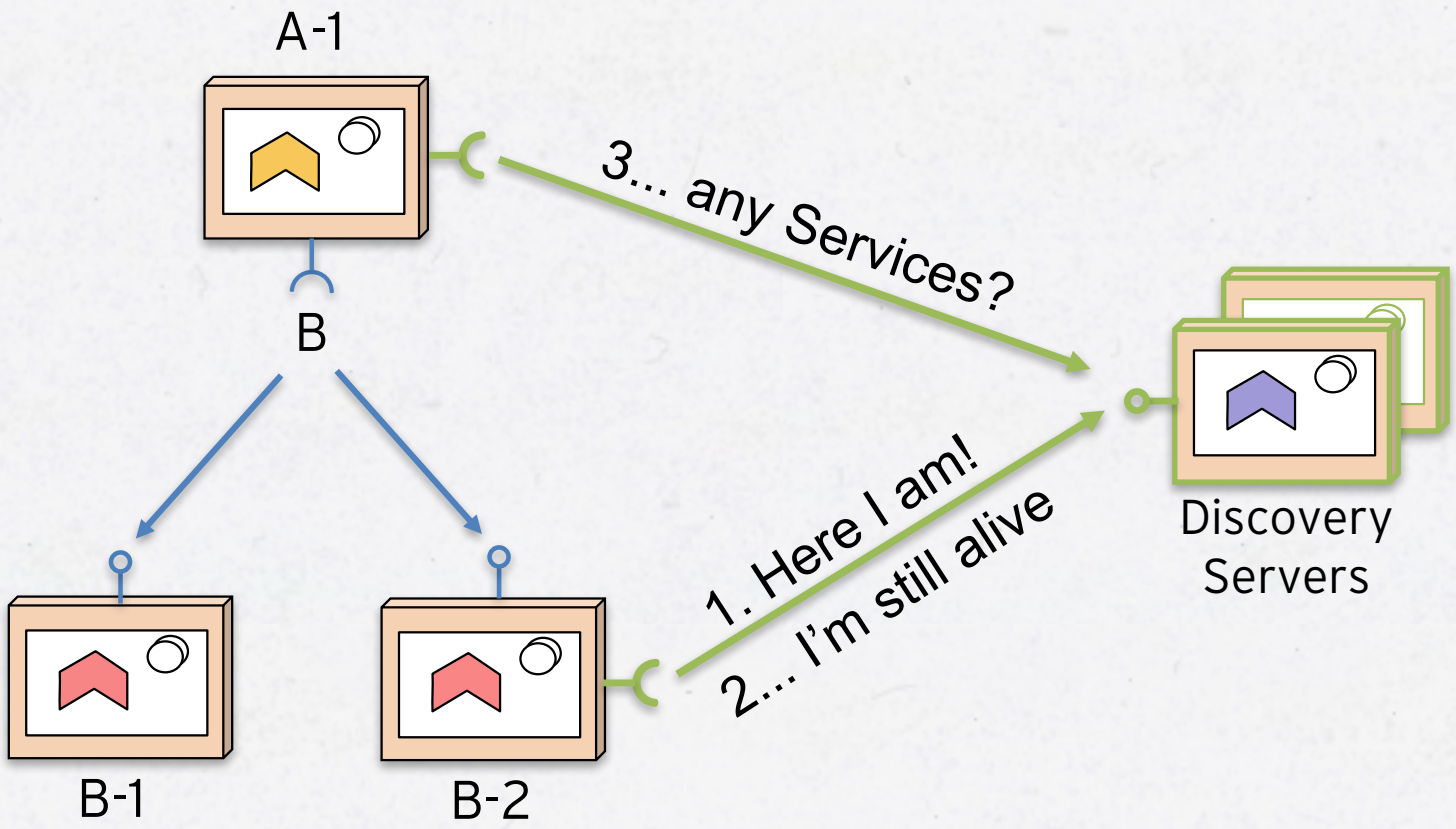
WHAT IS SERVICE DISCOVERY?

Keeping track of many small moving parts…



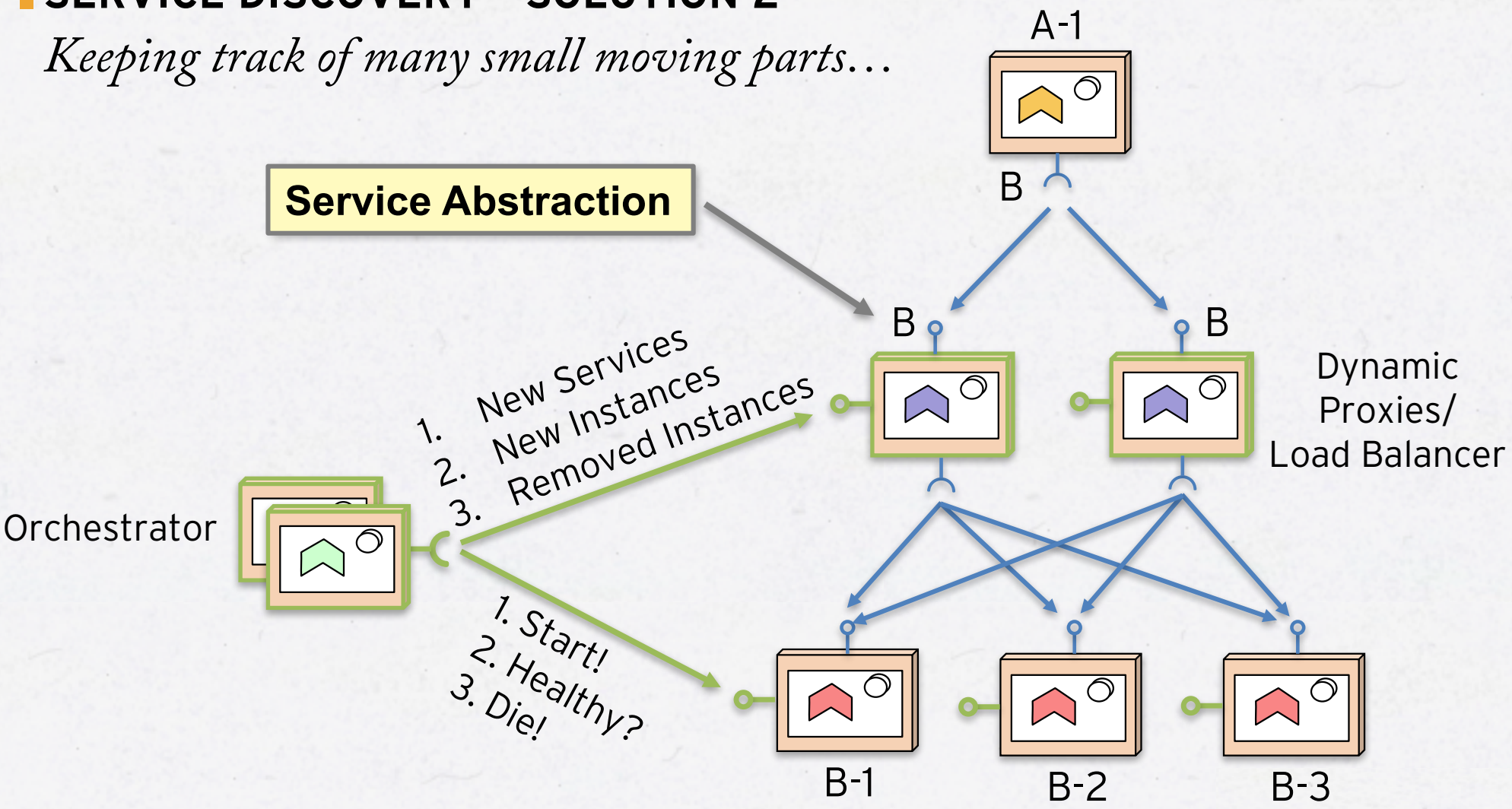
SERVICE DISCOVERY - SOLUTION 1

Keeping track of many small moving parts…



SERVICE DISCOVERY - SOLUTION 2

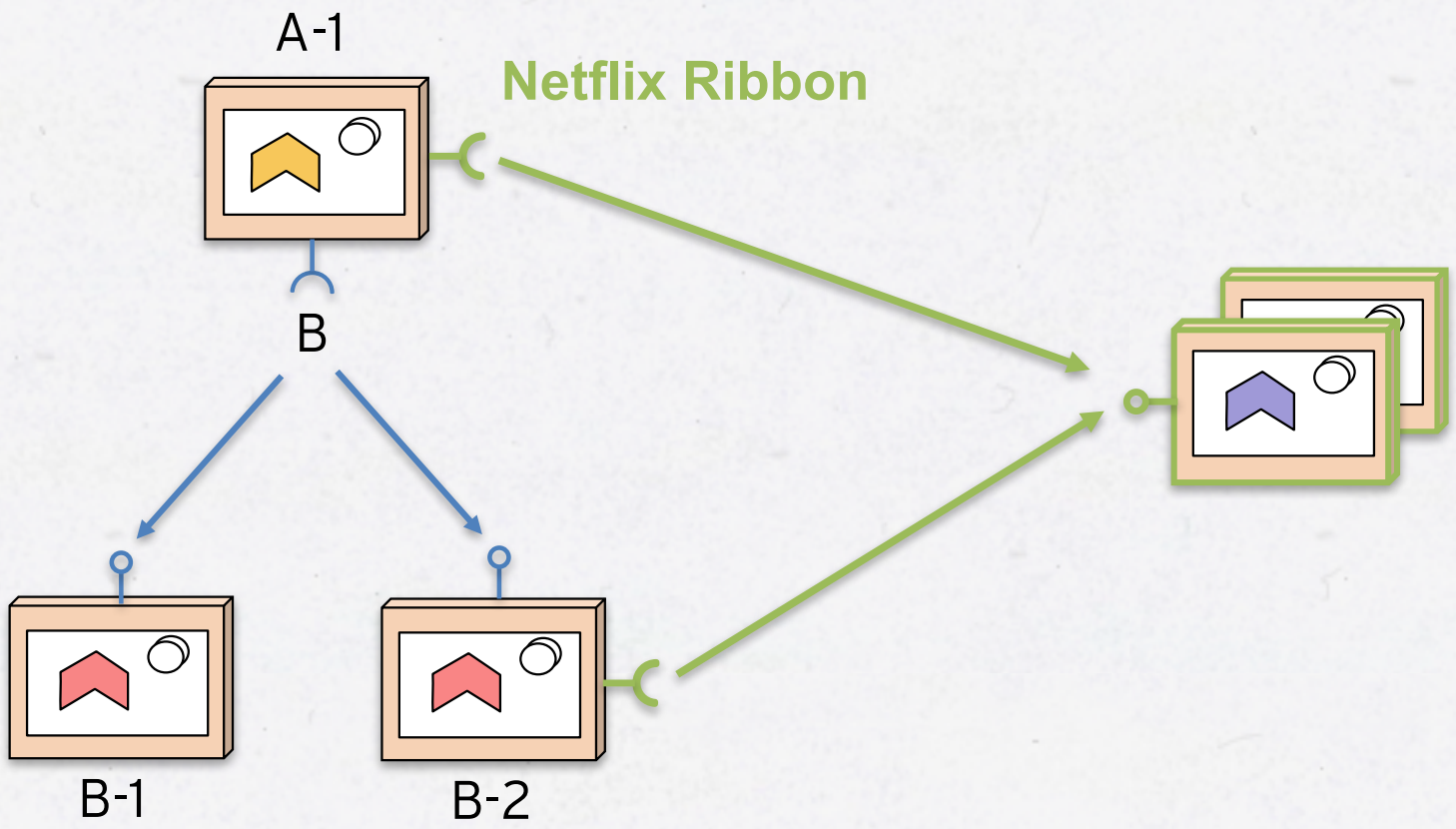
Keeping track of many small moving parts…



TIMELINE, MAJOR IMPROVEMENTS DURING 2016...

