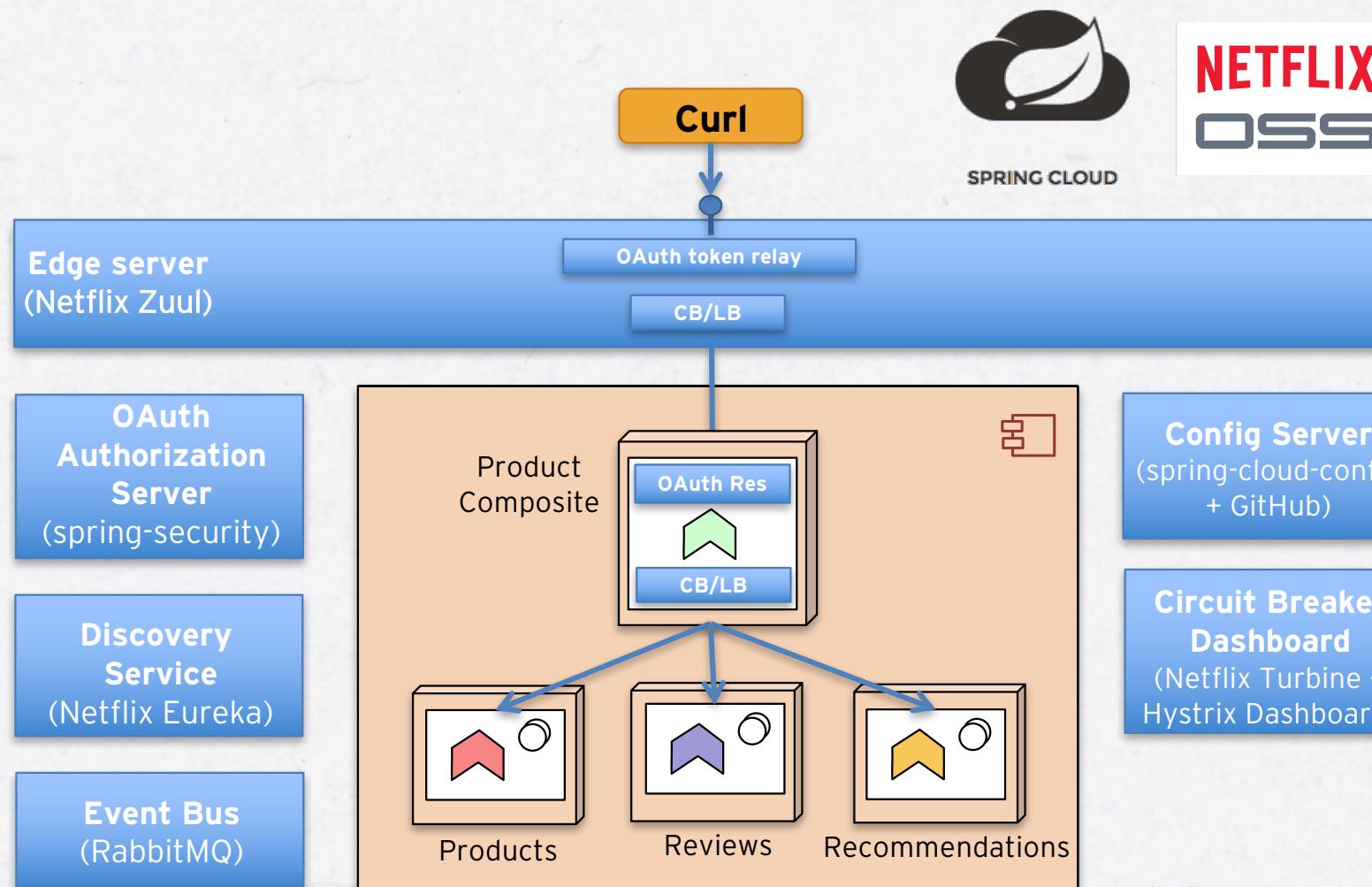


CADEC 2017 - EXPERIENCES FROM USING DISCOVERY SERVICES IN A MICROSERVICE LANDSCAPE

MAGNUS LARSSON

2017-01-25 | CALLISTAENTERPRISE.SE

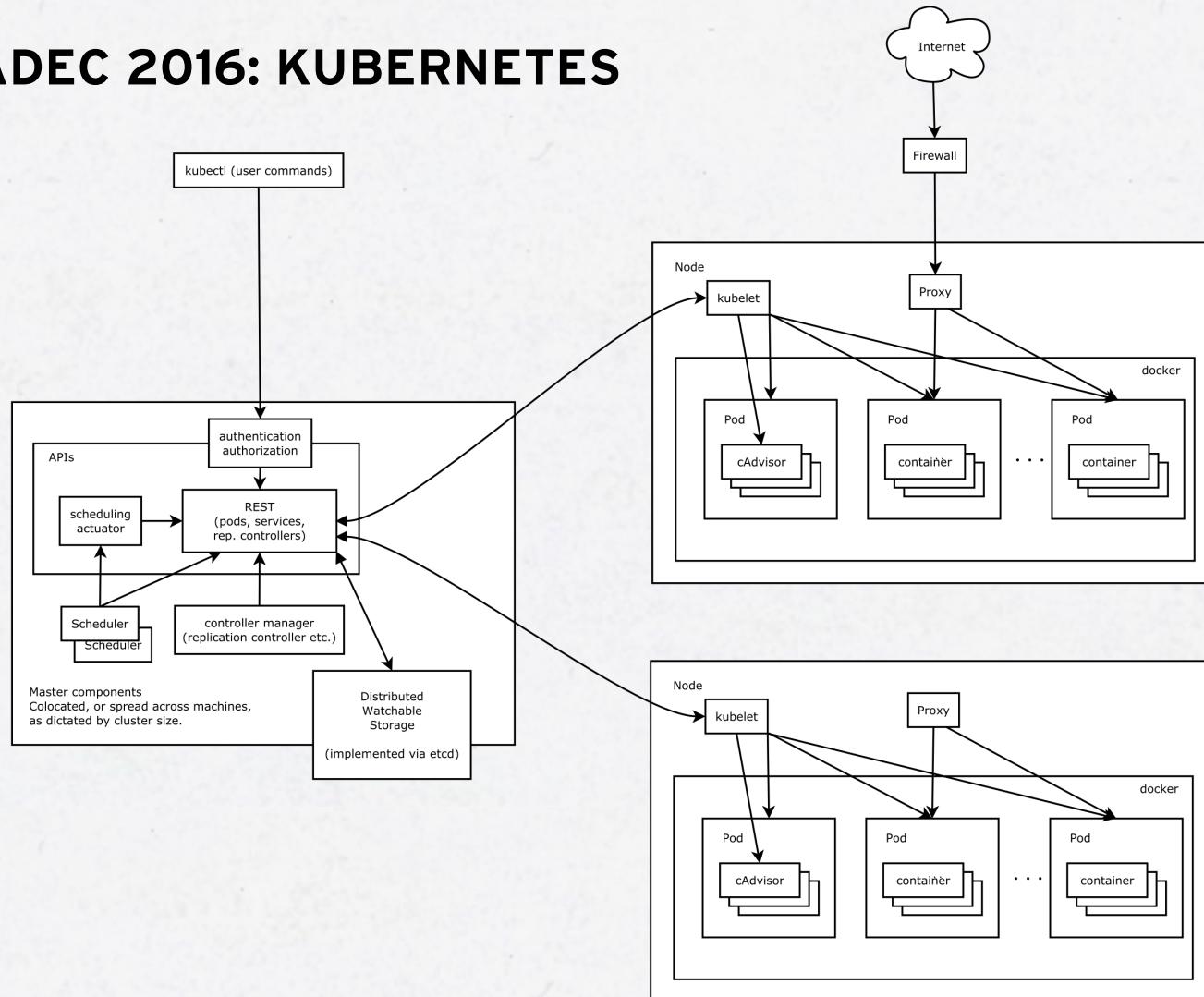
FROM CADEC 2016: MICROSERVICES WITH SPRING CLOUD AND NETFLIX OSS



FROM CADEC 2016: KUBERNETES



kubernetes

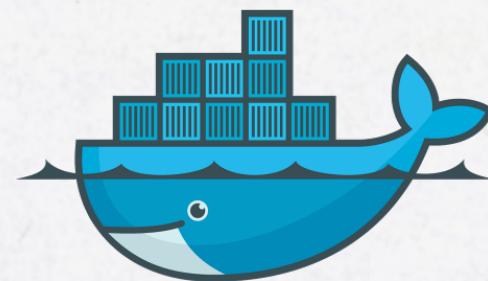


