Title: Cloud-based E-commerce shopping application
Members: Arpeet Desai, Tsung-Min Huang
Team: 5

Task to tackle

In our project, our goal is to develop cloud native e-commerce application that can be easily scaled and handle large number of user requests. To achieve this, our e-commerce application will use micro-service architecture and deployed with container deployment and orchestration technique known as Docker Swarm on top of AWS.

Functionalities

- Login service for User as well as Seller
- Email service for User/Seller
- Inventory service by sellers
- Searching products for a user from multiple sellers.
- Customer Review for Products.
- Checkout and Past Shopping History data for users.

Solution Design

Planned: For implementing the application as micro-service architecture, each component will be deployed and ran in a separate container. Each component has its own database to store data and will communicate via RESTful API. To handle the surge of incoming requests, we plan to use message queue service as a bridge to hold and dispatch the request. We plan to use a load balancer to distribute requests to multiple web servers. To cope with many requests, cause a heavy load to a single node, we will deploy with container orchestration technique such as Docker Swarm to scale a number of containers and manage a large number of containers. We plan to design our application using Java Spring Boot and database using MongoDB.

Implementation: We designed our e-commerce application with components such as customer service, product service, email service, review service and each service provides RESTful API for communication. To overcome the complexity and logic business between front-end and back-end, gateway service was implemented. All the services are implemented as containers. Instead of using different database containers for all services, we use same MongoDB setup which has 3 MongoDB database containers set up in a replica mode. We use docker swarm on AWS for deploying our application. Our swarm is setup with 6 EC2 instances. 3 EC2 as managers and 3 EC2 as worker nodes. We use multiple managers nodes for redundancy purpose. We have an ELB setup on our swarm with a DNS (*E-commerc-External-1CLBG7QGVLRTF-1802061480.us-west-1.elb.amazonaws.com*). Whenever a request to this URL is made, ELB will route the request to different containers to distribute the load.