- **2016-01-27**: Cadec 2016
  - Can't find a common abstraction that works over Spring Cloud/Netflix OSS, Amazon ECS, Kubernetes and Docker Swarm!
- **2016-02-04**: Docker 1.10
  - Built in dynamic DNS server
- **2016-07-28**: Docker 1.12
  - Docker Swarm mode with a Service concept similar to Kubernetes
- **2016-08-11**: Amazon Application Load Balancer
  - Extends Amazon ELB to work with microservices in Amazon ECS
- **2017-01-25**: Cadec 2017
  - Looks better now, let's try it out!

EXAMPLE #1 - SPRING CLOUD AND NETFLIX OSS (NO CONTAINERS)



Discovery Servers

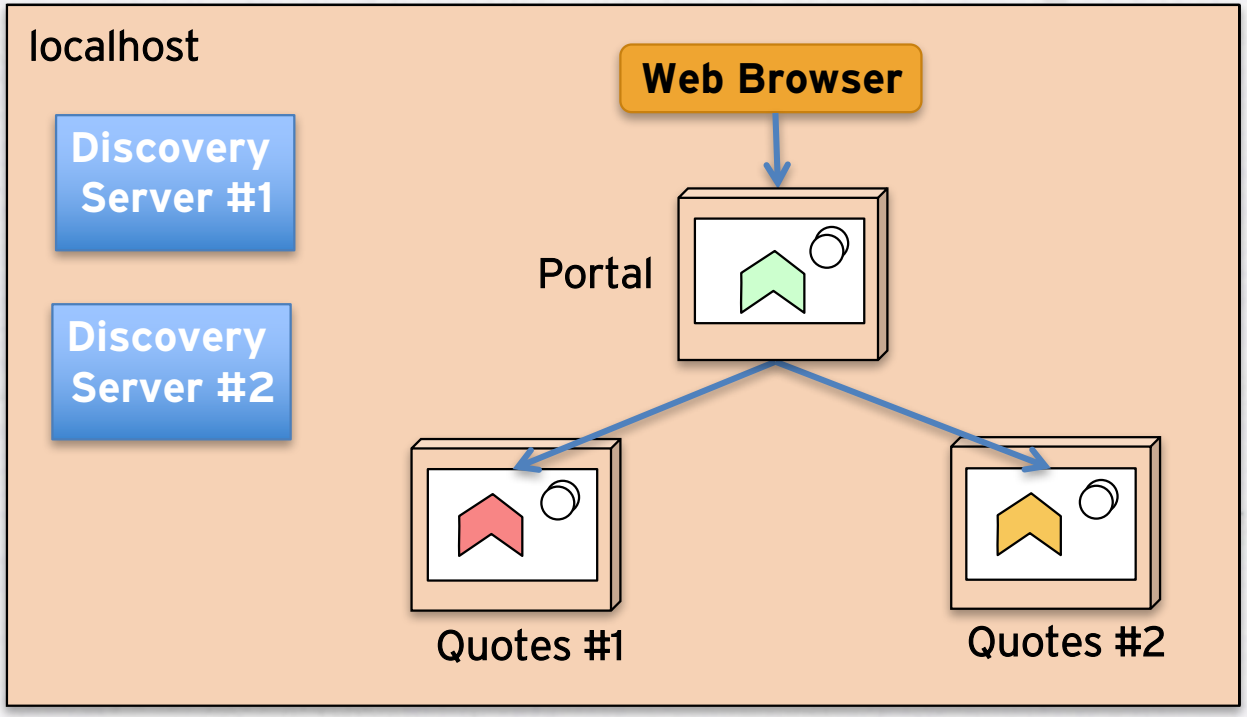
Netflix Eureka

Spring Cloud alternatives:

- **Apache ZooKeeper**
- **Hashicorp Consul**
- **CoreOS etcd**

EXAMPLE #1 - SPRING CLOUD AND NETFLIX OSS (NO CONTAINERS)

DEMO



EXAMPLE #1 - SPRING CLOUD AND NETFLIX OSS (NO CONTAINERS)

DEMO

- Start up all 5 Java programs
  - Two discovery servers, one portal instance, two quote services
  - `termrc start`

- Open portal web app: localhost:9090

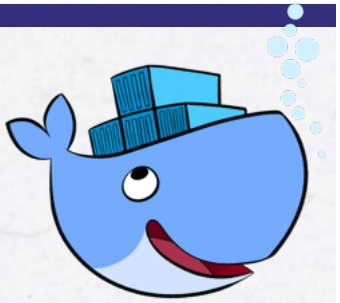
- Verify load balancing
- Kill one of the quotes instances…
- Start it again…

EXAMPLE #1 - SPRING CLOUD AND NETFLIX OSS (NO CONTAINERS)

Expected result:

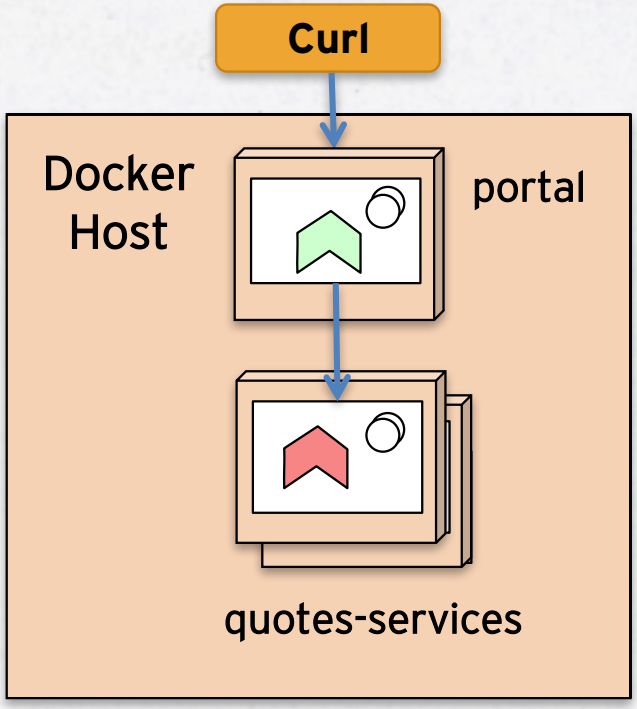
Timestamp	Response Time	Quote	IP Address
2017-01-03 14:16:31 743	355	You, too, Brutus?	Magnus-MBP.lan/192.168.1.19 :8080
2017-01-03 14:16:30 736	349	Champagne should be cold, dry and free	Magnus-MBP.lan/192.168.1.19 :8081
2017-01-03 14:16:29 719	332	You, too, Brutus?	Magnus-MBP.lan/192.168.1.19 :8082
2017-01-03 14:16:28 726	340	To be or not to be	Magnus-MBP.lan/192.168.1.19 :8080
2017-01-03 14:16:27 708	322	You, too, Brutus?	Magnus-MBP.lan/192.168.1.19 :8081
2017-01-03 14:16:26 701	315	Champagne should be cold, dry and free	Magnus-MBP.lan/192.168.1.19 :8082

EXAMPLE #2 - PLAIN CONTAINERS (NO ORCHESTRATION)?



- **2016-02-04:** Docker 1.10 released
  - Docker gets a built in dynamic DNS server!
  - Plain Docker Compose sufficient?

```
version: '2'

services:

  quotes-service:
    image: magnuslarsson/quotes:16

  portal-service:
    image: magnuslarsson/portal:17
    ports:
      - "9090:9090"
```



EXAMPLE #2 - PLAIN CONTAINERS (NO ORCHESTRATION)?

- Docker 1.10 with a built in DNS server!