POST CADEC 2016 QUESTION IN MY HEAD



SPRING CLOUD

+



kubernetes

=

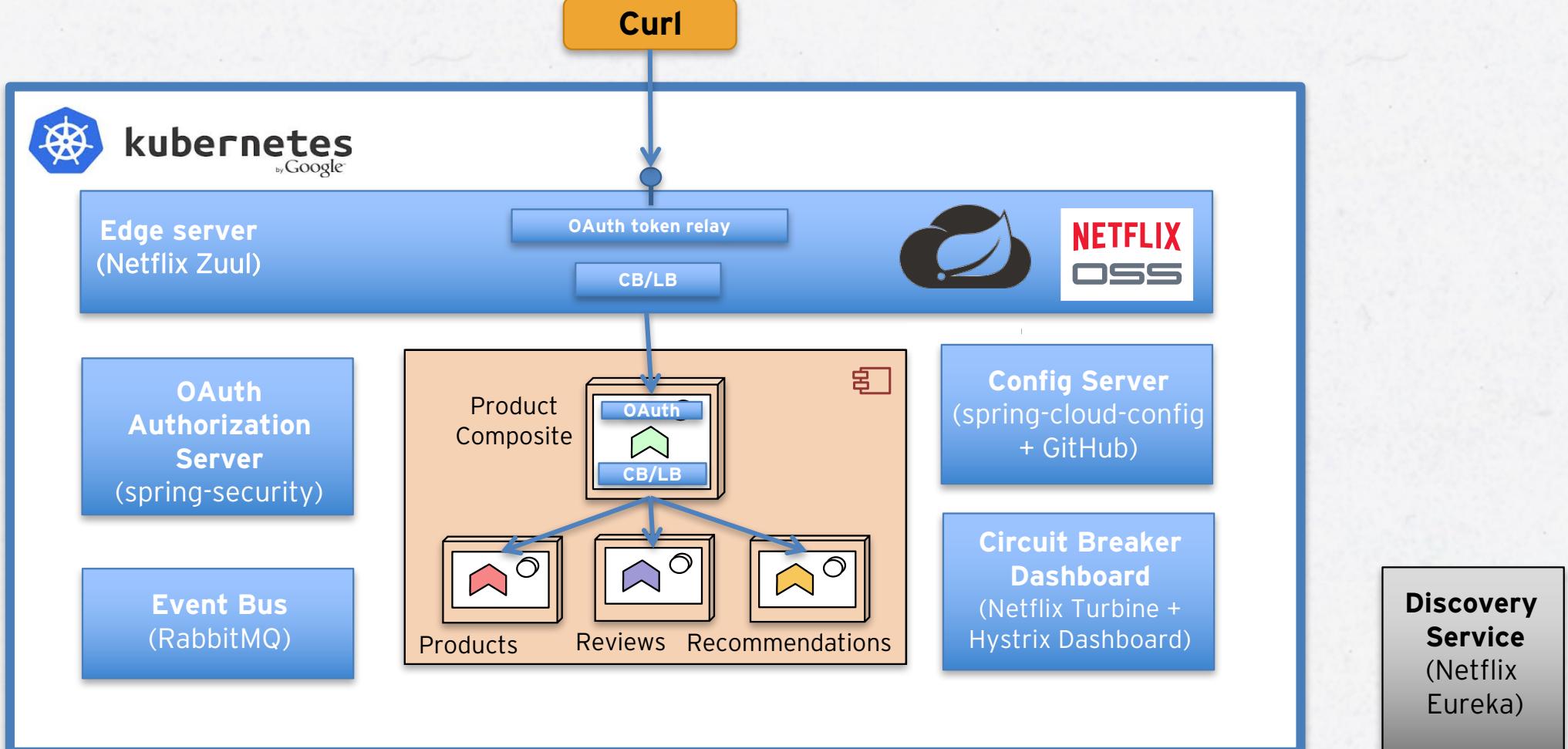


POST CADEC 2016 QUESTION IN MY HEAD

- 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?

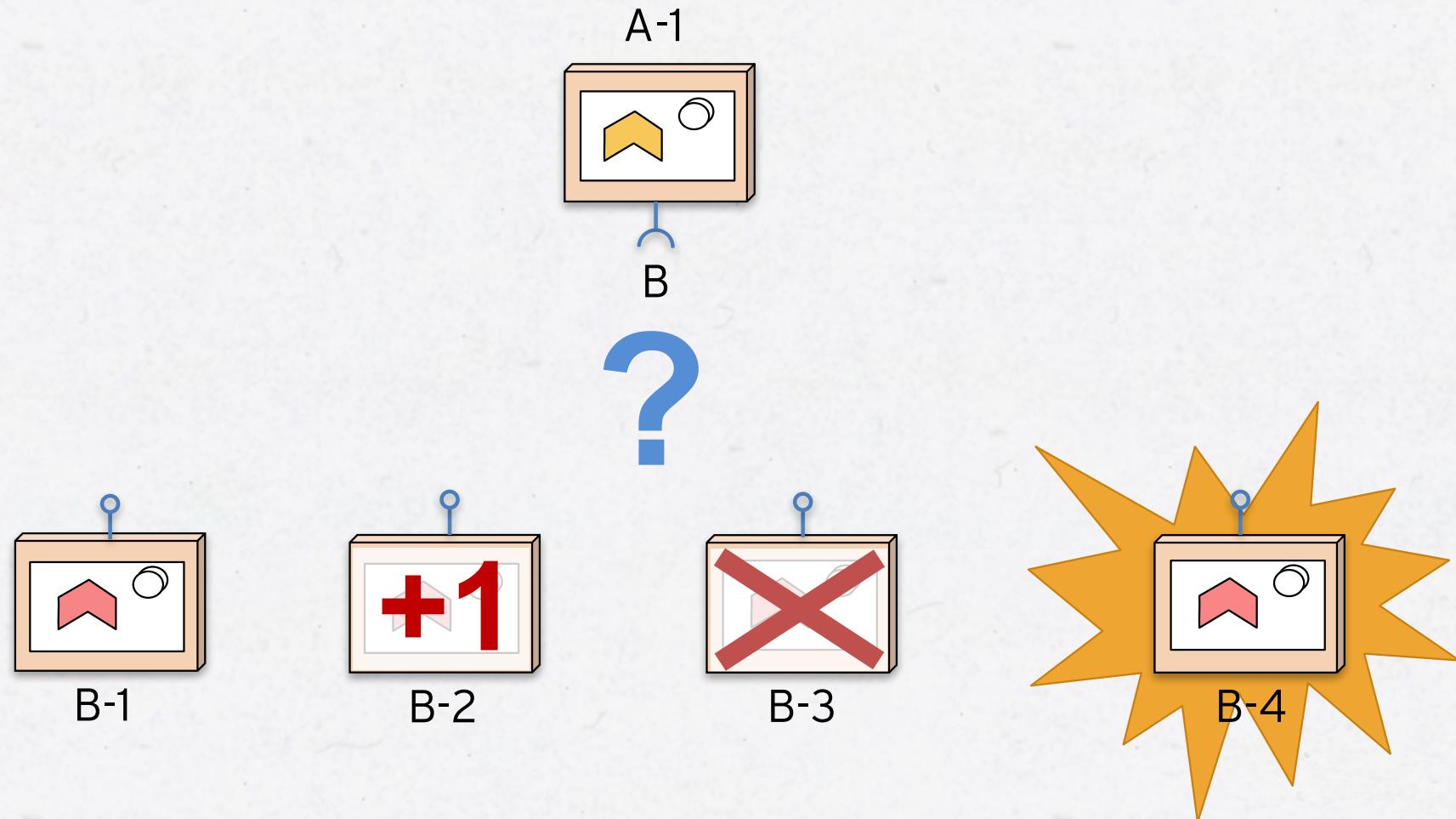
- Can we use Netflix Eureka as Service Discovery when containers are not used?

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?