Project Architecture Diagram

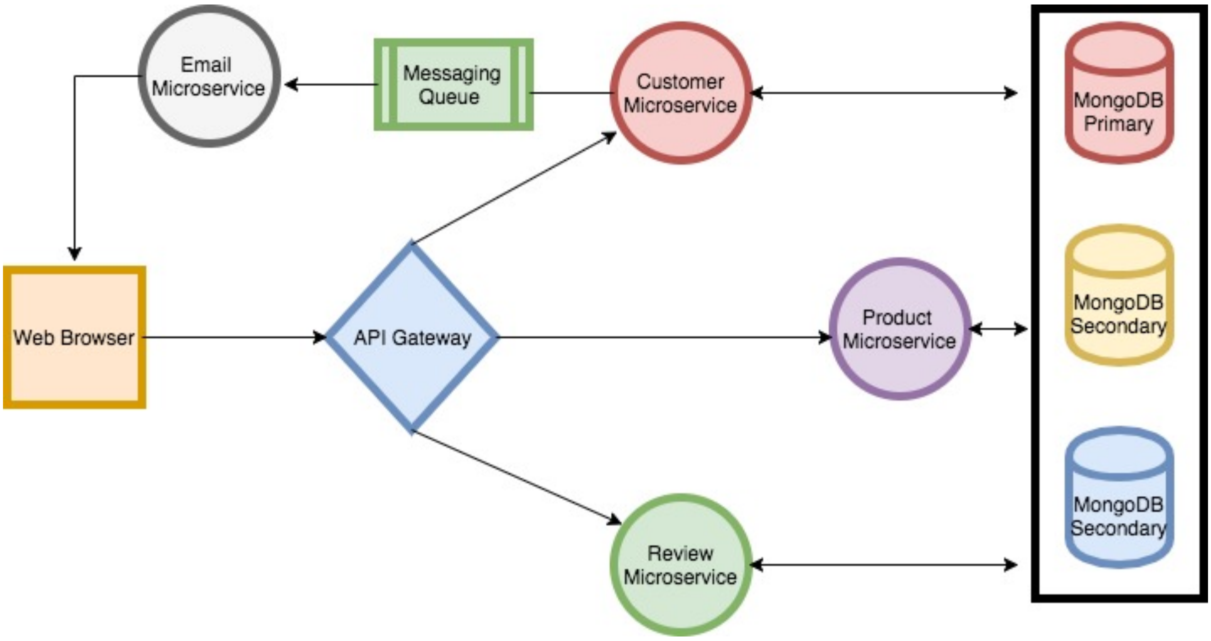


Fig 1. Project Architecture Diagram

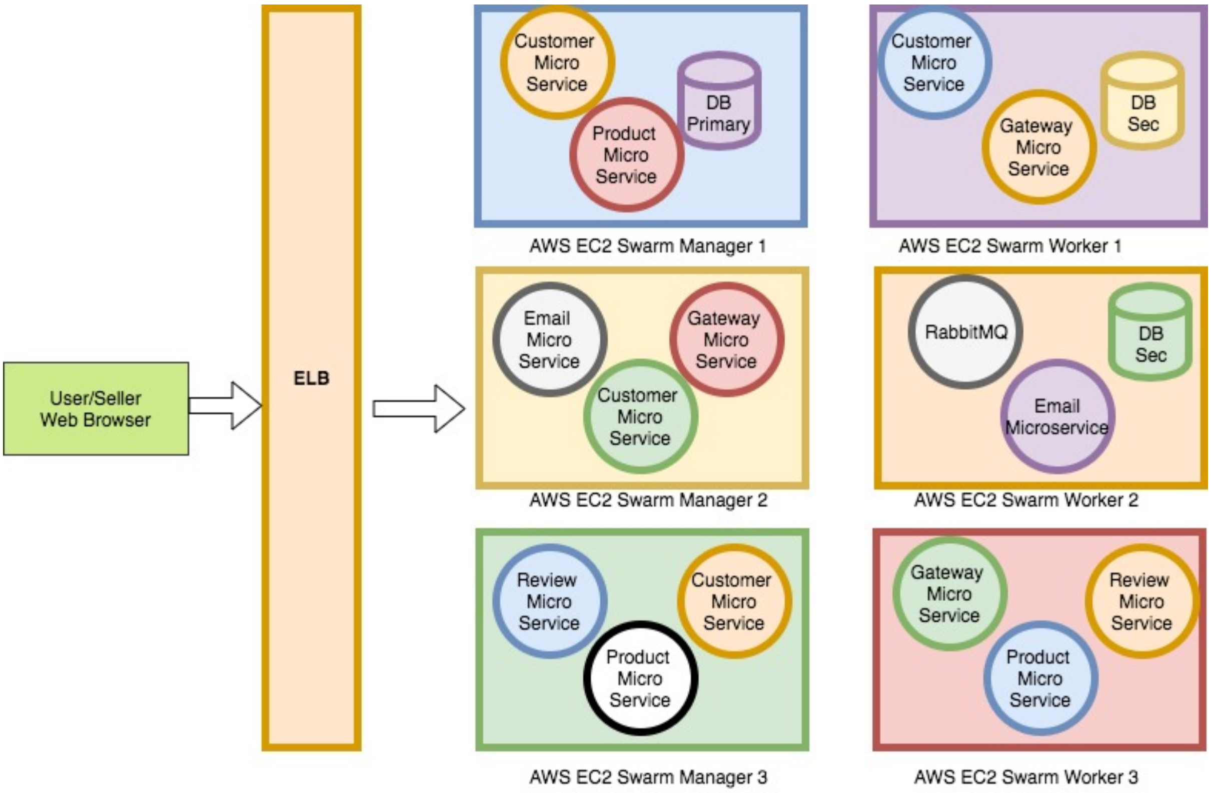


Fig 2. Micro service deployment on AWS with Docker Swarm

Database Tier

- NoSQL: Used MongoDB for customer, product & review data store

Middle Tier

api-gateway service: The opened entry point to handle all the request and forward to other micro-service.

customer service: customer service to handle to request from gateway api and respond these request and save customer data to data store component. Customer service also sends email task request to message queue.

- Get list of customers
- Create a new customer
- Authenticate a customer
- Send email request to message queue

email service: email service listen message from message queue and serve these email task.

- Send email to new register customer
- Send email checkout order to customer (not implemented)

product service: product service respond to the gateway api request and do operations on product and save product data to data store component.

- Get list of products
- Create a new product
- Search product by keyword
- Delete a product
- Decrease a product stock

Review service: Review service process the request forward from gateway api to

- Create a product average review
- Get a product average review

Architecture Design patterns

- micro-services architecture pattern

Application Design Patterns

- Decomposition: As a Distributed application our e-commerce application is divided in 3-layer: front-end (UI), backend (process component) and data storage. UI only access the API exposed by gateway service, and gateway forward request to correspond micro service component. Micro-service manipulates the data store when necessary.

- Workload: The e-commerce application is expected have continuously change workload, each micro-service instances deployed with multi-instances and managed with Docker-Swarm and hence application containers can be scaled easily.

- Data: For all the micro-service component, customer, product, review, email process component all stateless. The data component used MongoDB as data store is state-full.

- Communication: For import request like send email service, we used RabbitMQ to hold and dispatch message to guaranteed send email request will be served. And other components communication is through RESTful API call.

- Scalability: Auto Scaling group is created to take care of scalability of both Manager Nodes as well as worker nodes. Scalability of micro services is taken care by the Docker Swarm. Whenever we have one or more service container going down, swarm will recreate the service container.

- Availability: As we have used AWS to deploy our cloud native application, we have good availability for node instances. Availability of services is taken care by Docker Swarm.

- Load Balancing: Load balancing is done by AWS ELB. There is a DNS address given to access different services on swarm and ELB will load balance request to different containers.

Final list of functionalities/operations (and, if different from the proposal, status of each planned one)

- Login service for user – implemented
- Email service for new registerd user - implemented
- Search products for user with keyword - implemented
- Customer review for product – implemented
- Checkout and Past Shopping History data for users - not implemented
- Deploying on AWS - implemented
- Docker Swarm for container orchestration - implemented
- Load balancer - implemented (AWS ELB)
- AWS - Auto scaling - implemented
- AWS SQS - not implemented
- AWS ECS - not implemented

Project URL

E-commerc-External-1CLBG7QGVLRTF-1802061480.us-west-1.elb.amazonaws.com:8080/gateway/product

E-commerc-External-1CLBG7QGVLRTF-1802061480.us-west-1.elb.amazonaws.com:8080/gateway/customer

E-commerc-External-1CLBG7QGVLRTF-1802061480.us-west-1.elb.amazonaws.com:8080/gateway/review

Existing cloud service/feature leveraged

- AWS EC2 to launch instances.
- AWS Cloudformation create stack template to create a Docker Swarm configuration with 3 Managers and 3 Workers. Template chooses EC2 instance specifically created for Docker with Docker Engine setup very well and with all the docker process running for swarm.
- AWS Auto Scaling group.
- AWS ELB to load balance between multiple containers in swarm.
- Docker Swarm for container orchestration, container scalability and availability.

Technology Stack

- Platform: AWS EC2, Docker
- Technologies: MongoDB, Docker Swarm, ELB, RabbitMQ.
- Frameworks: Spring boot, REST.
- Languages: Java

Design Trade-off

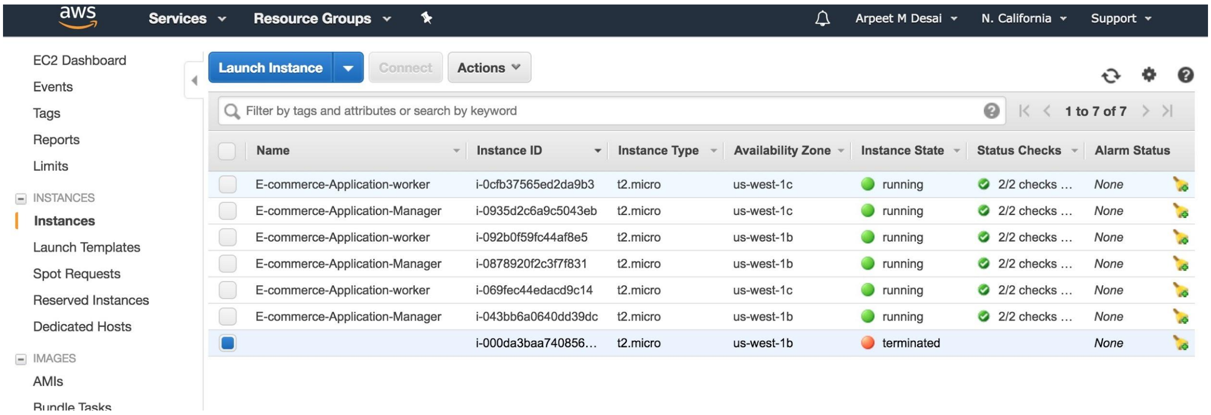
The communication between component usually done by message queue. Using message queue as bridge between components had several benefit. One is while receiver component is busy to handle request, message queue can serve as buffer to store the sender request. The other benefit is message queue can guarantee the request will be delivered between sender and receiver. However, in our implementation our most communication is very light-weight and we use auto-balancer to distribute the requests to multiple component instances. For each email notification, we want make sure its must be handled and send email to customer, we keep message queue between email component and customer component.

Which component are in which containers

- Product Microservice component is in product_microservice container.
- Customer Microservice is in cutomer_microservice container.
- Review Microservice is in review_microservice container.
- Email Microservice is in email_microservice container.
- MongoDB database with 3 replicas setup are in mongo1, mongo2 and mongo3 container.
- On top of that as we are running everything in Docker Swarm, swarm will launch different containers on different EC2 instances.