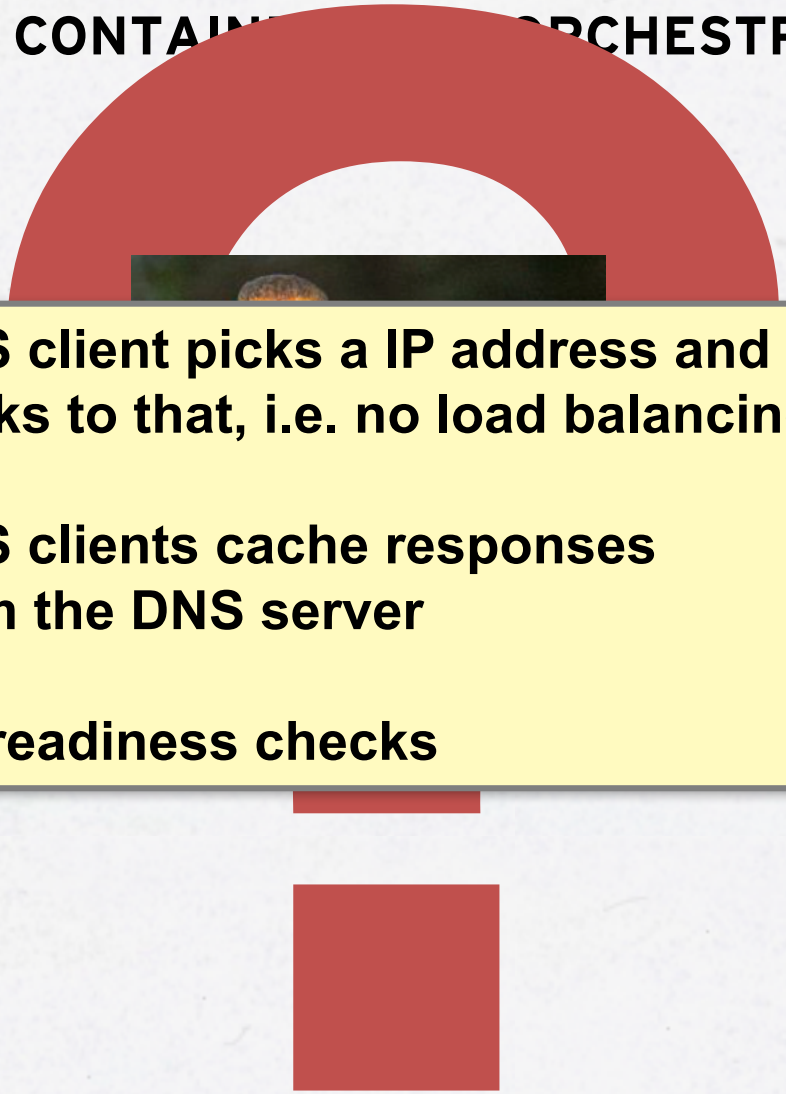
```
docker-compose up
docker-compose scale quotes-service=2

docker-compose exec portal-service nslookup quotes-service
```

- Sample output

```
$ docker-compose exec portal-service nslookup quotes-service
Name:      quotes-service
Address 1: 172.19.0.3 dockercomposev2_quotes-service_1.dockercomposev2_default
Address 2: 172.19.0.4 dockercomposev2_quotes-service_2.dockercomposev2_default
```

EXAMPLE #2 - PLAIN CONTAIN… …RCHESTRATION)?

- localhost:9090

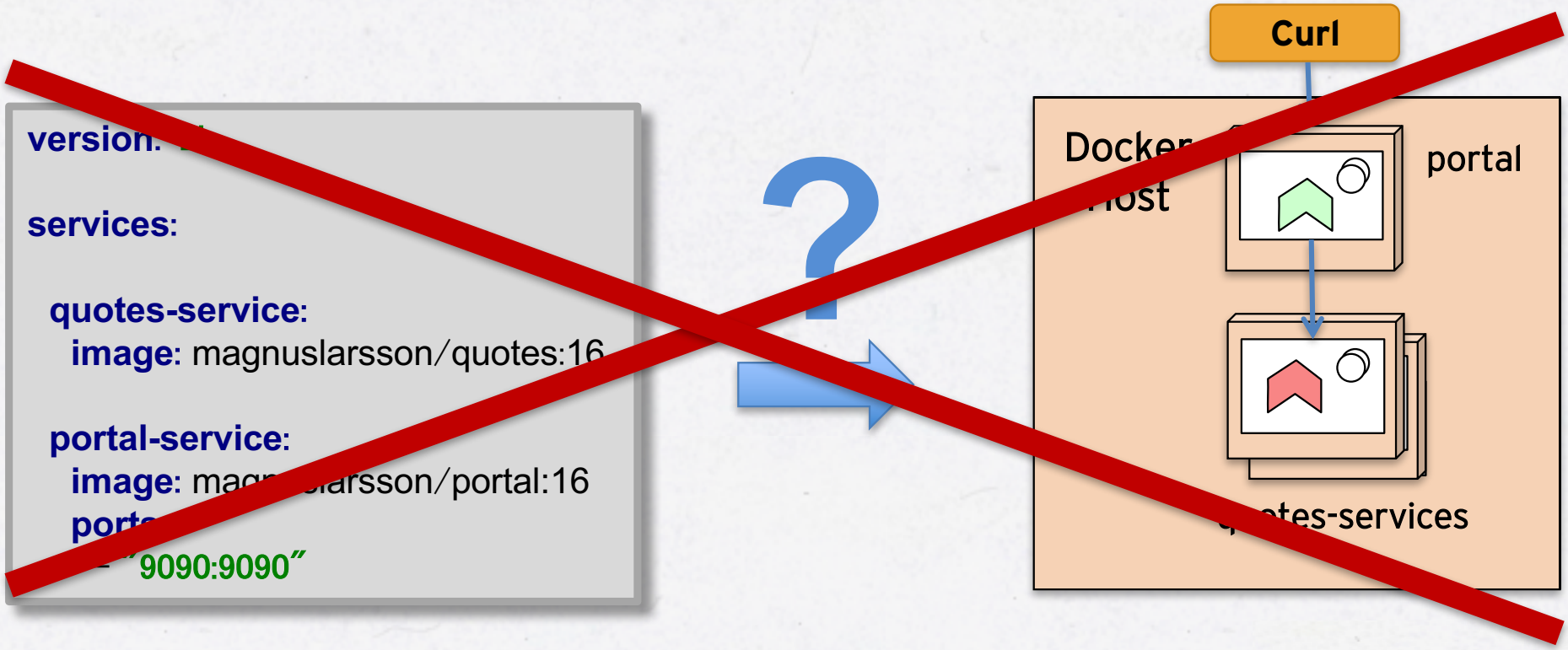


1. DNS client picks a IP address and sticks to that, i.e. no load balancing!
2. DNS clients cache responses from the DNS server
3. No readiness checks

2017-01-02 17:26:28 801		a4dc9d5c6114	172.19.0.3 8080
2017-01-02 17:26:27 797		a4dc9d5c6114	172.19.0.3 8080
2017-01-02 17:26:26 800		a4dc9d5c6114	172.19.0.3 8080
2017-01-02 17:26:25 794		a4dc9d5c6114	172.19.0.3 8080

EXAMPLE #2 - PLAIN CONTAINERS (NO ORCHESTRATION)?

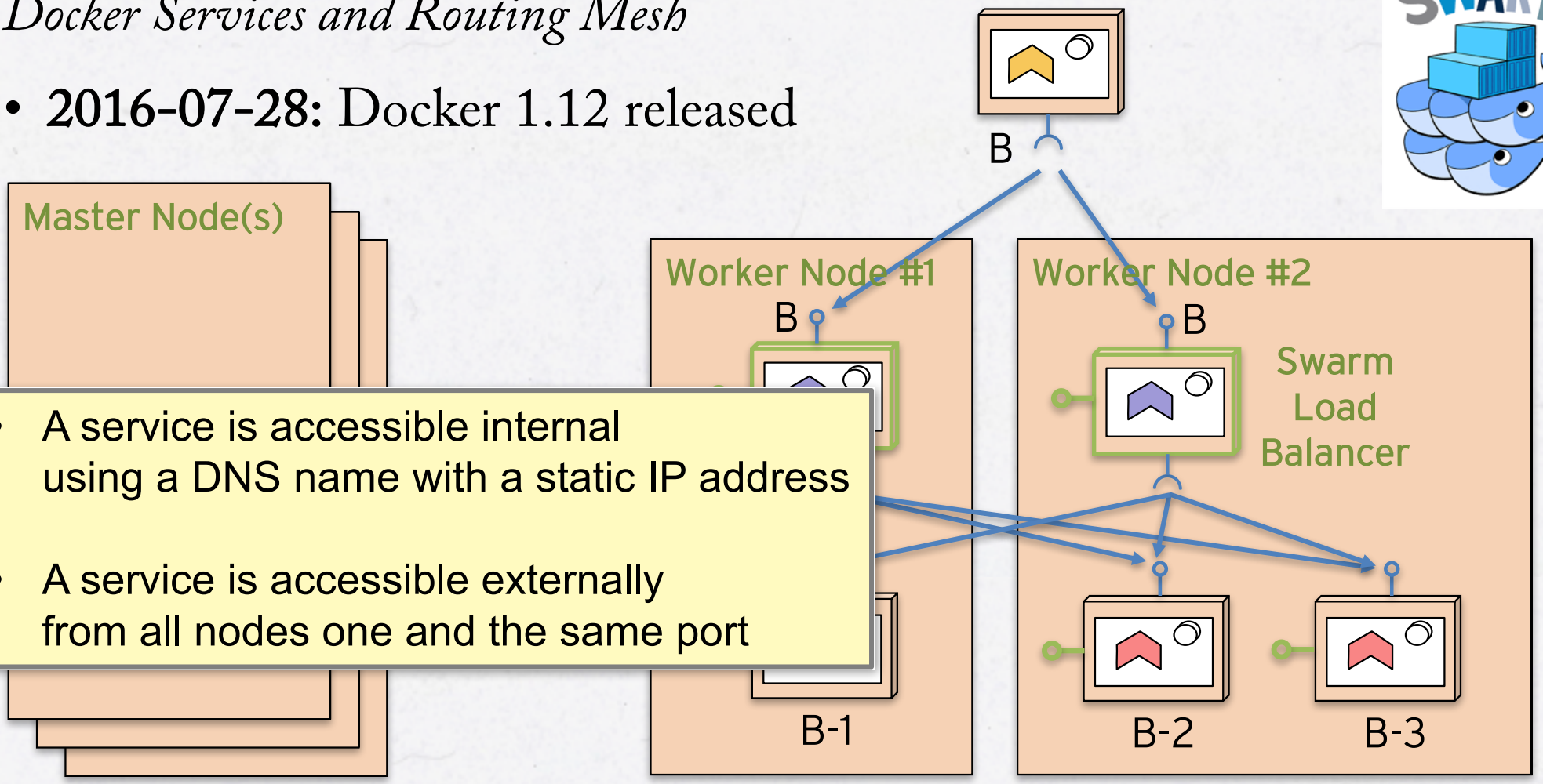
- **2016-02-04:** Docker 1.10 released
  - Docker gets a built in DNS server!