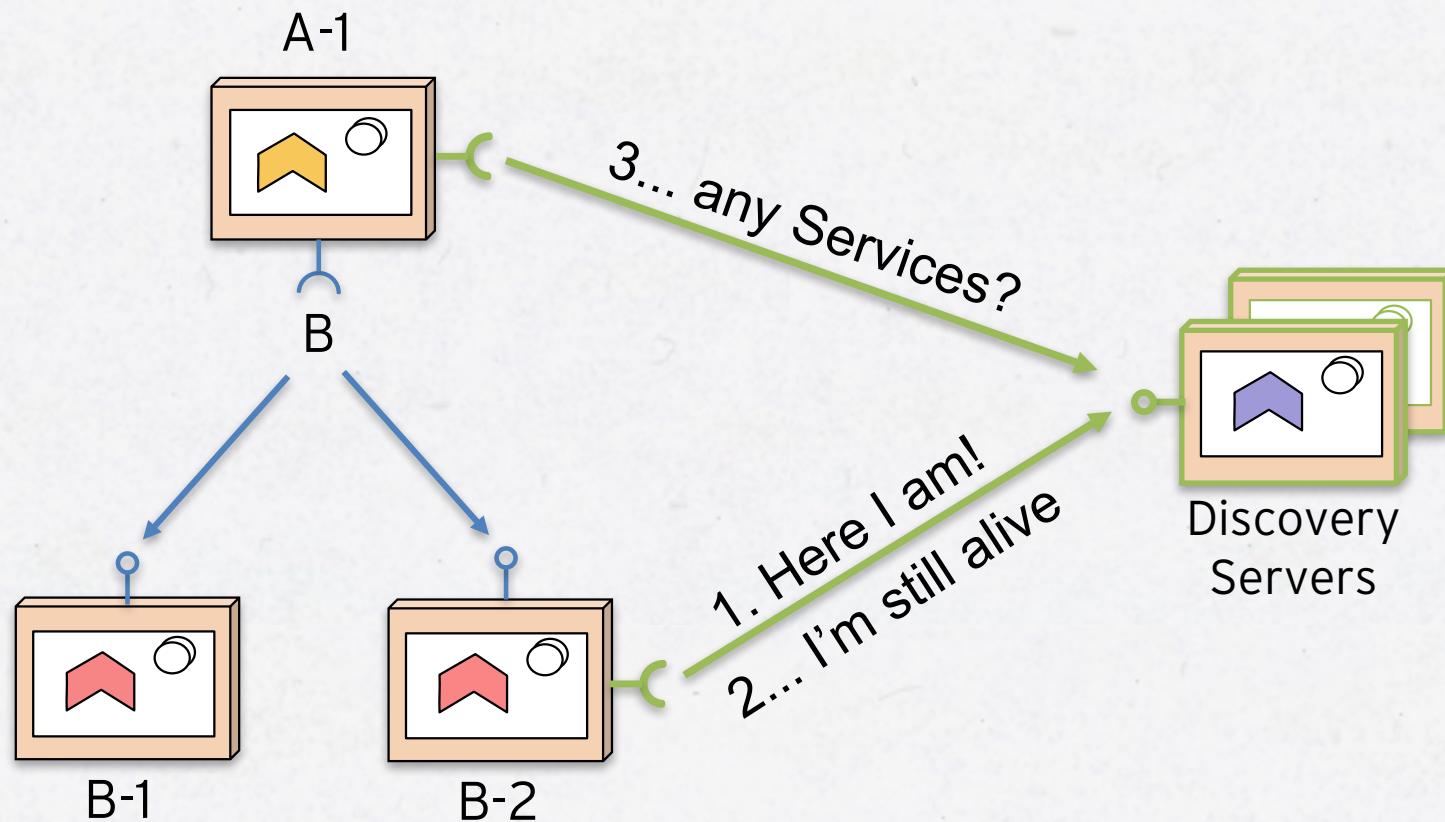
WHAT IS SERVICE DISCOVERY?

Keeping track of many small moving parts...



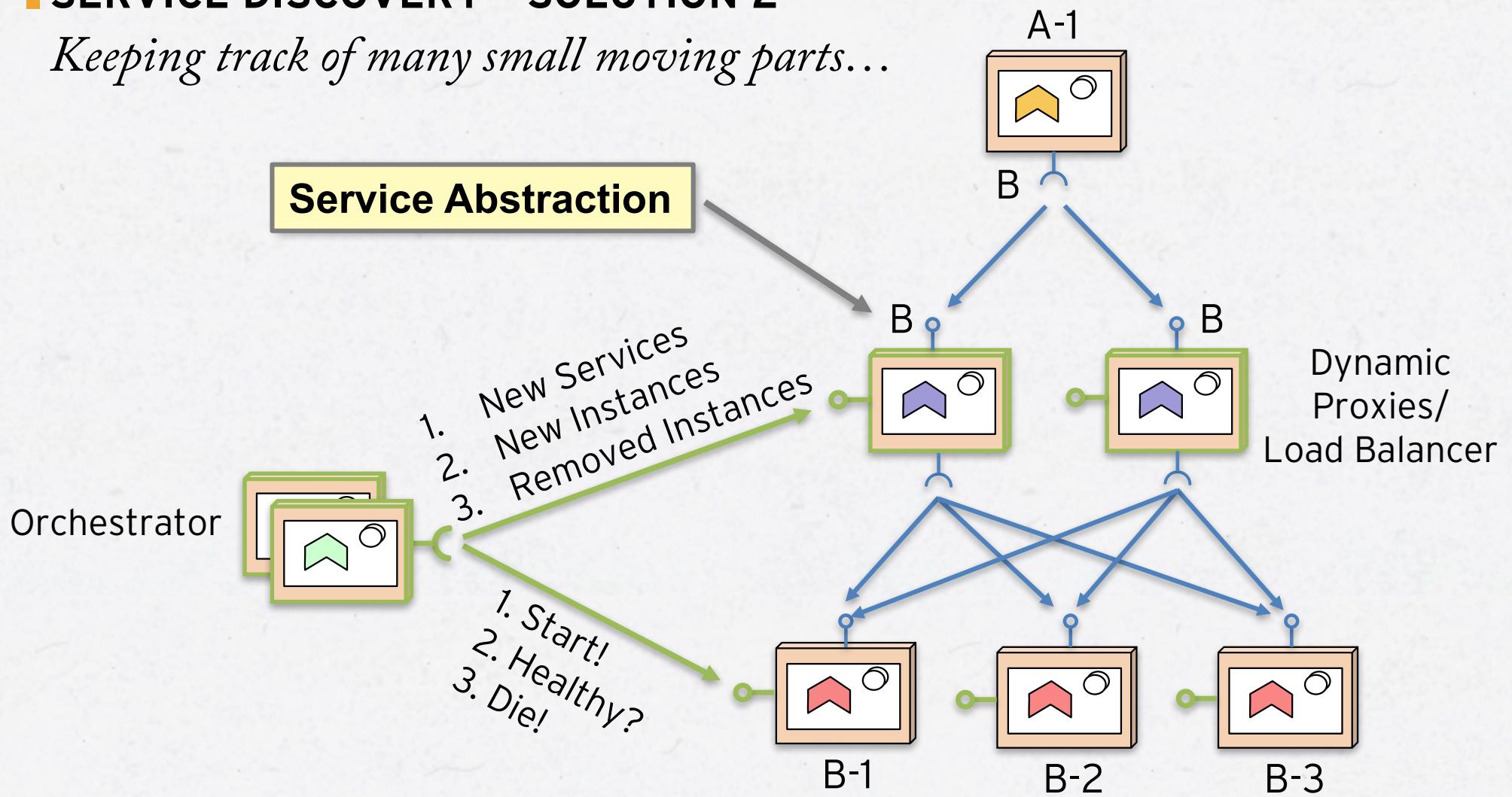
SERVICE DISCOVERY - SOLUTION 1

Keeping track of many small moving parts...



SERVICE DISCOVERY - SOLUTION 2

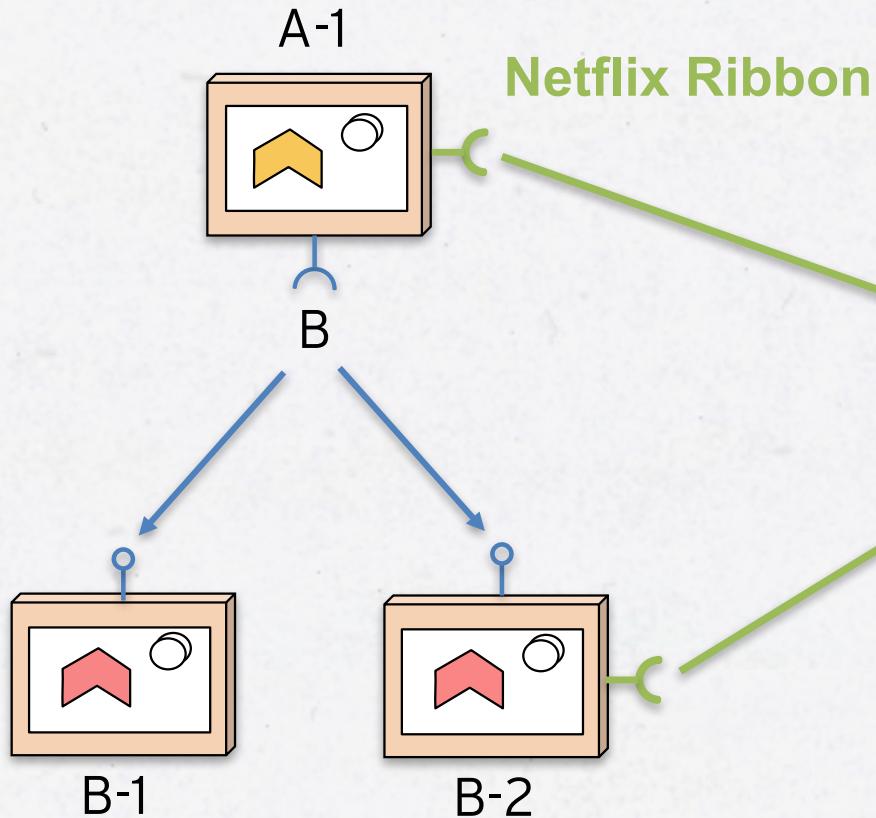
Keeping track of many small moving parts...