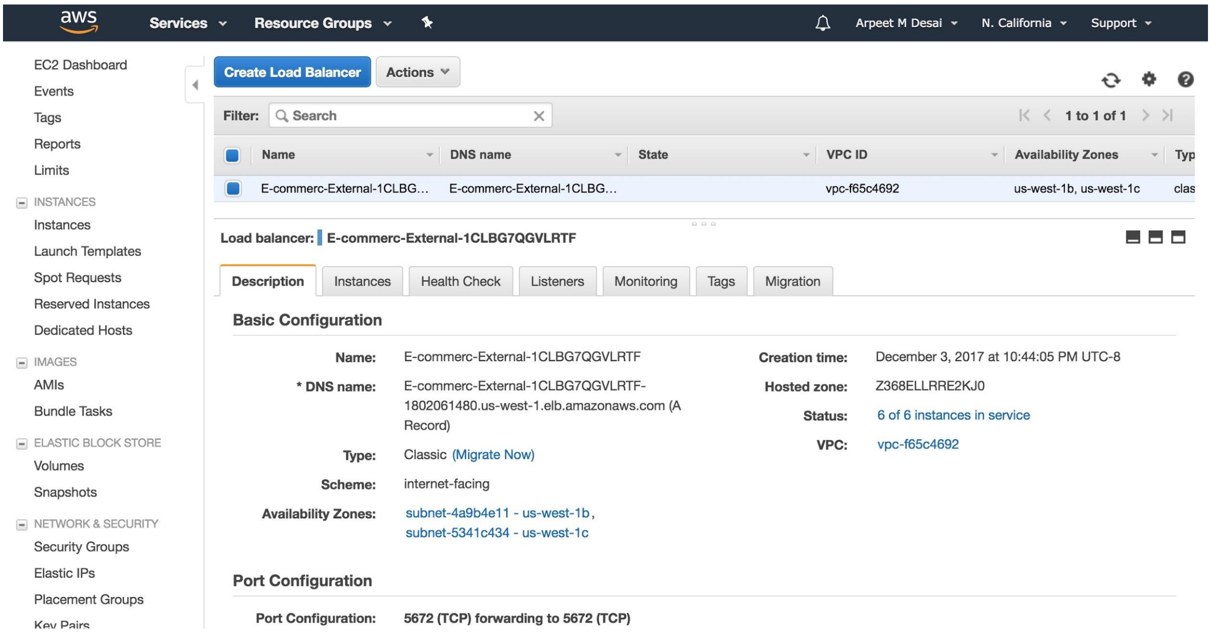
Sample execution (screen shots, etc.) for important functionalities / operations

Screenshot for AWS EC2 instances



□	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
□	E-commerce-Application-worker	i-0cfb37565ed2da9b3	t2.micro	us-west-1c	● running	✓ 2/2 checks ...	None
□	E-commerce-Application-Manager	i-0935d2c6a9c5043eb	t2.micro	us-west-1c	● running	✓ 2/2 checks ...	None
□	E-commerce-Application-worker	i-092b0f59fc44af8e5	t2.micro	us-west-1b	● running	✓ 2/2 checks ...	None
□	E-commerce-Application-Manager	i-0878920f2c3f7f831	t2.micro	us-west-1b	● running	✓ 2/2 checks ...	None
□	E-commerce-Application-worker	i-069fec44edacd9c14	t2.micro	us-west-1c	● running	✓ 2/2 checks ...	None
□	E-commerce-Application-Manager	i-043bb6a0640dd39dc	t2.micro	us-west-1b	● running	✓ 2/2 checks ...	None
☑		i-000da3baa740856...	t2.micro	us-west-1b	● terminated		None

Screenshot for AWS ELB (Elastic load balancing)



Name	DNS name	State	VPC ID	Availability Zones	Typ
E-commerc-External-1CLBG…	E-commerc-External-1CLBG…		vpc-f65c4692	us-west-1b, us-west-1c	clas

Load balancer: E-commerc-External-1CLBG7QGVLRTF

Basic Configuration

Name: E-commerc-External-1CLBG7QGVLRTF

* DNS name: E-commerc-External-1CLBG7QGVLRTF-1802061480.us-west-1.elb.amazonaws.com (A Record)

Type: Classic (Migrate Now)

Scheme: internet-facing

Availability Zones: subnet-4a9b4e11 - us-west-1b, subnet-5341c434 - us-west-1c

Creation time: December 3, 2017 at 10:44:05 PM UTC-8

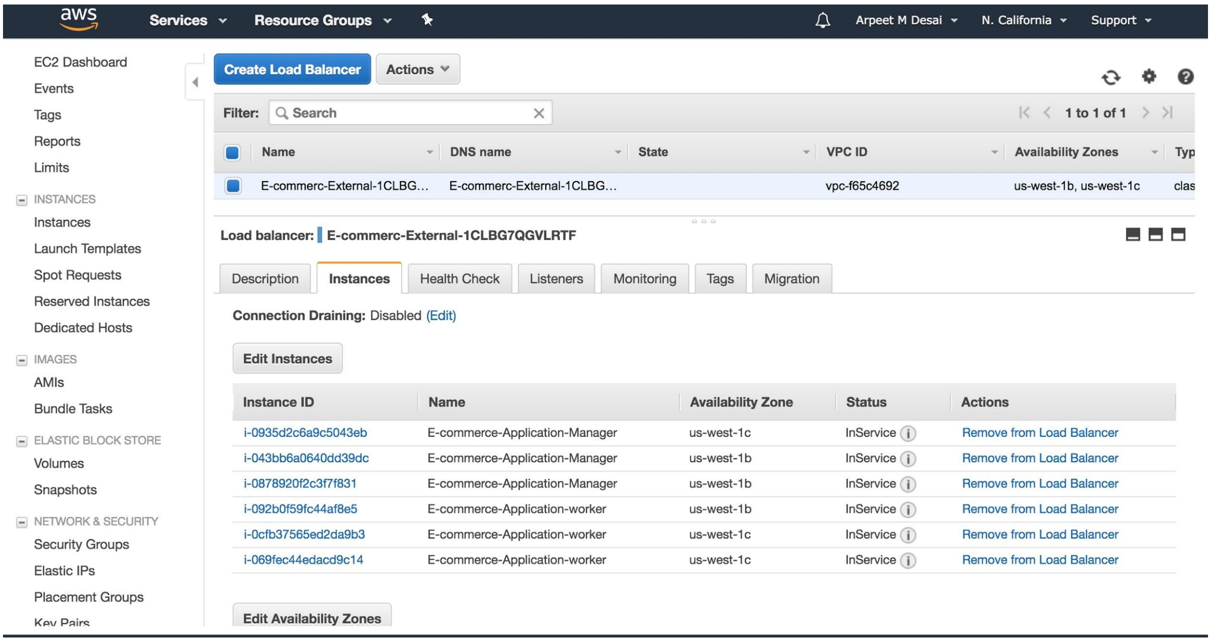
Hosted zone: Z368ELLRRE2KJ0

Status: 6 of 6 instances in service

VPC: vpc-f65c4692

Port Configuration

Port Configuration: 5672 (TCP) forwarding to 5672 (TCP)