EXAMPLE #3: DOCKER SWARM MODE

Docker Services and Routing Mesh

- **2016-07-28:** Docker 1.12 released



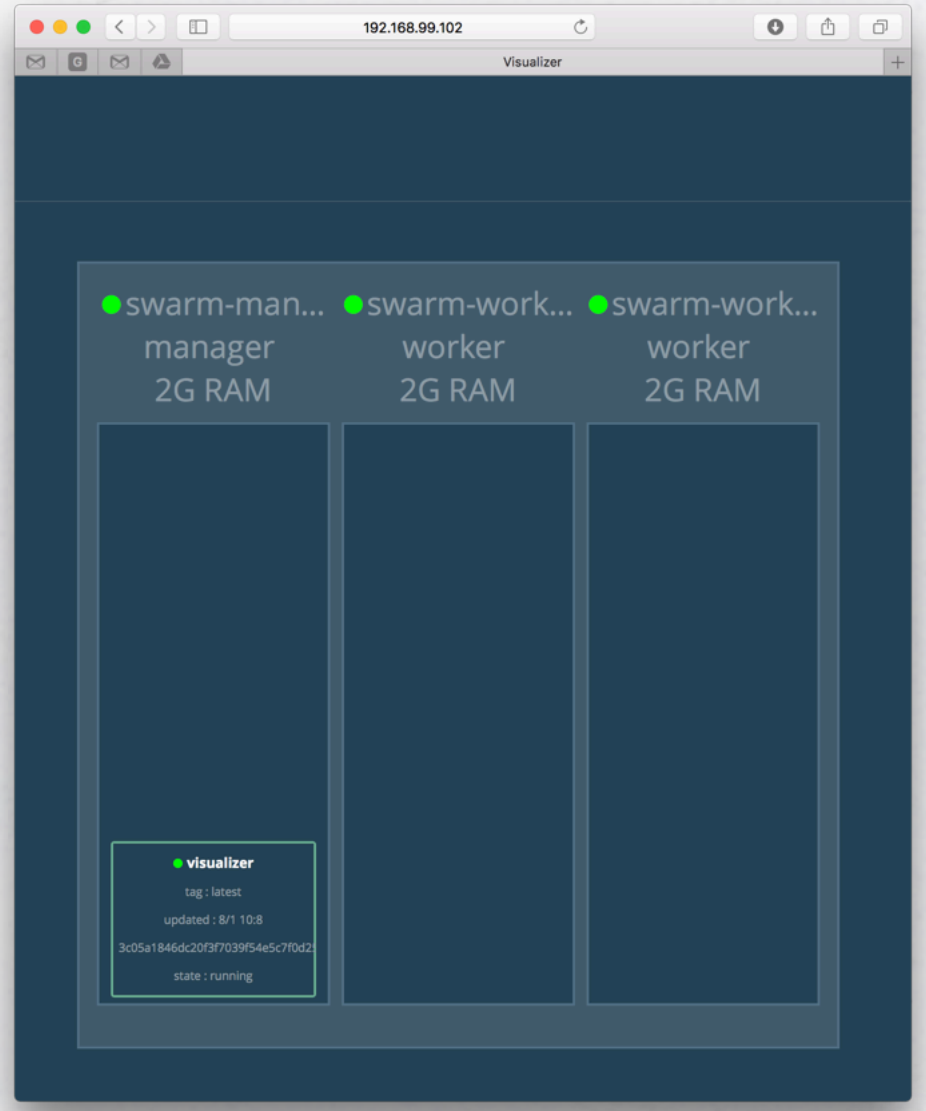
- A service is accessible internal using a DNS name with a static IP address
- A service is accessible externally from all nodes one and the same port

EXAMPLE #3: DOCKER SWARM MODE

DEMO

- 3 node cluster
  (local VirtualBox nodes)

```
$ docker swarm init ...
$ docker swarm join ...
$ docker swarm join ...

$ docker service create \
    --name=viz \
    --publish=8000:8080 \
    manomarks/visualizer
```



EXAMPLE #3: DOCKER SWARM MODE

- Deploy quotes and portal

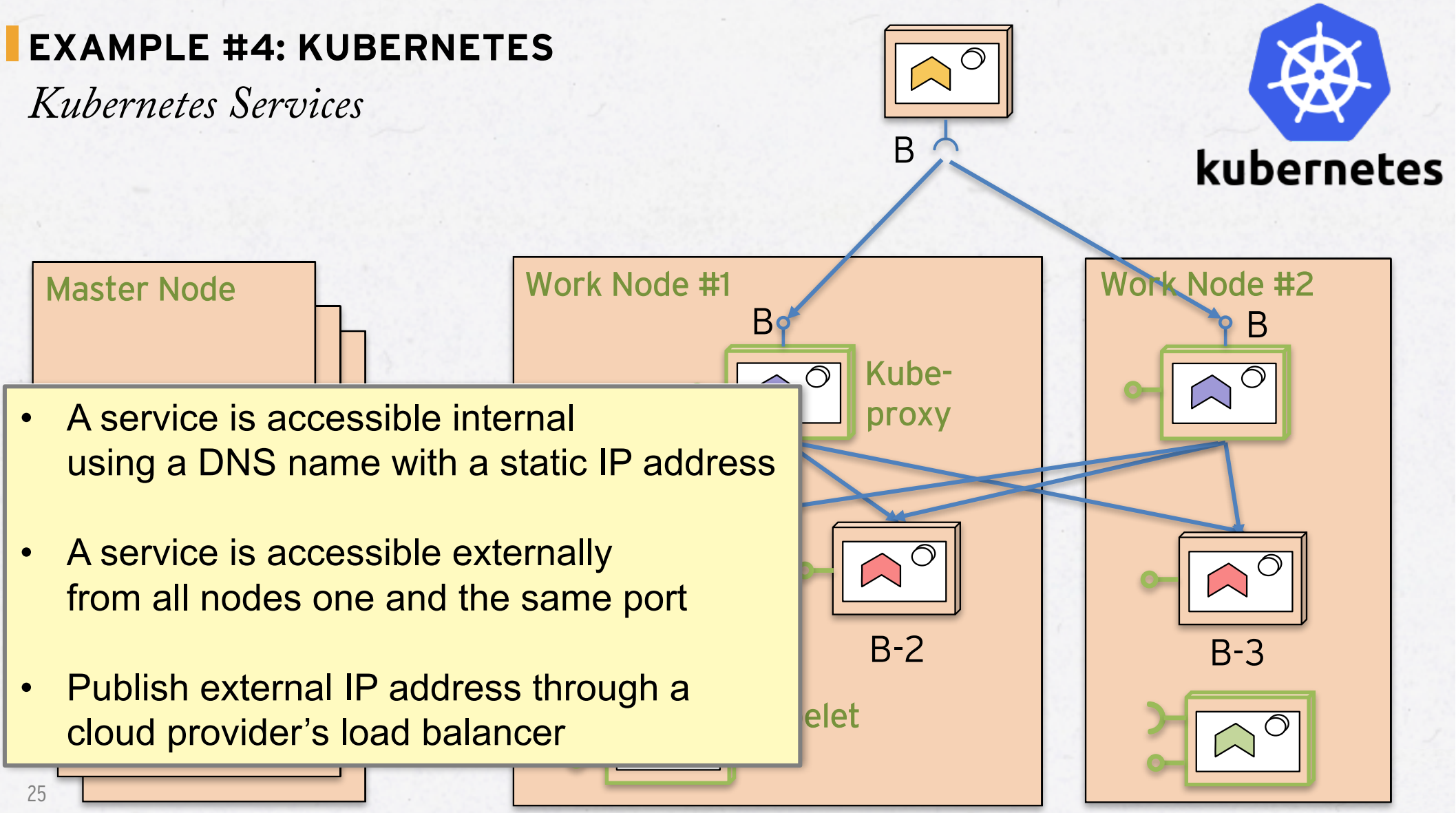
```
$ docker network create \
  --driver overlay my_network

$ docker service create \
    --name quotes-service \
    --replicas 3 \
    --network my_network \
    magnuslarsson/quotes:16 \

$ docker service create \
    --name portal \
    --publish 9090:9090 \
    --replicas 1 \
    --network my_network \
    magnuslarsson/portal:17
```

EXAMPLE #3: DOCKER SWARM MODE

1. Start portal
2. Kill quote container
3. Kill node with quote containers

EXAMPLE #4: KUBERNETES

Kubernetes Services



- A service is accessible internal using a DNS name with a static IP address
- A service is accessible externally from all nodes one and the same port
- Publish external IP address through a cloud provider's load balancer

EXAMPLE #4: KUBERNETES

DEMO

- Setup a Kubernetes Cluster in Google Cloud
- Let Kubernetes auto configure Google Cloud Load Balancer
- Let's try auto scaling!
  - Both pods (e.g. containers) and nodes
- Put some load on the cluster and see if we can get new pods and new nodes started up…

EXAMPLE #4: KUBERNETES

- Setup cluster on Google Cloud with auto scaling of nodes

```
$ export KUBE_GCE_ZONE=europe-west1-b
$ export NODE_SIZE=n1-standard-1
$ export KUBE_ENABLE_CLUSTER_AUTOSCALER=true
$ export KUBE_AUTOSCALER_MIN_NODES=1
$ export KUBE_AUTOSCALER_MAX_NODES=5

$ kube-up.sh
```

EXAMPLE #4: KUBERNETES

- Deploy quotes and portal using service type *LoadBalancer*

```
$ kubectl run quotes --image=magnuslarsson/quotes:16 --port=8080
$ kubectl expose deployment quotes --type=LoadBalancer --name quotes-service

$ kubectl run portal --image=magnuslarsson/portal:17 --port=9090
$ kubectl expose deployment portal --type=LoadBalancer --name portal-service
```

- Enable auto scaling of quote pods

```
$ kubectl autoscale deployment quotes --cpu-percent=50 --min=1 --max=10
```

EXAMPLE #4: KUBERNETES

- Services

```
$ kubectl get svc
NAME              CLUSTER-IP    EXTERNAL-IP      PORT(S)    AGE
portal-service    10.0.94.30    146.148.16.15    9090/TCP   3d
quotes-service    10.0.17.114   23.251.139.163   8080/TCP   3d
```

- Pods

```
$ kubectl get pods
NAME                       READY     STATUS    RESTARTS   AGE
portal-329106472-6qw13     1/1       Running   0          3d
quotes-4111254610-x7ng5    1/1       Running   0          3d
```

- Nodes

```
$ kubectl get nodes
NAME                          STATUS                        AGE
kubernetes-master             Ready,SchedulingDisabled      3d
kubernetes-minion-group-clbp  Ready                         3d
```

EXAMPLE #4: KUBERNETES

- Put some load using the portal and wait for a while…

```
$ kubectl get hpa
NAME      REFERENCE             TARGET    CURRENT   MINPODS   MAXPODS   AGE
quotes    Deployment/quotes     50%       433%      1         10        3m
```