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)



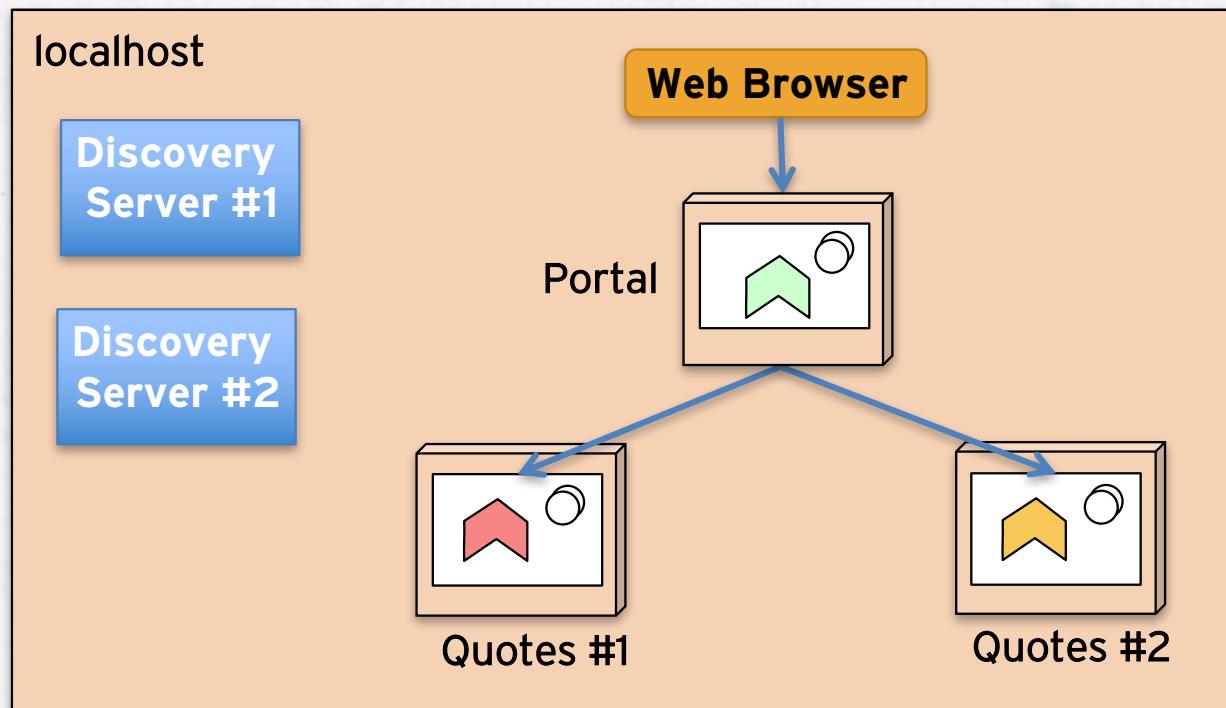
Discovery
Servers
Netflix Eureka

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:

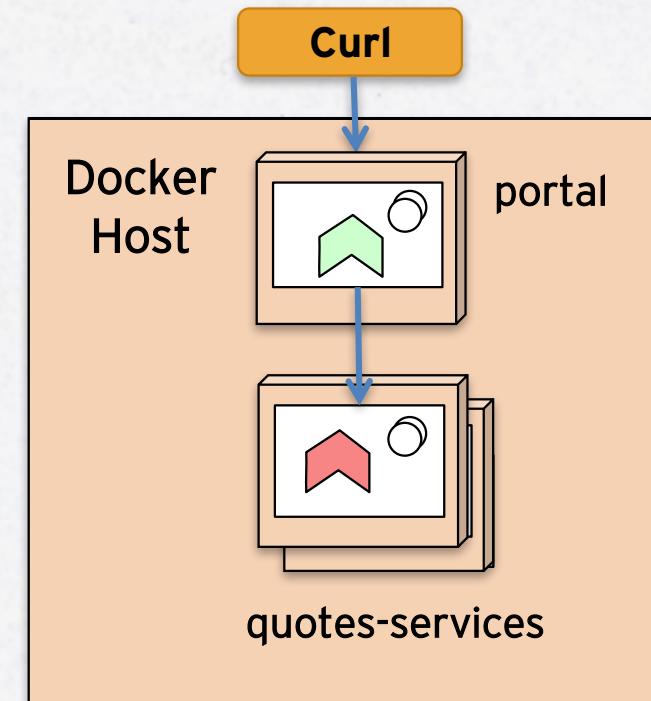
Timestamp	Response Time	Quote	IP Address
2017-01-03 14:16:31 743	355	You, too, Brutus?	Magnus-MBP.lan/192.168.1.19:8080
2017-01-03 14:16:30 736	349	Champagne should be cold, dry and free	Magnus-MBP.lan/192.168.1.19:8081
2017-01-03 14:16:29 719	332	You, too, Brutus?	Magnus-MBP.lan/192.168.1.19:8082
2017-01-03 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 CONTAINER ORCHESTRATION?

- [localhost:9090](#)

2017-01-02 17:26:28 801

2017-01-02 17:26:27 797

2017-01-02 17:26:26 800

2017-01-02 17:26:25 794

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

a4dc9d5c6114 172.19.0.3 8080

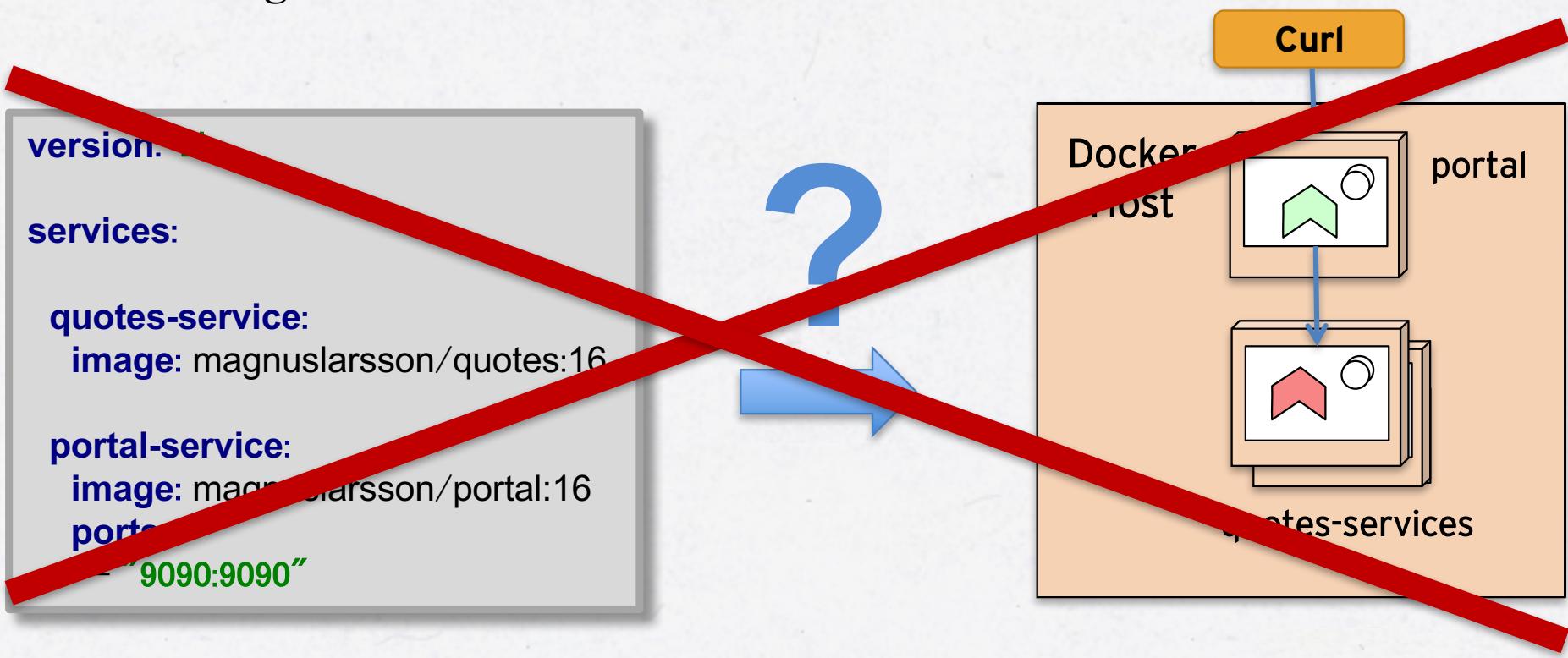
a4dc9d5c6114 172.19.0.3 8080

a4dc9d5c6114 172.19.0.3 8080

a4dc9d5c6114 172.19.0.3 8080

EXAMPLE #2 - PLAIN CONTAINERS (NO ORCHESTRATION)?