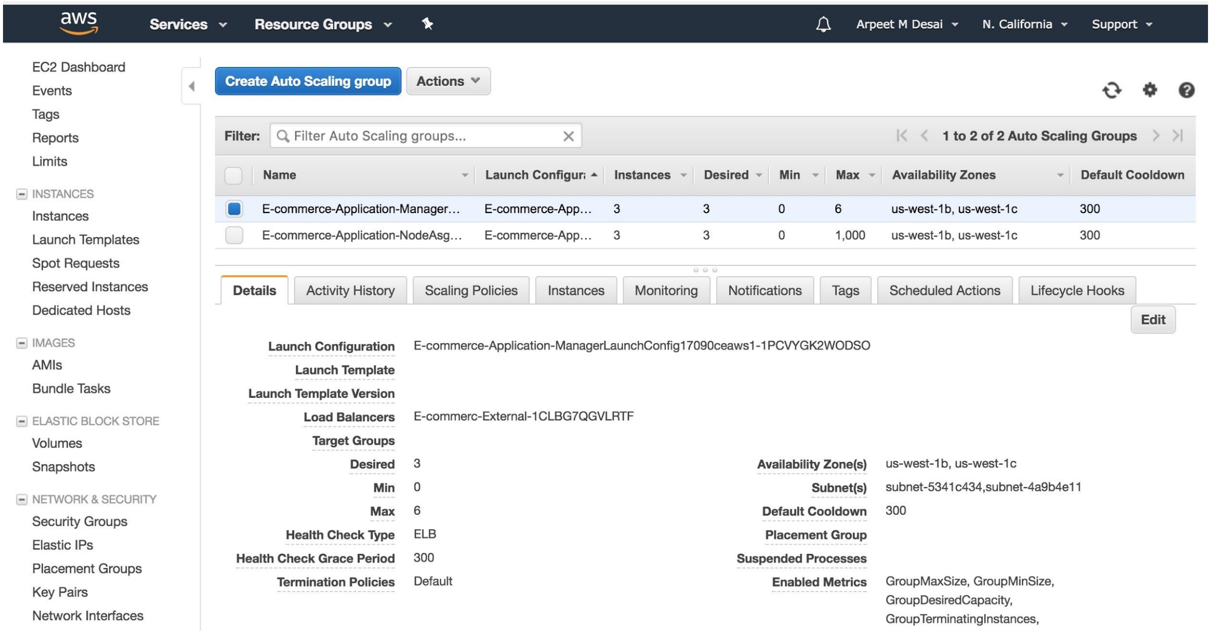
Name	DNS name	State	VPC ID	Availability Zones	Typ
E-commerc-External-1CLBG…	E-commerc-External-1CLBG…		vpc-f65c4692	us-west-1b, us-west-1c	clas

Load balancer: E-commerc-External-1CLBG7QGVLRTF

Connection Draining: Disabled (Edit)

Instance ID	Name	Availability Zone	Status	Actions
i-0935d2c6a9c5043eb	E-commerce-Application-Manager	us-west-1c	InService	Remove from Load Balancer
i-043bb6a0640dd39dc	E-commerce-Application-Manager	us-west-1b	InService	Remove from Load Balancer
i-0878920f2c3f7f831	E-commerce-Application-Manager	us-west-1b	InService	Remove from Load Balancer
i-092b0f59fc44af8e5	E-commerce-Application-worker	us-west-1b	InService	Remove from Load Balancer
i-0cfb37565ed2da9b3	E-commerce-Application-worker	us-west-1c	InService	Remove from Load Balancer
i-069fec44edacd9c14	E-commerce-Application-worker	us-west-1c	InService	Remove from Load Balancer

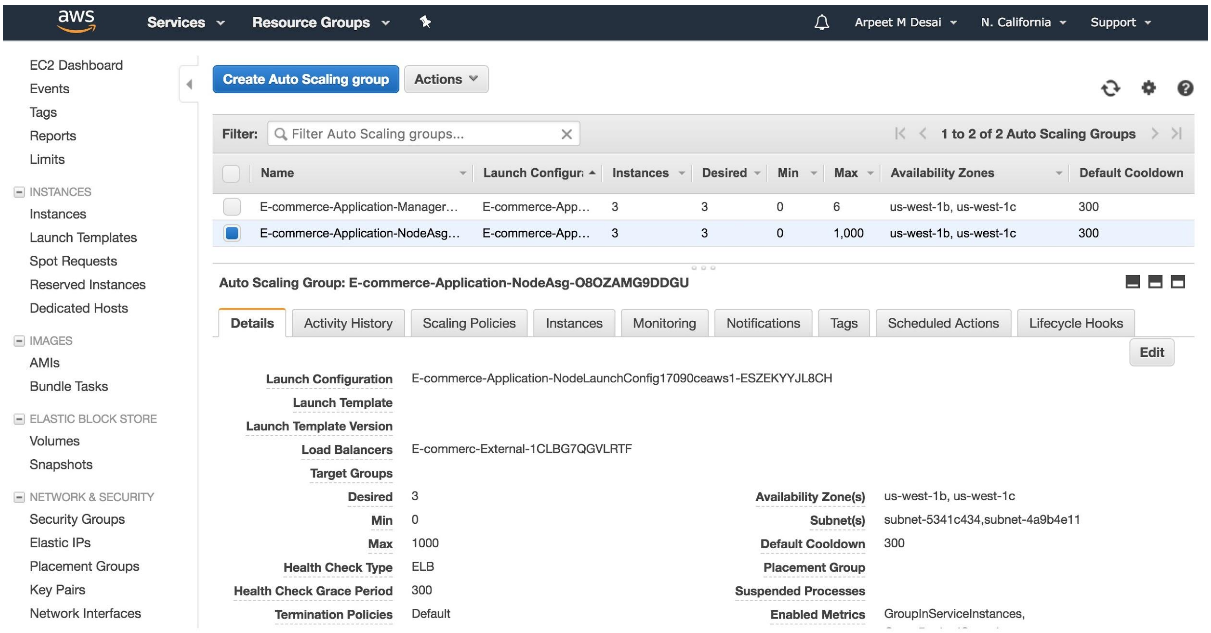
Screenshot for AWS Auto scaling group for Manager as well as worker nodes



□	Name	Launch Configur	Instances	Desired	Min	Max	Availability Zones	Default Cooldown
☑	E-commerce-Application-Manager…	E-commerce-App…	3	3	0	6	us-west-1b, us-west-1c	300
□	E-commerce-Application-NodeAsg…	E-commerce-App…	3	3	0	1,000	us-west-1b, us-west-1c	300