- Any new pods?

```
$ kubectl get pods
NAME                       READY     STATUS    RESTARTS   AGE
quotes-4029858897-2nsm6    1/1       Running   0          2m
quotes-4029858897-5xn93    1/1       Running   0          17m
quotes-4029858897-82vdc    1/1       Running   0          6m
quotes-4029858897-d5ctp    1/1       Running   0          6m
quotes-4029858897-s6sl4    0/1       Pending   0          2m
quotes-4029858897-t7sbj    1/1       Running   0          6m
quotes-4029858897-w8crj    0/1       Pending   0          2m
quotes-4029858897-xm68g    1/1       Running   0          2m
```

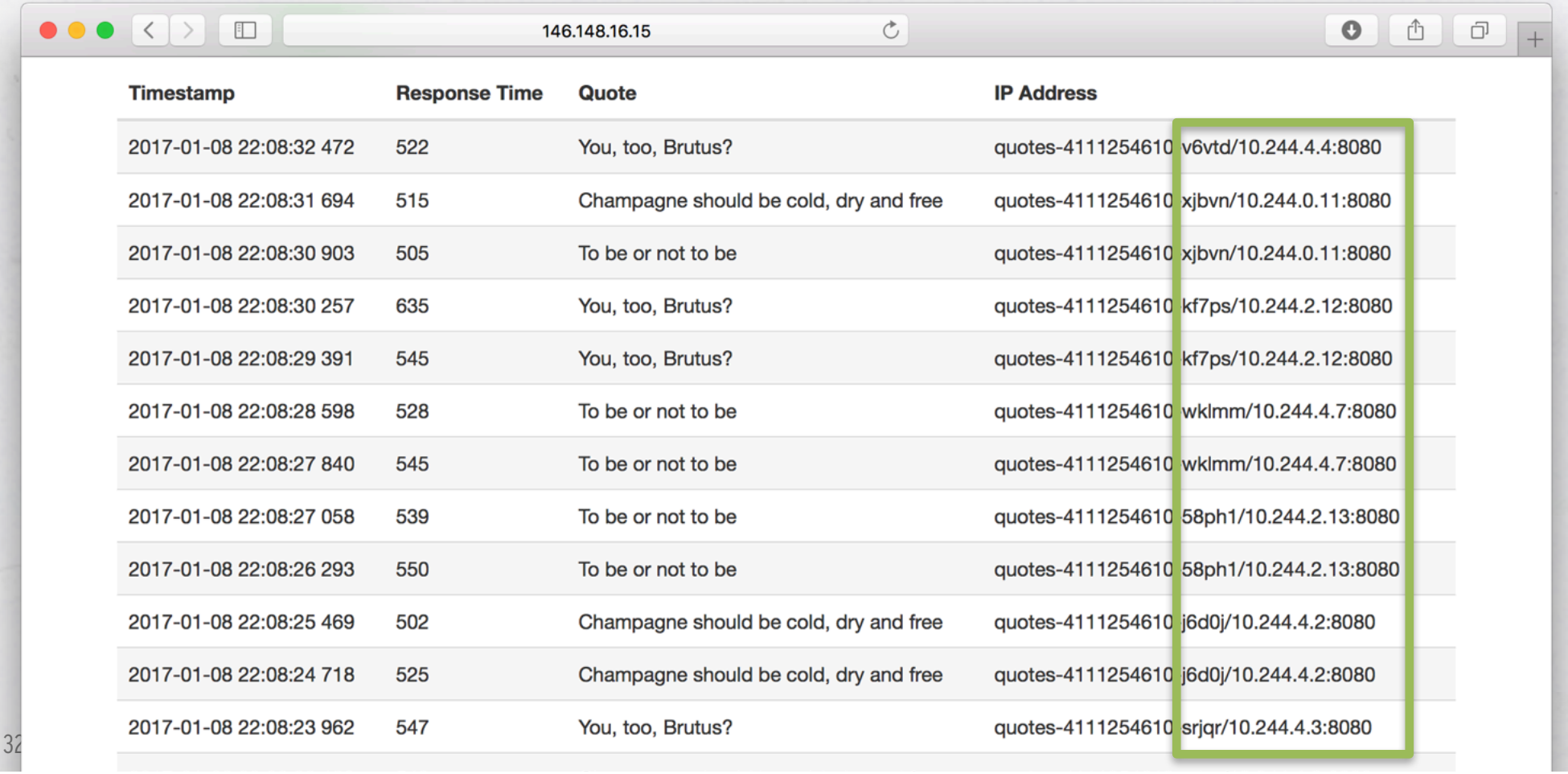
EXAMPLE #4: KUBERNETES

- Any new nodes?

```
$ kubectl get nodes
NAME                          STATUS                        AGE
kubernetes-master             Ready,SchedulingDisabled      21m
kubernetes-minion-group-4ptj  Ready                         22m
kubernetes-minion-group-l6kv  NotReady                      6s
kubernetes-minion-group-xq6d  Ready                         18m
```

EXAMPLE #4: KUBERNETES

- Portal uses new pods!

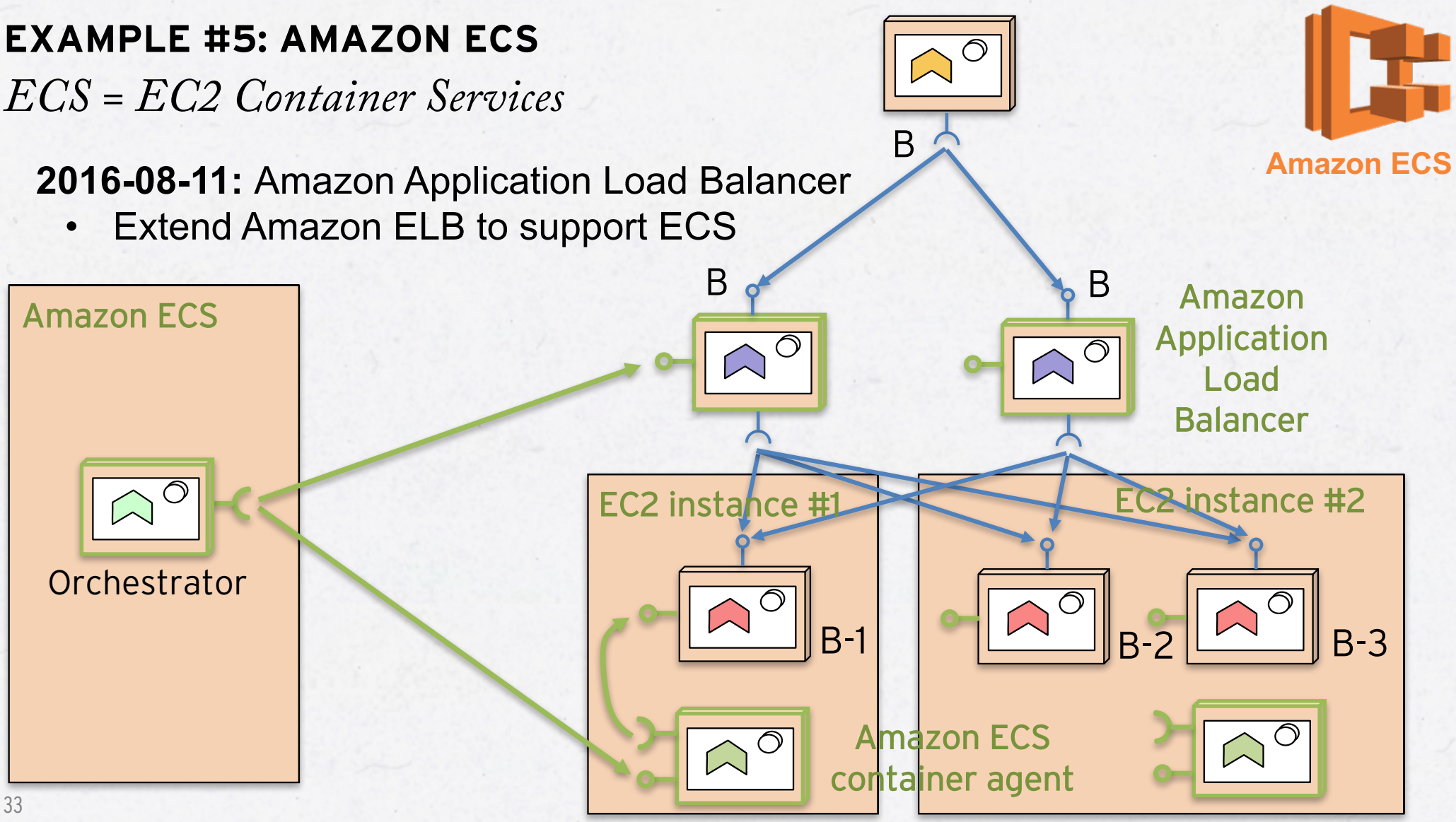


Timestamp	Response Time	Quote	IP Address
2017-01-08 22:08:32 472	522	You, too, Brutus?	quotes-4111254610 v6vtd/10.244.4.4:8080
2017-01-08 22:08:31 694	515	Champagne should be cold, dry and free	quotes-4111254610 xjbvn/10.244.0.11:8080
2017-01-08 22:08:30 903	505	To be or not to be	quotes-4111254610 xjbvn/10.244.0.11:8080
2017-01-08 22:08:30 257	635	You, too, Brutus?	quotes-4111254610 kf7ps/10.244.2.12:8080
2017-01-08 22:08:29 391	545	You, too, Brutus?	quotes-4111254610 kf7ps/10.244.2.12:8080
2017-01-08 22:08:28 598	528	To be or not to be	quotes-4111254610 wklmm/10.244.4.7:8080
2017-01-08 22:08:27 840	545	To be or not to be	quotes-4111254610 wklmm/10.244.4.7:8080
2017-01-08 22:08:27 058	539	To be or not to be	quotes-4111254610 58ph1/10.244.2.13:8080
2017-01-08 22:08:26 293	550	To be or not to be	quotes-4111254610 58ph1/10.244.2.13:8080
2017-01-08 22:08:25 469	502	Champagne should be cold, dry and free	quotes-4111254610 j6d0j/10.244.4.2:8080
2017-01-08 22:08:24 718	525	Champagne should be cold, dry and free	quotes-4111254610 j6d0j/10.244.4.2:8080
2017-01-08 22:08:23 962	547	You, too, Brutus?	quotes-4111254610 srjqr/10.244.4.3:8080