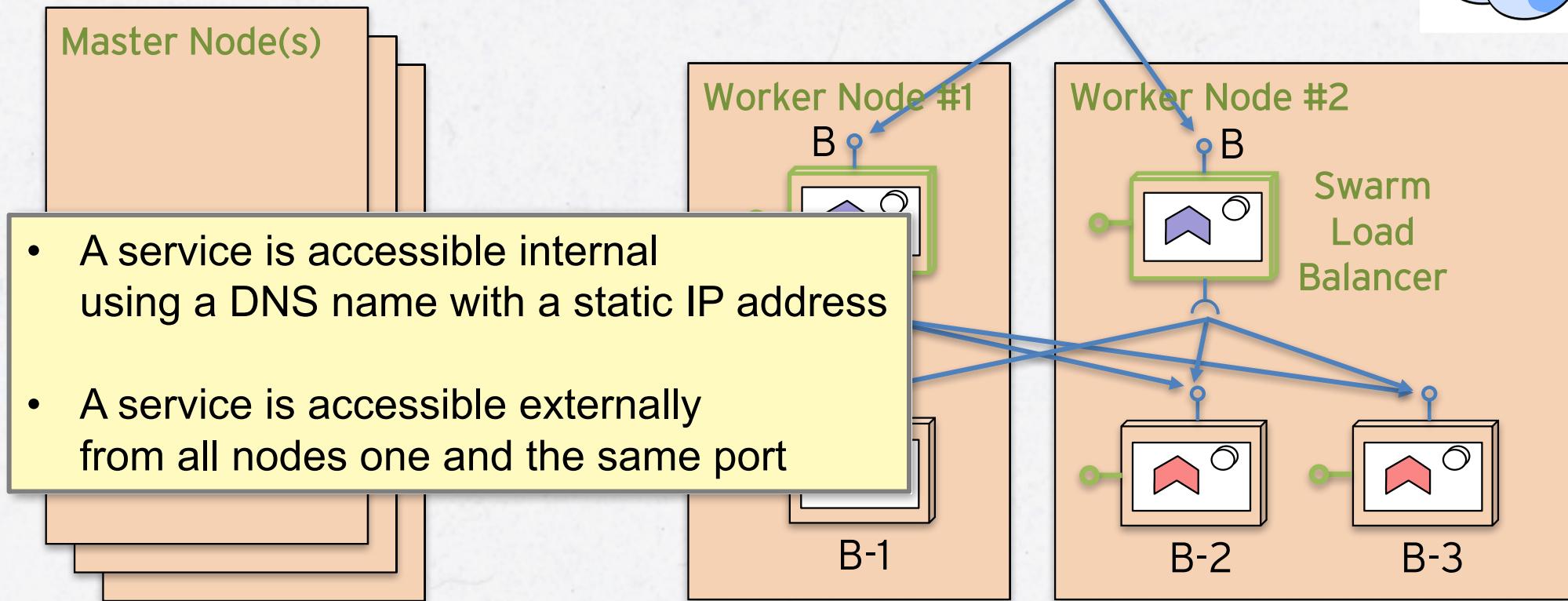
- 2016-02-04: Docker 1.10 released
 - Docker gets a built in DNS server!



EXAMPLE #3: DOCKER SWARM MODE

Docker Services and Routing Mesh

- 2016-07-28: Docker 1.12 released



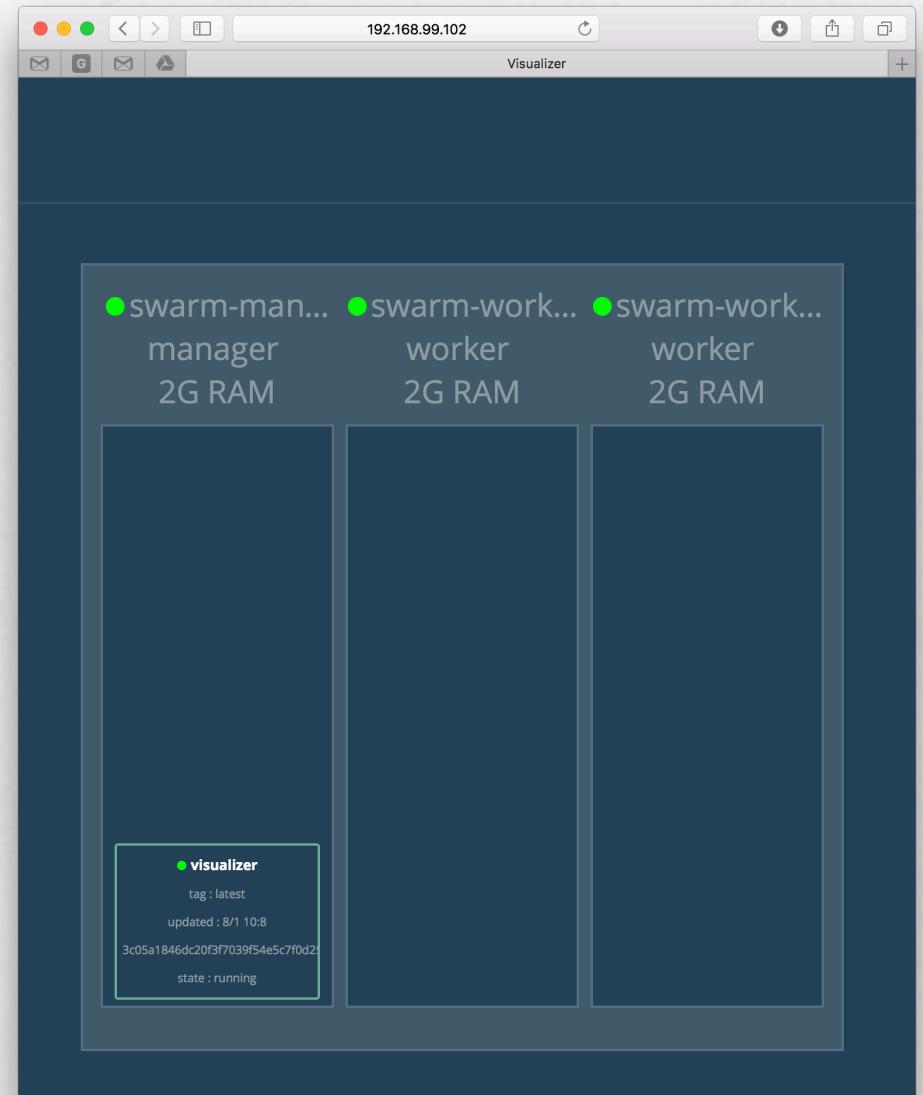
EXAMPLE #3: DOCKER SWARM MODE

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 \
  --constraint node.role==worker