Launch Configuration E-commerce-Application-ManagerLaunchConfig17090ceaws1-1PCVYGK2WODSO
Launch Template
Launch Template Version
Load Balancers E-commerc-External-1CLBG7QGVLRTF
Target Groups
Desired 3
Min 0
Max 6
Health Check Type ELB
Health Check Grace Period 300
Termination Policies Default

Availability Zone(s) us-west-1b, us-west-1c
Subnet(s) subnet-5341c434,subnet-4a9b4e11
Default Cooldown 300
Placement Group
Suspended Processes
Enabled Metrics GroupMaxSize, GroupMinSize, GroupDesiredCapacity, GroupTerminatingInstances,



□	Name	Launch Configur	Instances	Desired	Min	Max	Availability Zones	Default Cooldown
□	E-commerce-Application-Manager…	E-commerce-App…	3	3	0	6	us-west-1b, us-west-1c	300
☑	E-commerce-Application-NodeAsg…	E-commerce-App…	3	3	0	1,000	us-west-1b, us-west-1c	300

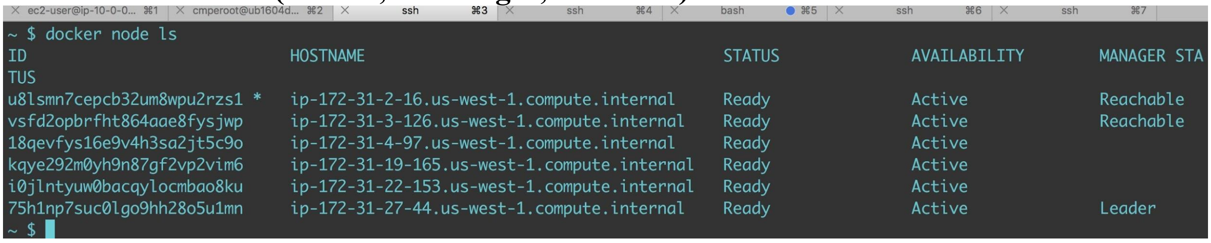
Auto Scaling Group: E-commerce-Application-NodeAsg-O8OZAMG9DDGU

Launch Configuration E-commerce-Application-NodeLaunchConfig17090ceaws1-ESZEKYYJL8CH
Launch Template
Launch Template Version
Load Balancers E-commerc-External-1CLBG7QGVLRTF
Target Groups
Desired 3
Min 0
Max 1000
Health Check Type ELB
Health Check Grace Period 300
Termination Policies Default

Availability Zone(s) us-west-1b, us-west-1c
Subnet(s) subnet-5341c434,subnet-4a9b4e11
Default Cooldown 300
Placement Group
Suspended Processes
Enabled Metrics GroupInServiceInstances,

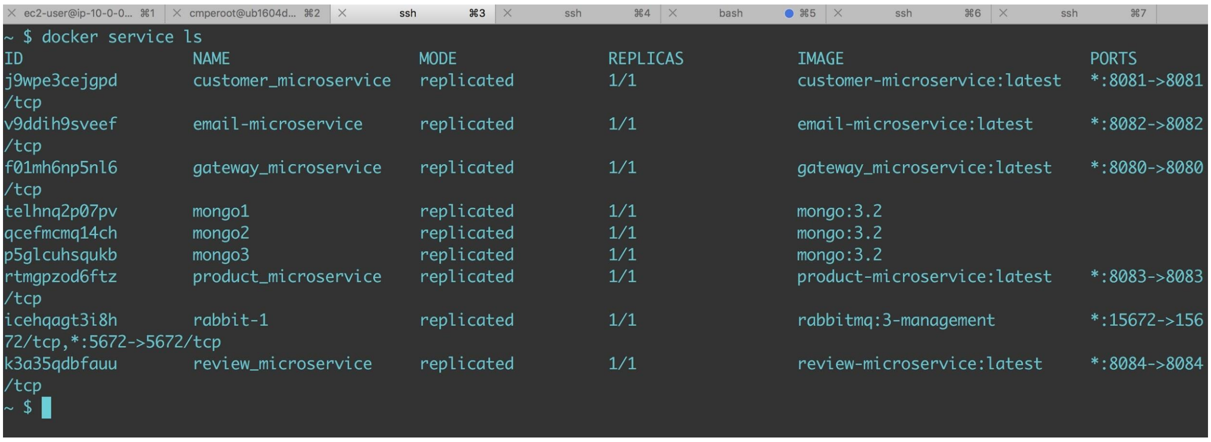
Screenshot related to Docker Swarm on AWS

All nodes in swarm (6 total, 3 manager, 3 worker)



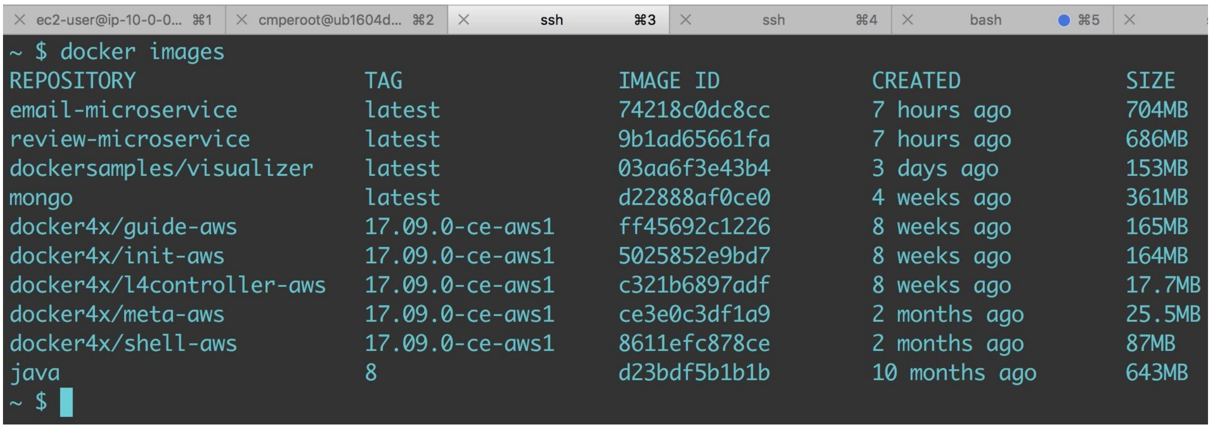
```
~ $ docker node ls
ID                            HOSTNAME                                  STATUS              AVAILABILITY        MANAGER STA
TUS
u8lsmn7cepcb32um8wpu2rzs1 *   ip-172-31-2-16.us-west-1.compute.internal    Ready               Active              Reachable
vsfd2opbrfht864aae8fysjwp     ip-172-31-3-126.us-west-1.compute.internal   Ready               Active              Reachable
18qevfys16e9v4h3sa2jt5c9o     ip-172-31-4-97.us-west-1.compute.internal    Ready               Active
kqye292m0yh9n87gf2vp2vim6     ip-172-31-19-165.us-west-1.compute.internal  Ready               Active
i0jlntyuw0bacqylocmbao8ku     ip-172-31-22-153.us-west-1.compute.internal  Ready               Active
75h1np7suc0lgo9hh28o5u1mn     ip-172-31-27-44.us-west-1.compute.internal   Ready               Active              Leader
~ $
```