EXAMPLE #5: AMAZON ECS

ECS = EC2 Container Services

- **2016-08-11:** Amazon Application Load Balancer
  - Extend Amazon ELB to support ECS

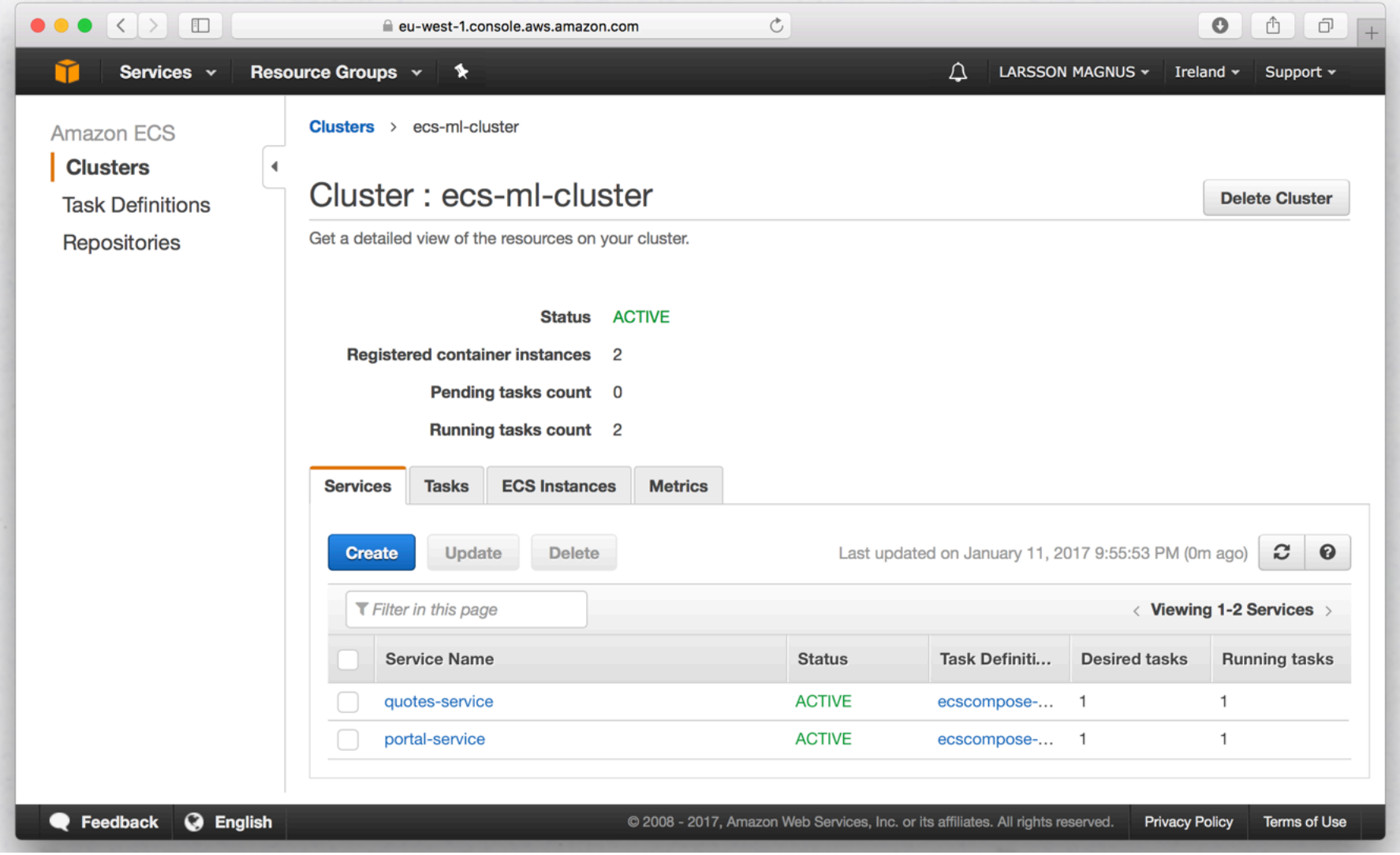


EXAMPLE #5: AMAZON ECS

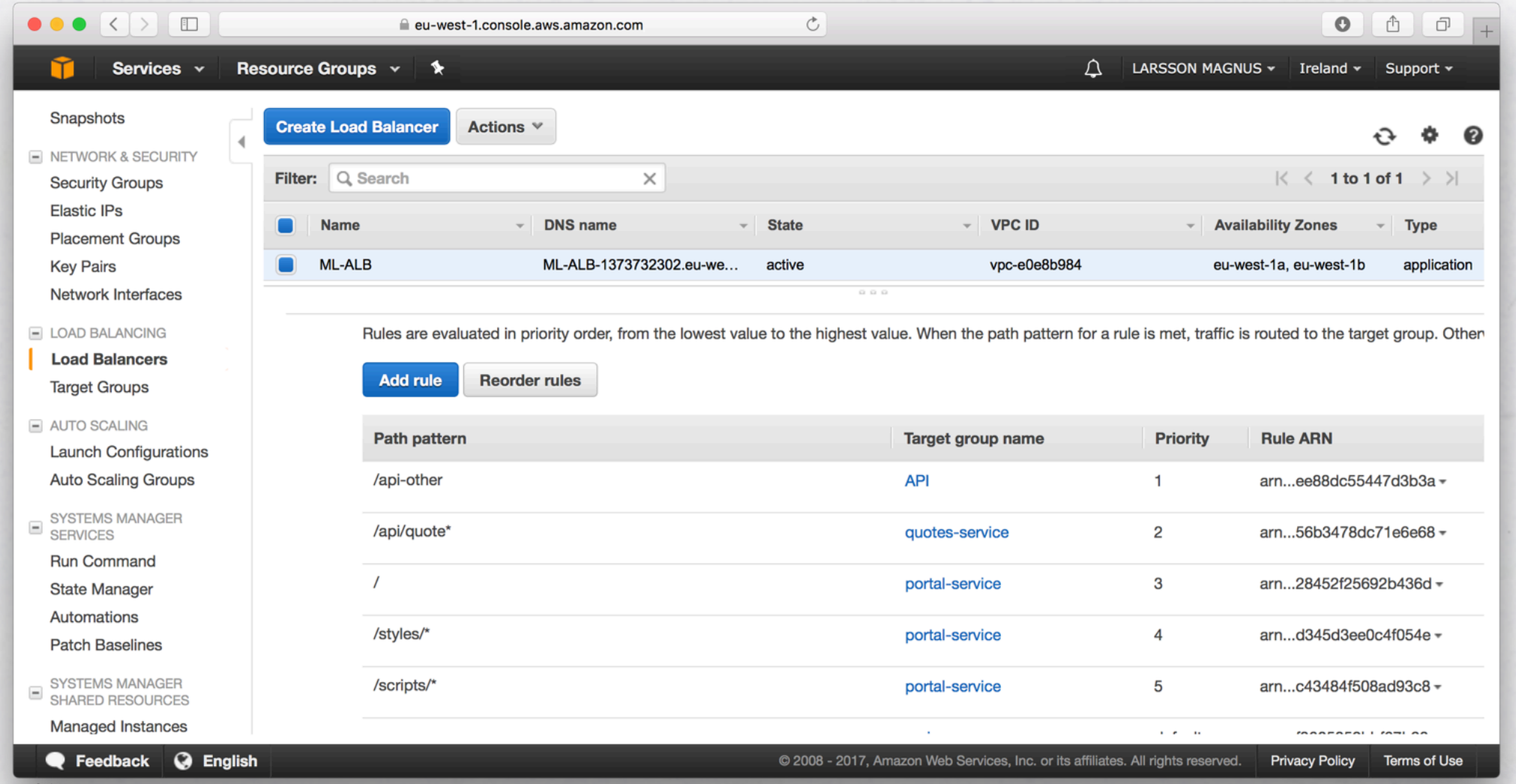
DEMO

- Setup a Amazon ECS Cluster with Application Load Balancer
- Let's try auto scaling!
  - Both pods (e.g. containers) and nodes
- Put some load on the cluster and see if we can get new tasks and new nodes started up…