$ 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

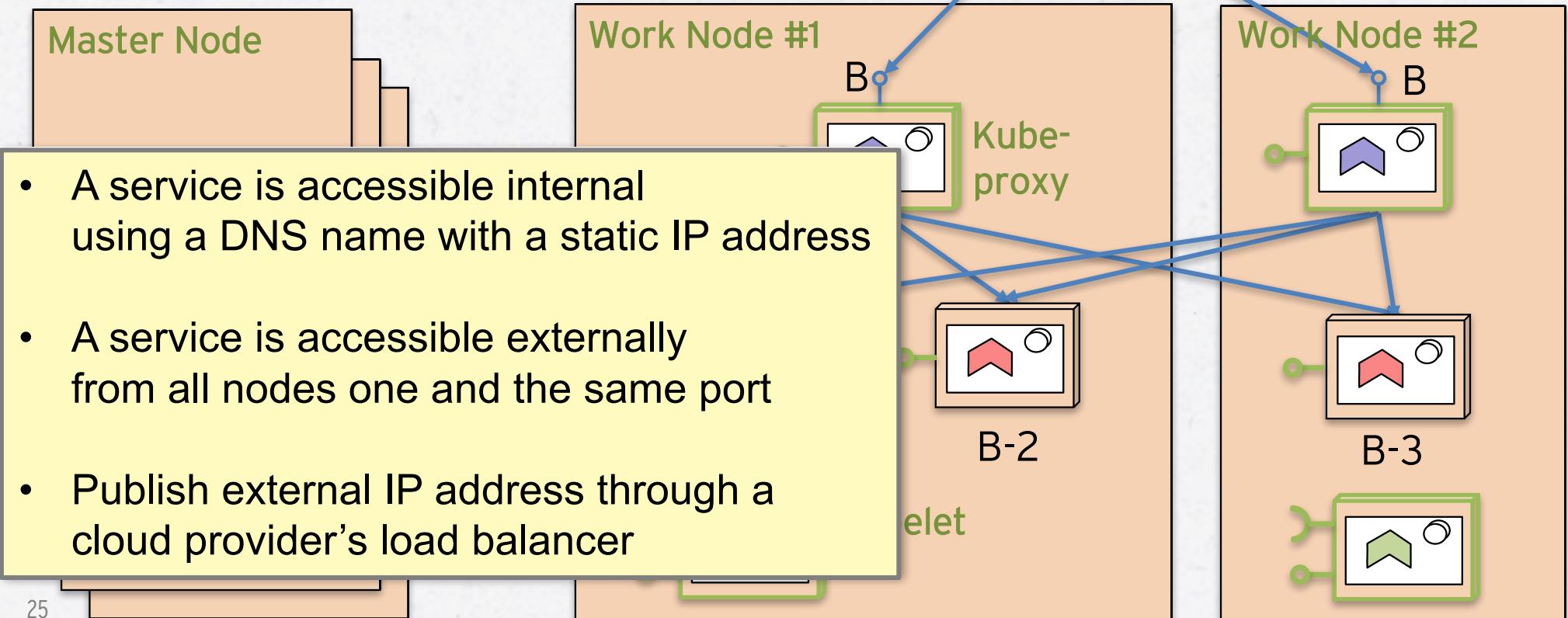


EXAMPLE #4: KUBERNETES

Kubernetes Services



kubernetes



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-s6s14	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