All services running in swarm

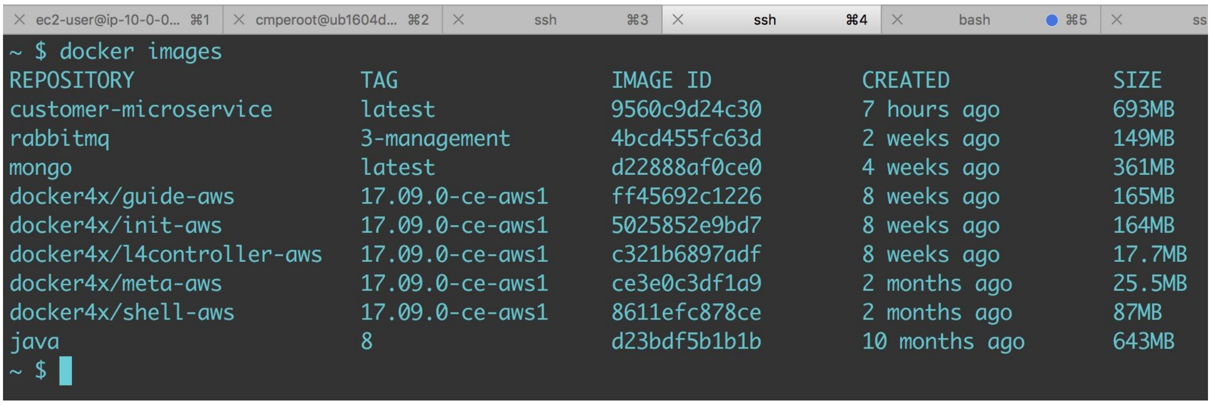


```
~ $ docker service ls
ID                  NAME                    MODE                REPLICAS            IMAGE                            PORTS
j9wpe3cejgpd        customer_microservice   replicated          1/1                 customer-microservice:latest     *:8081->8081
/tcp
v9ddih9sveef        email-microservice      replicated          1/1                 email-microservice:latest        *:8082->8082
/tcp
f01mh6np5nl6        gateway_microservice    replicated          1/1                 gateway_microservice:latest      *:8080->8080
/tcp
telhnq2p07pv        mongo1                  replicated          1/1                 mongo:3.2
qcefmcmq14ch        mongo2                  replicated          1/1                 mongo:3.2
p5glcuhsqukb        mongo3                  replicated          1/1                 mongo:3.2
rtmgpzod6ftz        product_microservice    replicated          1/1                 product-microservice:latest      *:8083->8083
/tcp
icehqagt3i8h        rabbit-1                replicated          1/1                 rabbitmq:3-management            *:15672->156
72/tcp,*:5672->5672/tcp
k3a35qdbfauu        review_microservice     replicated          1/1                 review-microservice:latest       *:8084->8084
/tcp
~ $
```

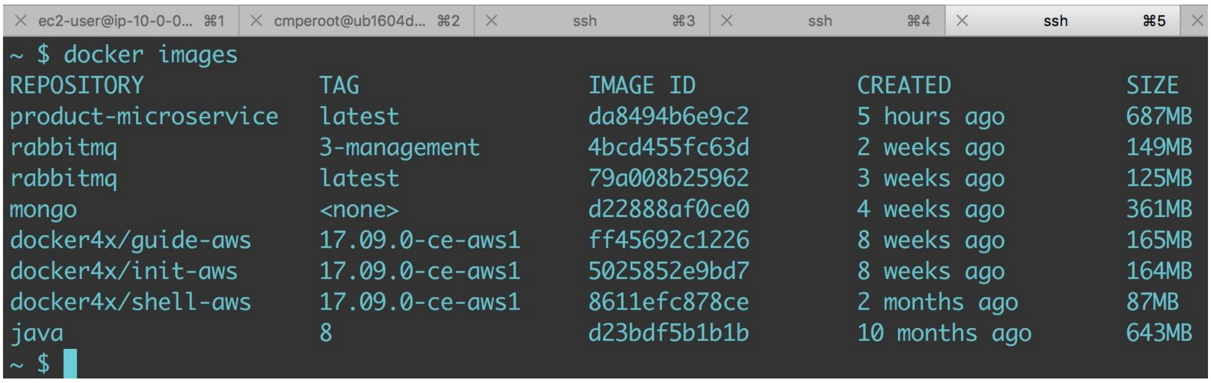
All docker images



```
~ $ docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
email-microservice          latest              74218c0dc8cc        7 hours ago         704MB
review-microservice         latest              9b1ad65661fa        7 hours ago         686MB
dockersamples/visualizer    latest              03aa6f3e43b4        3 days ago          153MB
mongo                       latest              d22888af0ce0        4 weeks ago         361MB
docker4x/guide-aws          17.09.0-ce-aws1     ff45692c1226        8 weeks ago         165MB
docker4x/init-aws           17.09.0-ce-aws1     5025852e9bd7        8 weeks ago         164MB
docker4x/l4controller-aws   17.09.0-ce-aws1     c321b6897adf        8 weeks ago         17.7MB
docker4x/meta-aws           17.09.0-ce-aws1     ce3e0c3df1a9        2 months ago        25.5MB
docker4x/shell-aws          17.09.0-ce-aws1     8611efc878ce        2 months ago        87MB
java                        8                   d23bdf5b1b1b        10 months ago       643MB
~ $
```