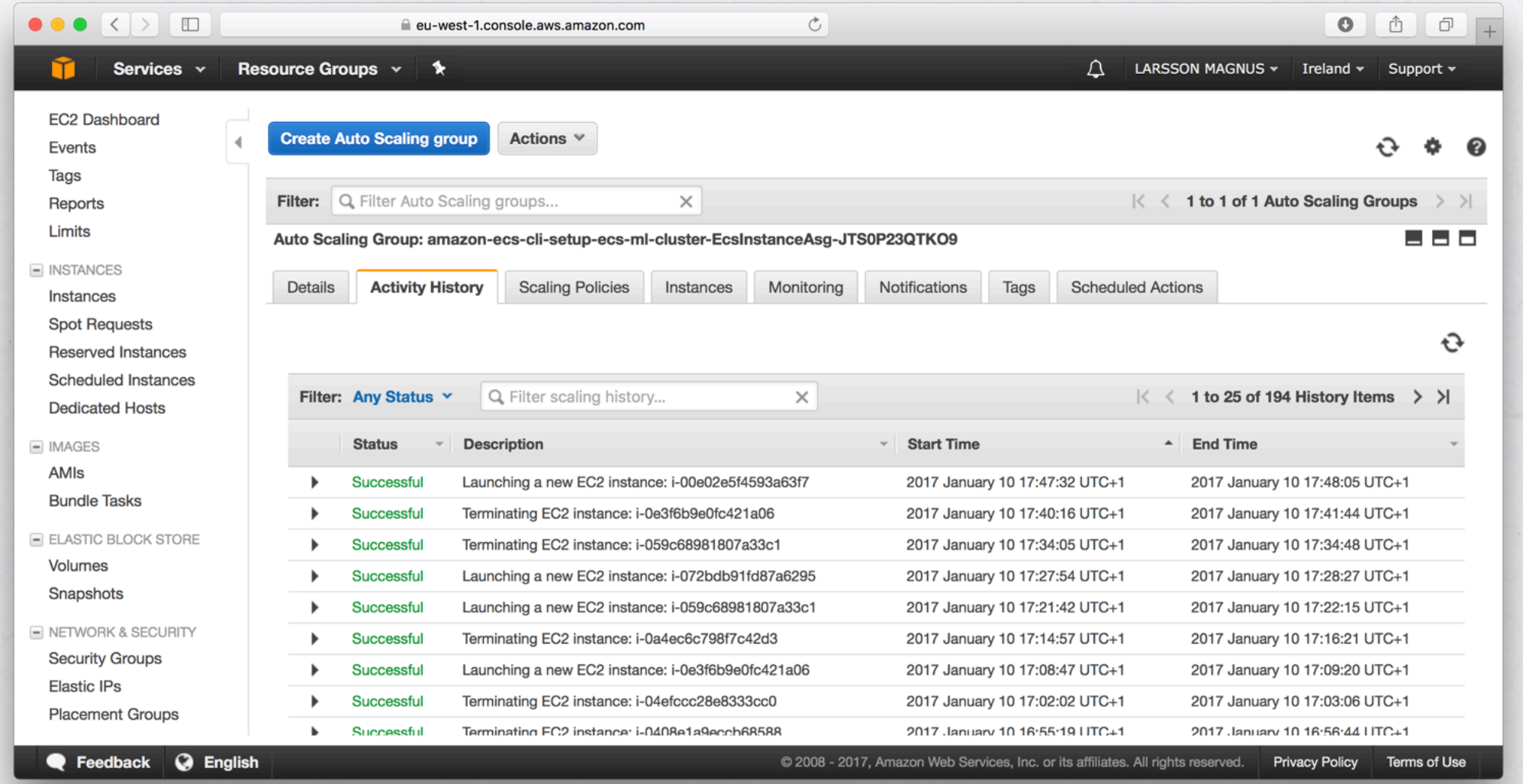
AMAZON ECS CLUSTER



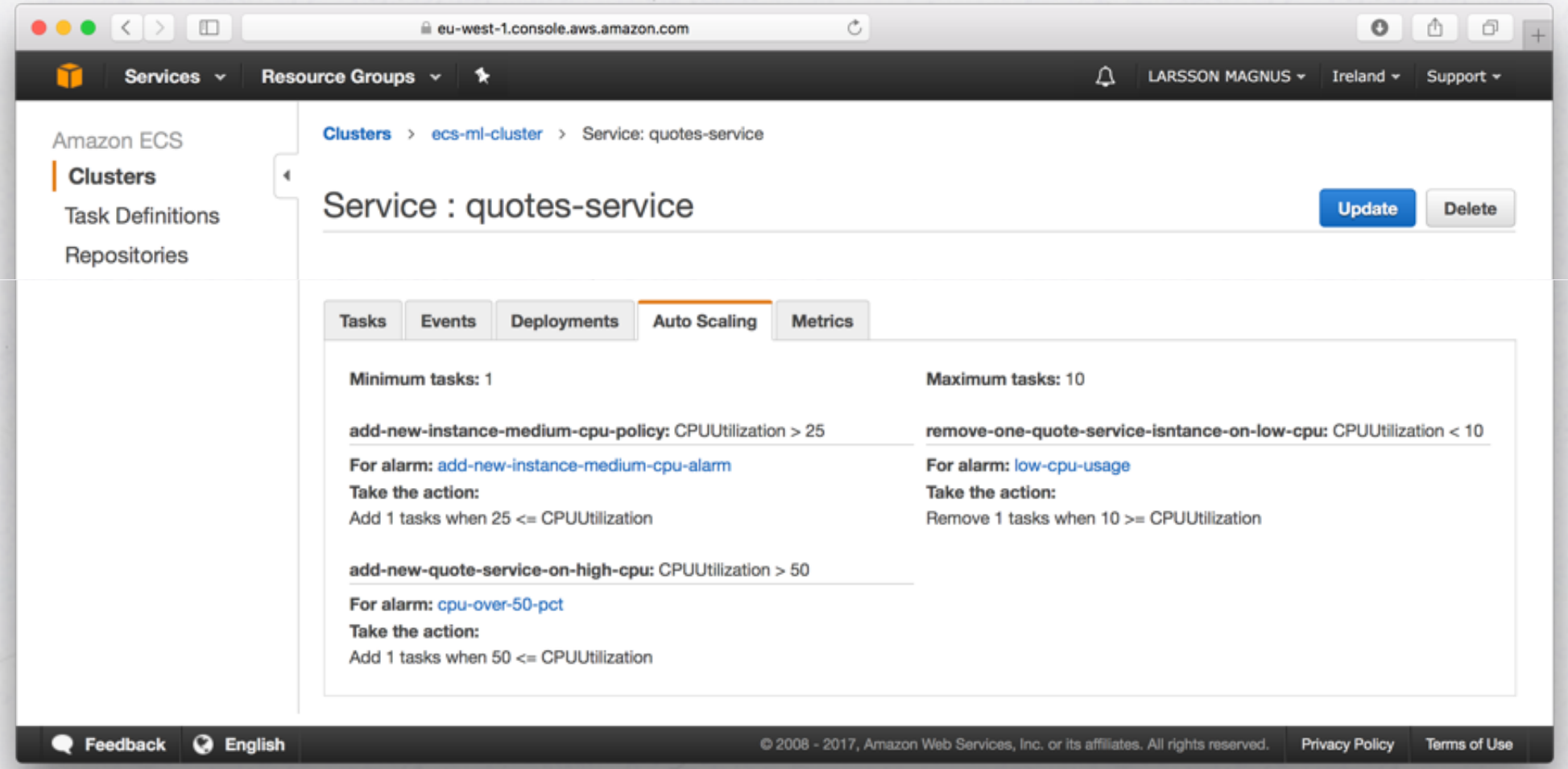
AMAZON APPLICATION LOAD BALANCER WITH PATH BASED ROUTING



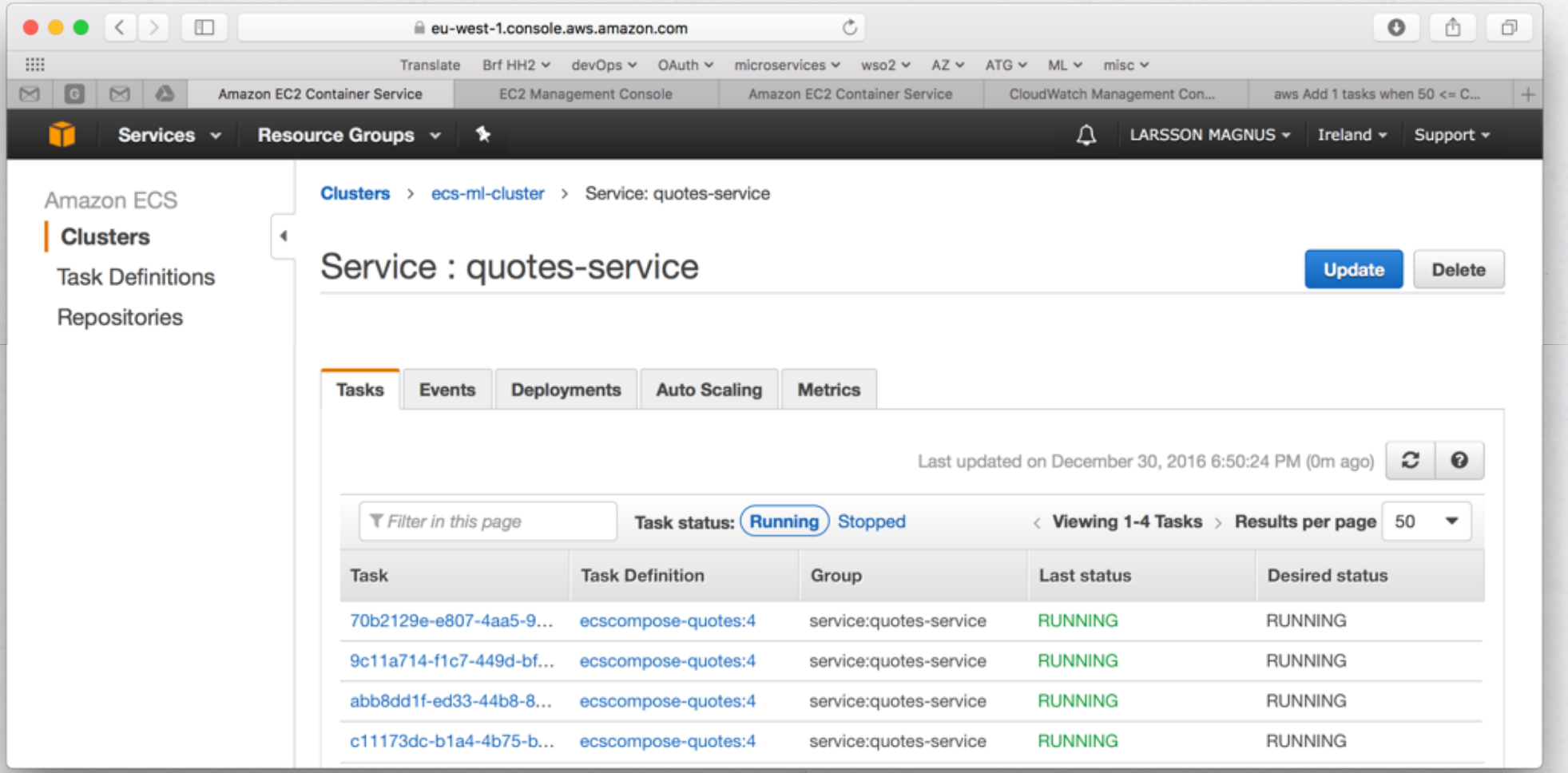
AMAZON EC2 INSTANCE (NODES) AUTO SCALING



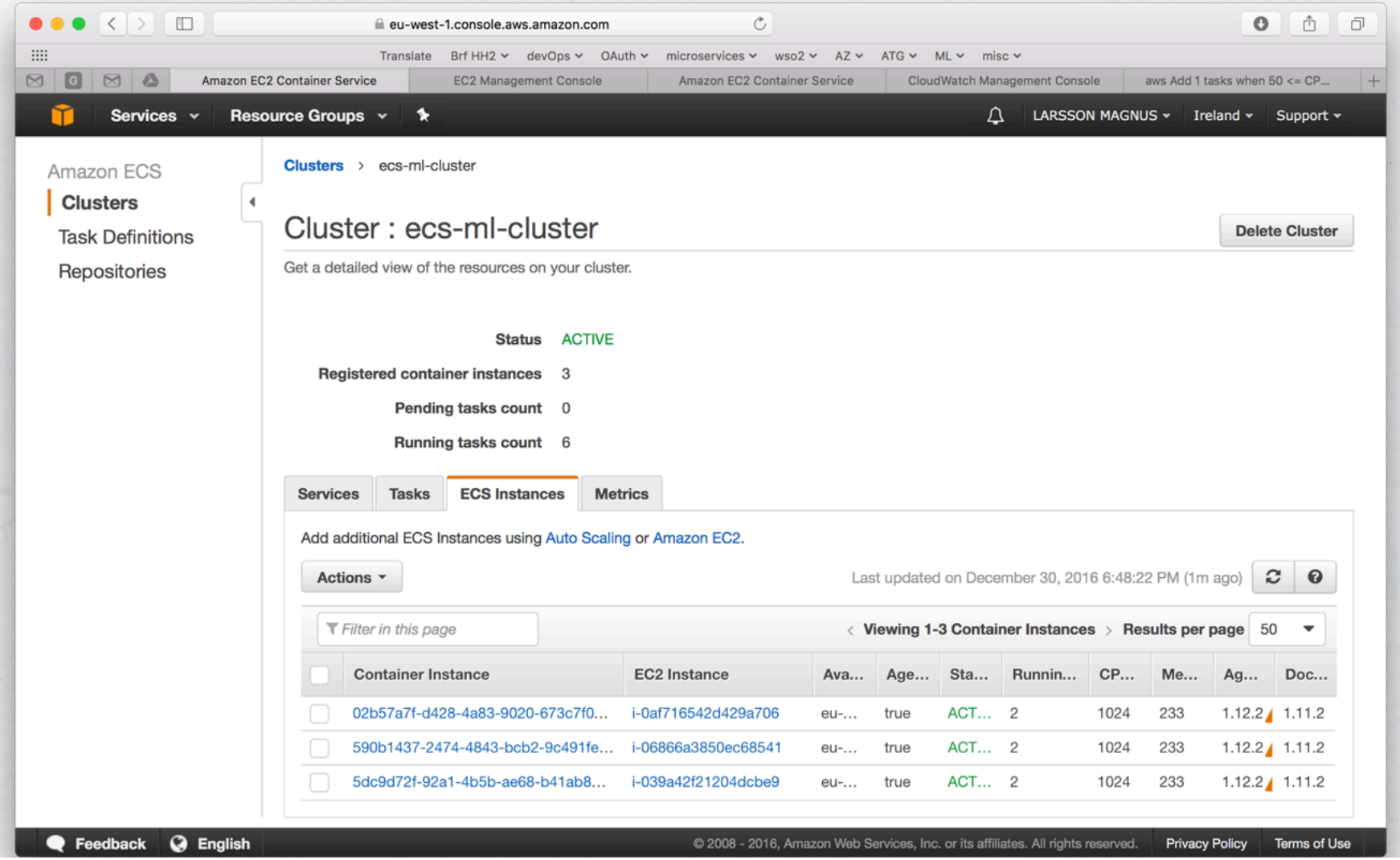
AMAZON ECS TASKS (CONTAINERS) AUTO SCALING



NEW ECC TASKS (CONTAINERS) AUTO CREATED UNDER LOAD

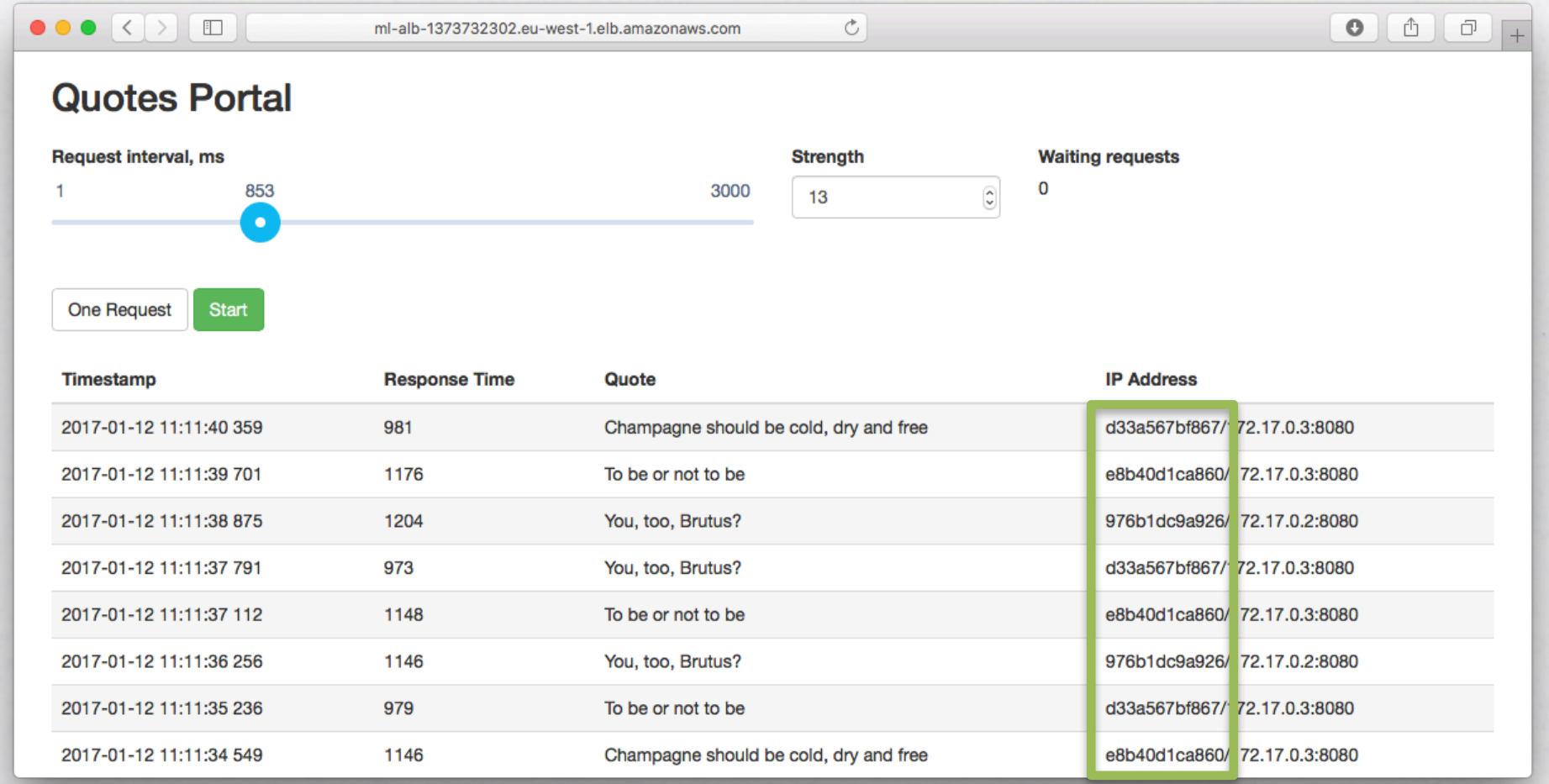


NEW EC2 INSTANCES (NODES) AUTO CREATED UNDER LOAD

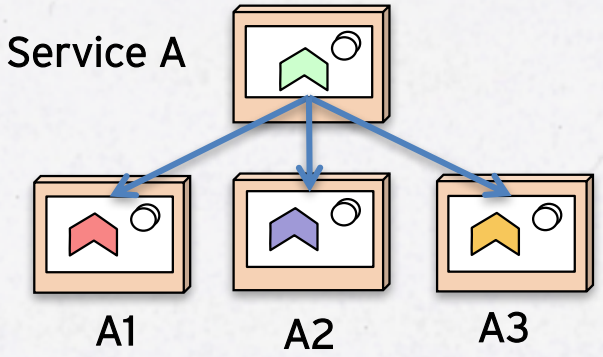


EXAMPLE #5: AMAZON ECS

- Portal uses new tasks!



PROGRAMMING MODEL



- I used Java and Spring Cloud…
  - Can be accomplished using any language
- Orchestration tools (Swarm, Kubernetes, ECS) expose an abstraction of a service on top of the actual containers
- Spring Cloud provides a similar abstraction on top of Netflix Eureka and Netflix Ribbon
- Spring Boot provides a customizable Health API for liveness and readiness probes

SOME CODE FRAGMENTS…

```
@SpringBootApplication
public class PortalApplication {


  @Bean
  public RestTemplate restTemplate() {
    return new RestTemplate();
  }
```

1. Integrate with Netflix Eureka
2. Wrap Netflix Ribbon
3. Abstract service name resolved by Netflix Ribbon using the configuration

```
@RestController
public class PortalController {

  @Inject RestTemplate restTemplate;

  Quote quote = restTemplate.getForObject("http://quotes-service/" + ..., Quote.class);
```