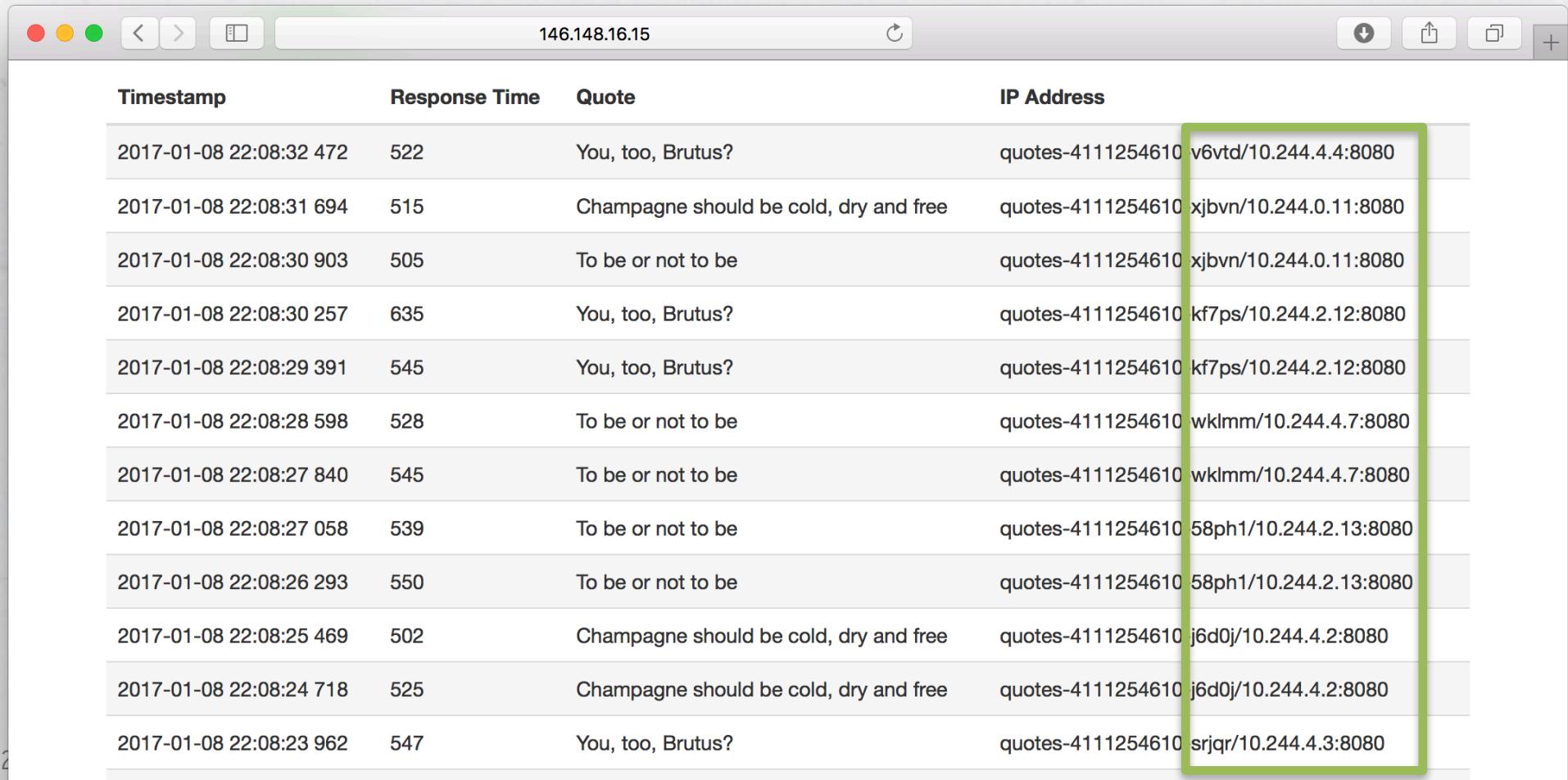
- Any new nodes?

```
$ 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!



A screenshot of a web browser window titled "146.148.16.15". The browser interface includes standard OS X-style buttons (red, yellow, green) and a toolbar with icons for back, forward, and refresh. The main content area displays a table with four columns: "Timestamp", "Response Time", "Quote", and "IP Address". The "IP Address" column is highlighted with a green rectangular selection. The table contains 12 rows of data, each representing a log entry. The "Timestamp" column shows dates and times from January 8, 2017, at 22:08:32 to 22:08:23. The "Response Time" column shows response times in milliseconds (e.g., 472, 522, 547). The "Quote" column contains various quotes, including "You, too, Brutus?", "Champagne should be cold, dry and free", and "To be or not to be". The "IP Address" column shows the pod name followed by its IP address and port (e.g., quotes-4111254610_v6vtd/10.244.4.4:8080).

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

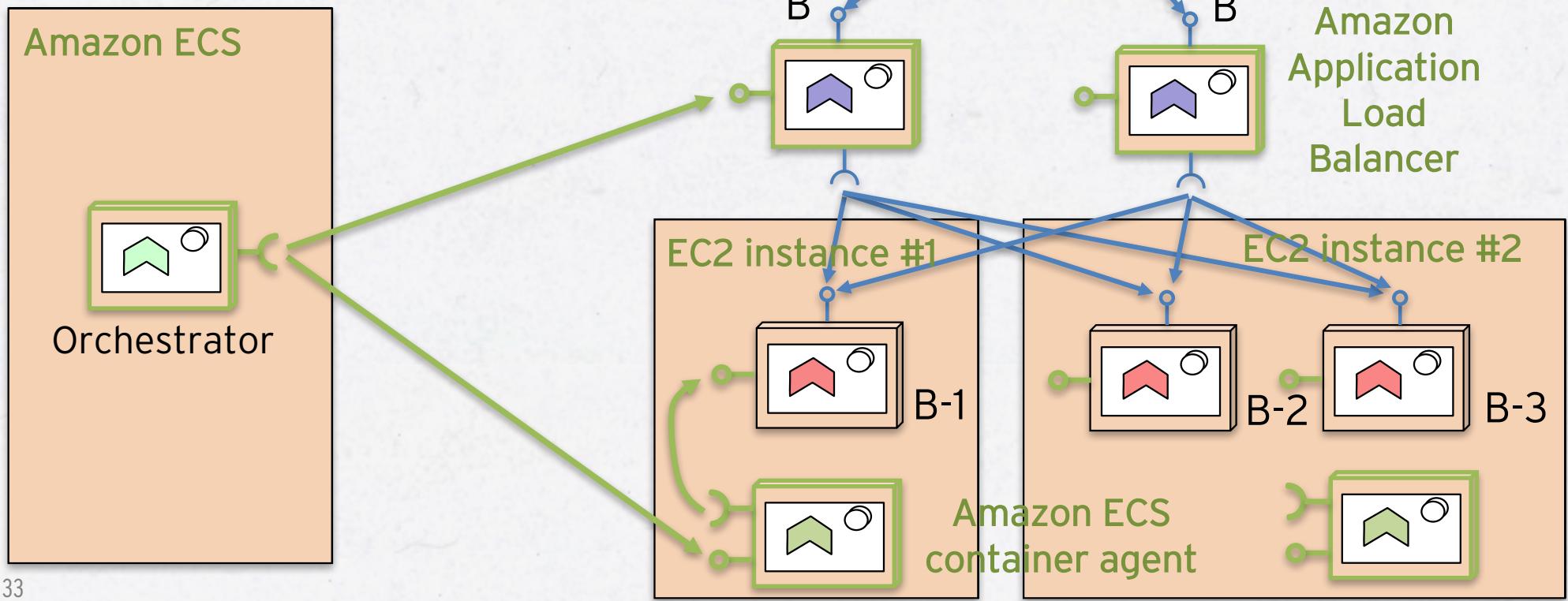
EXAMPLE #5: AMAZON ECS

ECS = EC2 Container Services



Amazon
ECS

- 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

The screenshot shows the AWS CloudWatch Metrics interface. At the top, there's a navigation bar with tabs for 'Services', 'Resource Groups', and a search bar. Below the navigation bar, the left sidebar has 'Amazon ECS' selected, with 'Clusters' highlighted. The main content area shows the 'Clusters' section for 'ecs-ml-cluster'. It displays the cluster status as 'ACTIVE' with 2 registered container instances, 0 pending tasks, and 2 running tasks. A 'Delete Cluster' button is visible. Below this, there are tabs for 'Services', 'Tasks', 'ECS Instances', and 'Metrics', with 'Services' currently selected. A 'Create' button is available. The table below lists two services: 'quotes-service' and 'portal-service', both in an 'ACTIVE' state with 1 desired task and 1 running task. A 'Filter in this page' input field and a 'Viewing 1-2 Services' link are also present.

Service Name	Status	Task Definiti...	Desired tasks	Running tasks
quotes-service	ACTIVE	ecscompose-...	1	1
portal-service	ACTIVE	ecscompose-....	1	1

Feedback English © 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

AMAZON APPLICATION LOAD BALANCER WITH PATH BASED ROUTING

The screenshot shows the AWS Application Load Balancer (ALB) console interface. On the left, a sidebar navigation menu includes options like Snapshots, NETWORK & SECURITY (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces), LOAD BALANCING (Load Balancers, Target Groups), AUTO SCALING (Launch Configurations, Auto Scaling Groups), SYSTEMS MANAGER SERVICES (Run Command, State Manager, Automations, Patch Baselines), and SYSTEMS MANAGER SHARED RESOURCES (Managed Instances). The 'Load Balancers' option is selected and highlighted in orange.

The main content area displays a table of load balancer rules. At the top of the table, there is a search bar labeled 'Filter: Search' and a header row with columns: Name, DNS name, State, VPC ID, Availability Zones, and Type. A single rule is listed:

Name	DNS name	State	VPC ID	Availability Zones	Type
ML-ALB	ML-ALB-1373732302.eu-we...	active	vpc-e0e8b984	eu-west-1a, eu-west-1b	application

Below the table, a note states: "Rules are evaluated in priority order, from the lowest value to the highest value. When the path pattern for a rule is met, traffic is routed to the target group. Other". There are two buttons: "Add rule" and "Reorder rules".

The table of rules has columns: Path pattern, Target group name, Priority, and Rule ARN. The rules listed are:

Path pattern	Target group name	Priority	Rule ARN
/api-other	API	1	arn...ee8dc55447d3b3a...
/api/quote*	quotes-service	2	arn...56b3478dc71e6e68...
/	portal-service	3	arn...28452f25692b436d...
/styles/*	portal-service	4	arn...d345d3ee0c4f054e...
/scripts/*	portal-service	5	arn...c43484f508ad93c8...

At the bottom of the page, there are links for Feedback, English, and a footer with copyright information: "© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved." and links for Privacy Policy and Terms of Use.

AMAZON EC2 INSTANCE (NODES) AUTO SCALING

The screenshot shows the AWS EC2 Auto Scaling console. On the left, a sidebar lists navigation options: EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES (with sub-options Instances, Spot Requests, Reserved Instances, Scheduled Instances, Dedicated Hosts), IMAGES (with sub-options AMIs, Bundle Tasks), ELASTIC BLOCK STORE (with sub-options Volumes, Snapshots), and NETWORK & SECURITY (with sub-options Security Groups, Elastic IPs, Placement Groups). The main content area has tabs for 'Create Auto Scaling group' and 'Actions'. A 'Filter' bar allows searching for Auto Scaling groups. The current view is '1 to 1 of 1 Auto Scaling Groups', specifically 'Auto Scaling Group: amazon-ecs-cli-setup-ecs-ml-cluster-EcsInstanceAsg-JTS0P23QTK09'. Below this, a 'Details' tab is active, followed by 'Activity History' (which is selected), 'Scaling Policies', 'Instances', 'Monitoring', 'Notifications', 'Tags', and 'Scheduled Actions'. An 'Activity History' table displays 194 items, filtered by 'Any Status'. The columns are: Status, Description, Start Time, and End Time. The table entries show a series of successful operations: launching and terminating EC2 instances. The most recent entry is at the bottom.

Status	Description	Start Time	End Time
Successful	Launching a new EC2 instance: i-00e02e5f4593a63f7	2017 January 10 17:47:32 UTC+1	2017 January 10 17:48:05 UTC+1