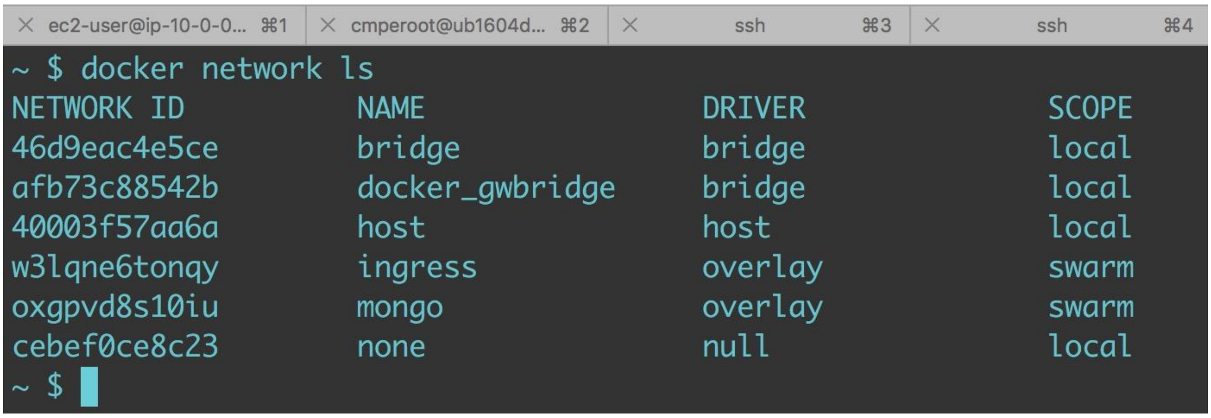


```
~ $ docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
customer-microservice       latest              9560c9d24c30        7 hours ago         693MB
rabbitmq                    3-management        4bcd455fc63d        2 weeks ago         149MB
mongo                       latest              d22888af0ce0        4 weeks ago         361MB
docker4x/guide-aws          17.09.0-ce-aws1     ff45692c1226        8 weeks ago         165MB
docker4x/init-aws           17.09.0-ce-aws1     5025852e9bd7        8 weeks ago         164MB
docker4x/l4controller-aws   17.09.0-ce-aws1     c321b6897adf        8 weeks ago         17.7MB
docker4x/meta-aws           17.09.0-ce-aws1     ce3e0c3df1a9        2 months ago        25.5MB
docker4x/shell-aws          17.09.0-ce-aws1     8611efc878ce        2 months ago        87MB
java                        8                   d23bdf5b1b1b        10 months ago       643MB
~ $
```



```
~ $ docker images
REPOSITORY             TAG                 IMAGE ID            CREATED             SIZE
product-microservice   latest              da8494b6e9c2        5 hours ago         687MB
rabbitmq               3-management        4bcd455fc63d        2 weeks ago         149MB
rabbitmq               latest              79a008b25962        3 weeks ago         125MB
mongo                  <none>              d22888af0ce0        4 weeks ago         361MB
docker4x/guide-aws     17.09.0-ce-aws1     ff45692c1226        8 weeks ago         165MB
docker4x/init-aws      17.09.0-ce-aws1     5025852e9bd7        8 weeks ago         164MB
docker4x/shell-aws     17.09.0-ce-aws1     8611efc878ce        2 months ago        87MB
java                   8                   d23bdf5b1b1b        10 months ago       643MB
~ $
```

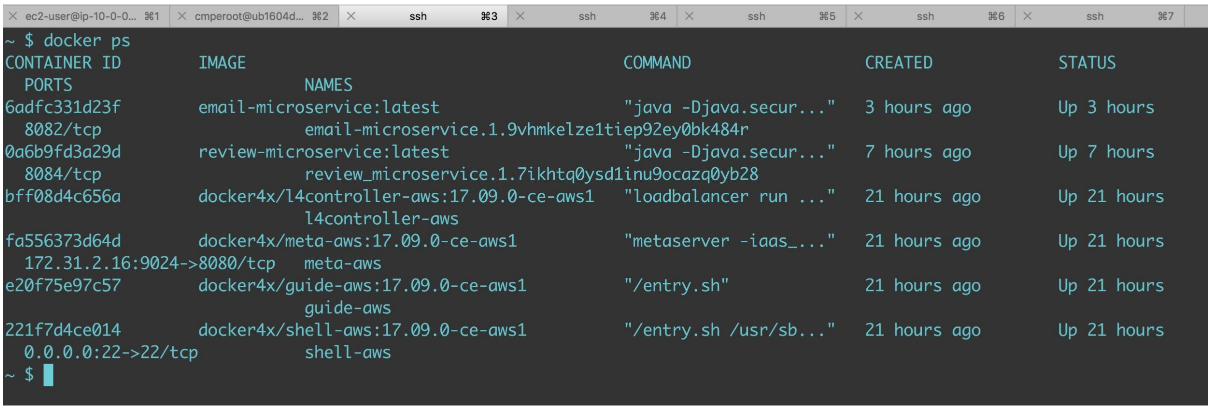
Docker networks



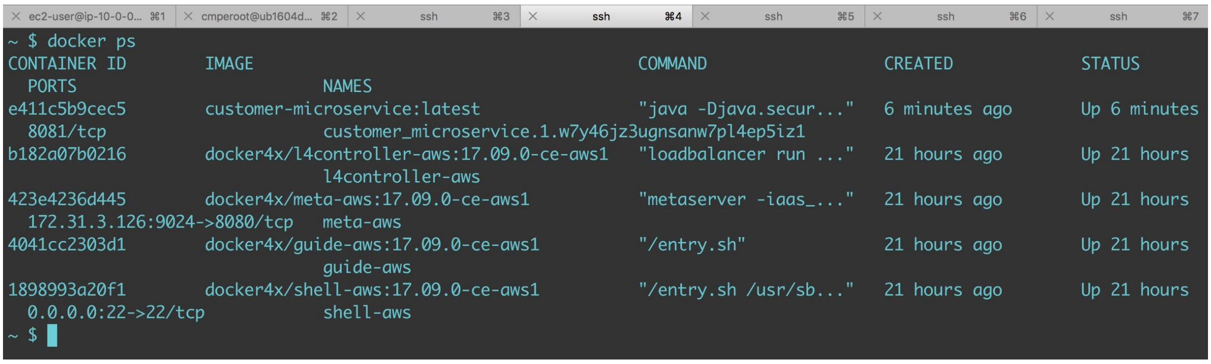
```
~ $ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
46d9eac4e5ce        bridge              bridge              local
afb73c88542b        docker_gwbridge     bridge              local
40003f57aa6a        host                host                local
w3lqne6tonqy        ingress             overlay             swarm
oxgpvd8s10iu        mongo               overlay             swarm
cebef0ce8c23        none                null                local
~ $
```

We use mongo network, which is setup as overlay network to work across the swarm with different containers on different instances.

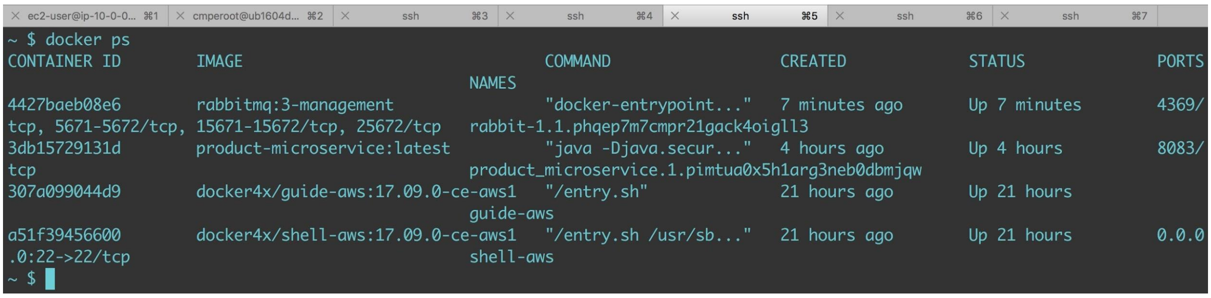
Docker ps on different instances.