SOME CONFIGURATION…

```
# Default usage in Docker with Orchestration tools like Swarm, K8S, ECS/ALB
spring.cloud.discovery.enabled: false
ribbon.eureka.enabled: false
quotes-service.ribbon.listOfServers: quotes-service:8080

---
# For usage with Amazon ECS/ALB
spring.profiles: aws-ecs

quotes-service.ribbon.listOfServers: ML-ALB-1373732302.eu-west-1.elb.amazonaws.com

---
# For usage with Eureka outside of Docker
spring.profiles: eureka

spring.cloud.discovery.enabled: true
ribbon.eureka.enabled: true
eureka.client.serviceUrl:
  defaultZone: http://localhost:8761/eureka/,http://localhost:8762/eureka/
```

IF YOU WANT TO LEARN MORE...

- Blog series – Building microservices:
  http://callistaenterprise.se/blogg/teknik/2015/05/20/blog-series-building-microservices/

- Workshop in developing microservices
  - Build a set of collaborating microservices from ground up using Spring Boot, Spring Cloud, Netflix OSS and Docker.
  - Jfokus – 2017-02-06
  - jDays – 2017-03-09

SUMMARY

- Mission accomplished, worked to write once and deployed on:
  - Without containers using ***Netflix Eureka*** as Service Discovery
  - Container orchestration tools (with built in Service Discovery):
    - » ***Kubernetes***
    - » ***Docker Swarm mode***
    - » ***Amazon ECS***
- Only differs in configuration
- Platform specific features (e.g. cloud auto scaling) can be used
- Prevents vendor lock in
- Can be accomplished using any language, not only Java