Successful	Terminating EC2 instance: i-0e3f6b9e0fc421a06	2017 January 10 17:40:16 UTC+1	2017 January 10 17:41:44 UTC+1
Successful	Terminating EC2 instance: i-059c68981807a33c1	2017 January 10 17:34:05 UTC+1	2017 January 10 17:34:48 UTC+1
Successful	Launching a new EC2 instance: i-072bdb91fd87a6295	2017 January 10 17:27:54 UTC+1	2017 January 10 17:28:27 UTC+1
Successful	Launching a new EC2 instance: i-059c68981807a33c1	2017 January 10 17:21:42 UTC+1	2017 January 10 17:22:15 UTC+1
Successful	Terminating EC2 instance: i-0a4ec6c798f7c42d3	2017 January 10 17:14:57 UTC+1	2017 January 10 17:16:21 UTC+1
Successful	Launching a new EC2 instance: i-0e3f6b9e0fc421a06	2017 January 10 17:08:47 UTC+1	2017 January 10 17:09:20 UTC+1
Successful	Terminating EC2 instance: i-04efccc28e8333cc0	2017 January 10 17:02:02 UTC+1	2017 January 10 17:03:06 UTC+1
Successful	Terminating EC2 instance: i-0408e1a9ecch68588	2017 January 10 16:55:19 UTC+1	2017 January 10 16:56:44 UTC+1

AMAZON ECS TASKS (CONTAINERS) AUTO SCALING

The screenshot shows the AWS ECS console interface. The top navigation bar includes 'Services' (selected), 'Resource Groups', and user information ('LARSSON MAGNUS', 'Ireland', 'Support'). The left sidebar has 'Amazon ECS' selected, with 'Clusters' (selected), 'Task Definitions', and 'Repositories' options. The main content area shows the 'Clusters > ecs-ml-cluster > Service: quotes-service' path. The service name 'Service : quotes-service' is displayed with 'Update' and 'Delete' buttons. A tab bar at the top of the main content area includes 'Tasks', 'Events', 'Deployments', 'Auto Scaling' (selected), and 'Metrics'. The 'Auto Scaling' section displays two scaling policies:

Policy Type	Condition	Action
Minimum tasks	CPUUtilization > 25	For alarm: add-new-instance-medium-cpu-alarm Take the action: Add 1 tasks when 25 <= CPUUtilization
Maximum tasks	CPUUtilization < 10	For alarm: low-cpu-usage Take the action: Remove 1 tasks when 10 >= CPUUtilization
	CPUUtilization > 50	For alarm: cpu-over-50-pct Take the action: Add 1 tasks when 50 <= CPUUtilization

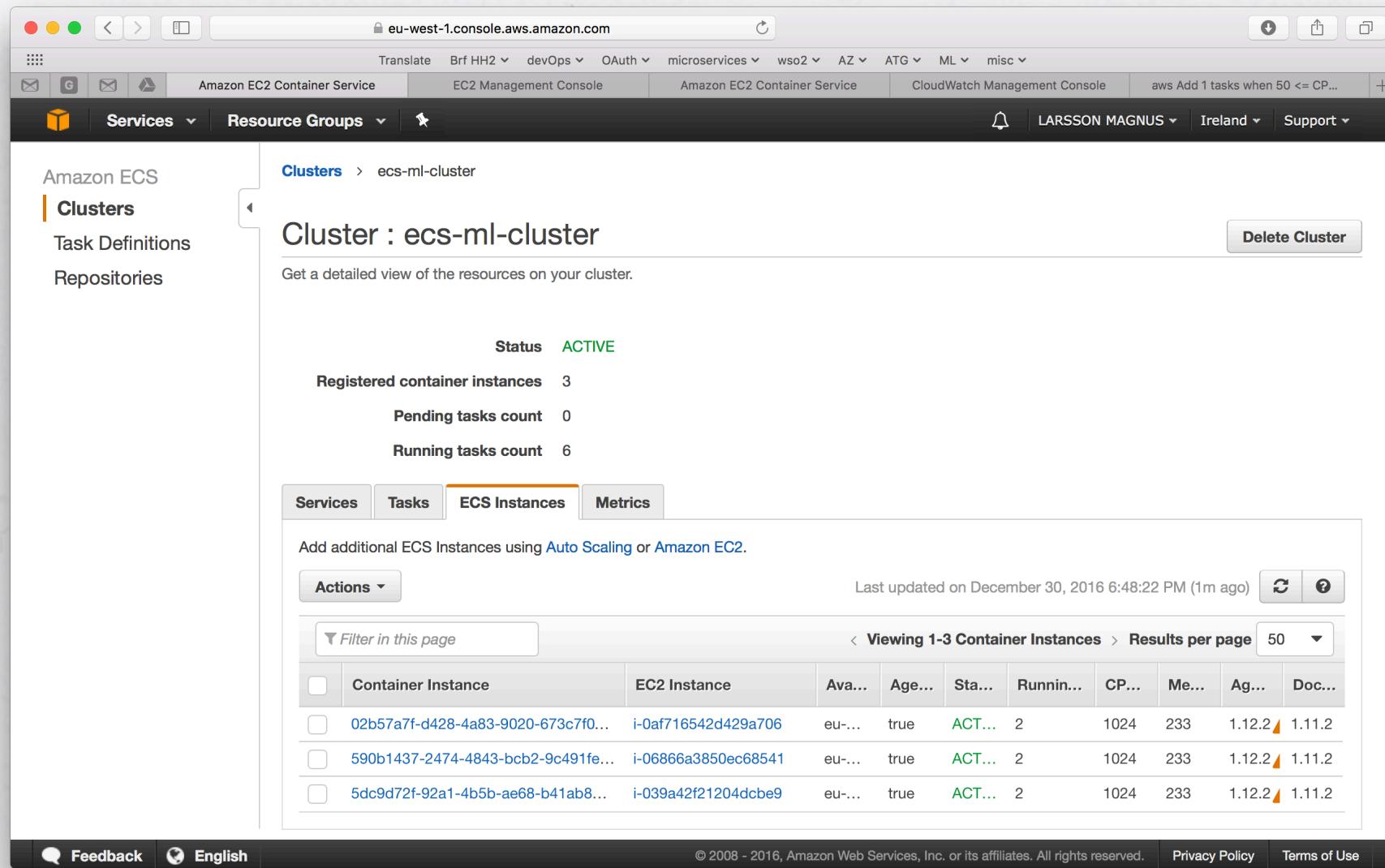
At the bottom of the page are links for 'Feedback', 'English', 'Privacy Policy', and 'Terms of Use'.

NEW ECC TASKS (CONTAINERS) AUTO CREATED UNDER LOAD

The screenshot shows the AWS ECS console interface. The top navigation bar includes links for Translate, Brf HH2, devOps, OAuth, microservices, wso2, AZ, ATG, ML, and misc. The main menu on the left has 'Amazon ECS' selected, with 'Clusters' currently active. The breadcrumb navigation shows 'Clusters > ecs-ml-cluster > Service: quotes-service'. The main content area is titled 'Service : quotes-service' with 'Update' and 'Delete' buttons. Below this, a table lists four tasks under the 'Tasks' tab. The table columns are Task, Task Definition, Group, Last status, and Desired status. All tasks are listed as 'Running'.

Task	Task Definition	Group	Last status	Desired status
70b2129e-e807-4aa5-9...	ecscompose-quotes:4	service:quotes-service	RUNNING	RUNNING
9c11a714-f1c7-449d-bf...	ecscompose-quotes:4	service:quotes-service	RUNNING	RUNNING
abb8dd1f-ed33-44b8-8...	ecscompose-quotes:4	service:quotes-service	RUNNING	RUNNING
c11173dc-b1a4-4b75-b...	ecscompose-quotes:4	service:quotes-service	RUNNING	RUNNING

NEW EC2 INSTANCES (NODES) AUTO CREATED UNDER LOAD



The screenshot shows the Amazon EC2 Container Service Cluster details page for the cluster 'ecs-ml-cluster'. The cluster is active and has three registered container instances. There are currently no pending tasks, but six tasks are running. The 'ECS Instances' tab is selected, displaying a table of three container instances. Each instance is associated with a specific EC2 instance ID, availability zone, age, status, and other metrics like CPU usage and memory.

Container Instance	EC2 Instance	Ava...	Age...	Sta...	Runnin...	CP...	Me...	Ag...	Doc...
02b57a7f-d428-4a83-9020-673c7f0...	i-0af716542d429a706	eu...	true	ACT...	2	1024	233	1.12.2	1.11.2
590b1437-2474-4843-bcb2-9c491fe...	i-06866a3850ec68541	eu...	true	ACT...	2	1024	233	1.12.2	1.11.2
5dc9d72f-92a1-4b5b-ae68-b41ab8...	i-039a42f21204dcbe9	eu...	true	ACT...	2	1024	233	1.12.2	1.11.2

EXAMPLE #5: AMAZON ECS

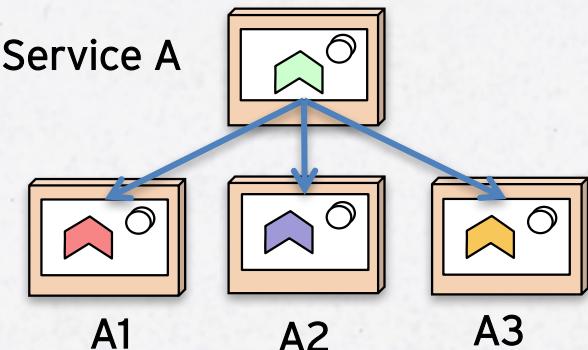
- Portal uses new tasks!

The screenshot shows a web browser window titled "Quotes Portal" with the URL "ml-alb-1373732302.eu-west-1.elb.amazonaws.com". The page has a header with "Request interval, ms" (set to 853), "Strength" (set to 13), and "Waiting requests" (set to 0). Below this is a button group with "One Request" and a green "Start" button. The main content is a table with columns: "Timestamp", "Response Time", "Quote", and "IP Address". The "IP Address" column for all rows is highlighted with a green border. The data in the table is as follows:

Timestamp	Response Time	Quote	IP Address
2017-01-12 11:11:40 359	981	Champagne should be cold, dry and free	d33a567bf867/ 72.17.0.3:8080
2017-01-12 11:11:39 701	1176	To be or not to be	e8b40d1ca860/ 72.17.0.3:8080
2017-01-12 11:11:38 875	1204	You, too, Brutus?	976b1dc9a926/ 72.17.0.2:8080
2017-01-12 11:11:37 791	973	You, too, Brutus?	d33a567bf867/ 72.17.0.3:8080
2017-01-12 11:11:37 112	1148	To be or not to be	e8b40d1ca860/ 72.17.0.3:8080
2017-01-12 11:11:36 256	1146	You, too, Brutus?	976b1dc9a926/ 72.17.0.2:8080
2017-01-12 11:11:35 236	979	To be or not to be	d33a567bf867/ 72.17.0.3:8080
2017-01-12 11:11:34 549	1146	Champagne should be cold, dry and free	e8b40d1ca860/ 72.17.0.3:8080

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