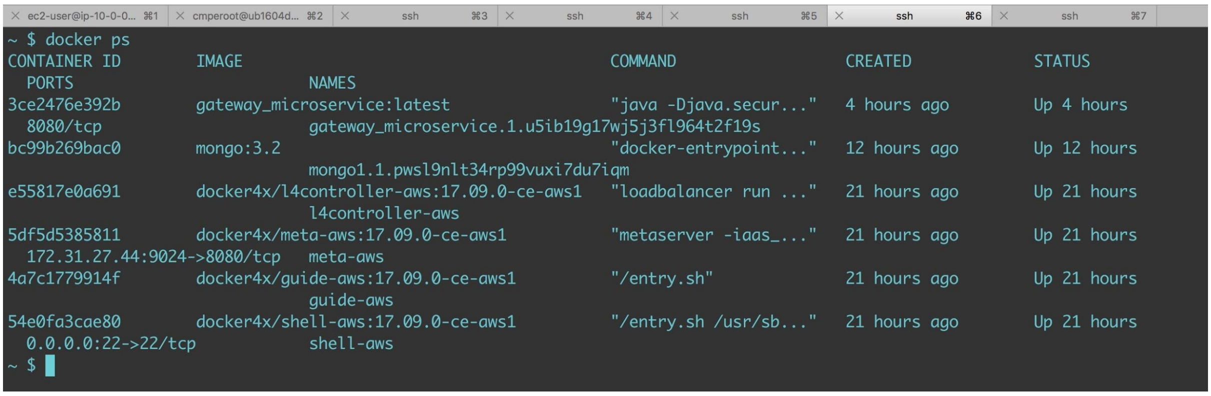
```
~ $ docker ps
CONTAINER ID        IMAGE                                         COMMAND                  CREATED             STATUS
  PORTS                        NAMES
6adfc331d23f        email-microservice:latest                     "java -Djava.secur..."   3 hours ago         Up 3 hours
  8082/tcp                     email-microservice.1.9vhmkelze1tiep92ey0bk484r
0a6b9fd3a29d        review-microservice:latest                    "java -Djava.secur..."   7 hours ago         Up 7 hours
  8084/tcp                     review_microservice.1.7ikhtq0ysd1inu9ocazq0yb28
bff08d4c656a        docker4x/l4controller-aws:17.09.0-ce-aws1     "loadbalancer run ..."   21 hours ago        Up 21 hours
                               l4controller-aws
fa556373d64d        docker4x/meta-aws:17.09.0-ce-aws1             "metaserver -iaas_..."   21 hours ago        Up 21 hours
  172.31.2.16:9024->8080/tcp   meta-aws
e20f75e97c57        docker4x/guide-aws:17.09.0-ce-aws1            "/entry.sh"              21 hours ago        Up 21 hours
                               guide-aws
221f7d4ce014        docker4x/shell-aws:17.09.0-ce-aws1            "/entry.sh /usr/sb..."   21 hours ago        Up 21 hours
  0.0.0.0:22->22/tcp           shell-aws
~ $
```



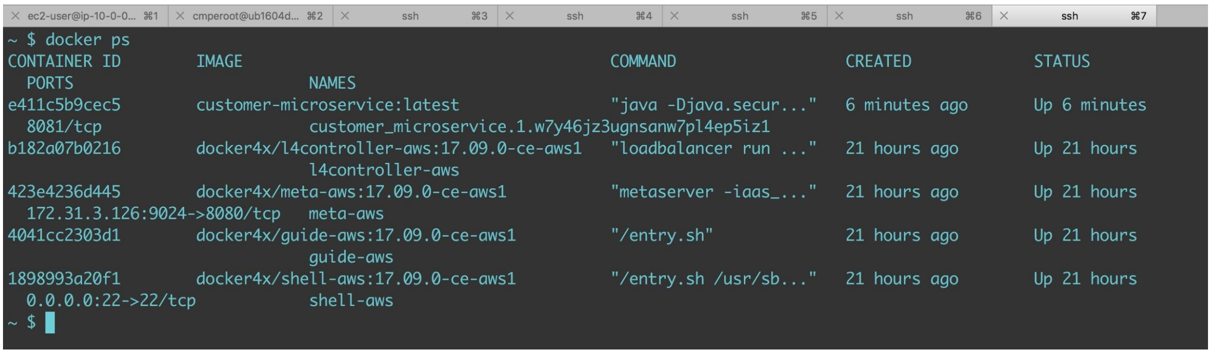
```
~ $ docker ps
CONTAINER ID        IMAGE                                       COMMAND                  CREATED             STATUS
  PORTS                         NAMES
e411c5b9cec5        customer-microservice:latest                "java -Djava.secur..."   6 minutes ago       Up 6 minutes
  8081/tcp                      customer_microservice.1.w7y46jz3ugnsanw7pl4ep5iz1
b182a07b0216        docker4x/l4controller-aws:17.09.0-ce-aws1   "loadbalancer run ..."   21 hours ago        Up 21 hours
                                l4controller-aws
423e4236d445        docker4x/meta-aws:17.09.0-ce-aws1           "metaserver -iaas_..."   21 hours ago        Up 21 hours
  172.31.3.126:9024->8080/tcp   meta-aws
4041cc2303d1        docker4x/guide-aws:17.09.0-ce-aws1          "/entry.sh"              21 hours ago        Up 21 hours
                                guide-aws
1898993a20f1        docker4x/shell-aws:17.09.0-ce-aws1          "/entry.sh /usr/sb..."   21 hours ago        Up 21 hours
  0.0.0.0:22->22/tcp            shell-aws
~ $ 
```



```
~ $ docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED             STATUS              PORTS
                                                 NAMES
4427baeb08e6        rabbitmq:3-management                "docker-entrypoint..."   7 minutes ago       Up 7 minutes        4369/
tcp, 5671-5672/tcp, 15671-15672/tcp, 25672/tcp   rabbit-1.1.phqep7m7cmpr21gack4oigll3
3db15729131d        product-microservice:latest          "java -Djava.secur..."   4 hours ago         Up 4 hours          8083/
tcp                                              product_microservice.1.pimtua0x5h1arg3neb0dbmjqw
307a099044d9        docker4x/guide-aws:17.09.0-ce-aws1   "/entry.sh"              21 hours ago        Up 21 hours
                                                 guide-aws
a51f39456600        docker4x/shell-aws:17.09.0-ce-aws1   "/entry.sh /usr/sb..."   21 hours ago        Up 21 hours         0.0.0
.0:22->22/tcp                                    shell-aws
~ $ 
```

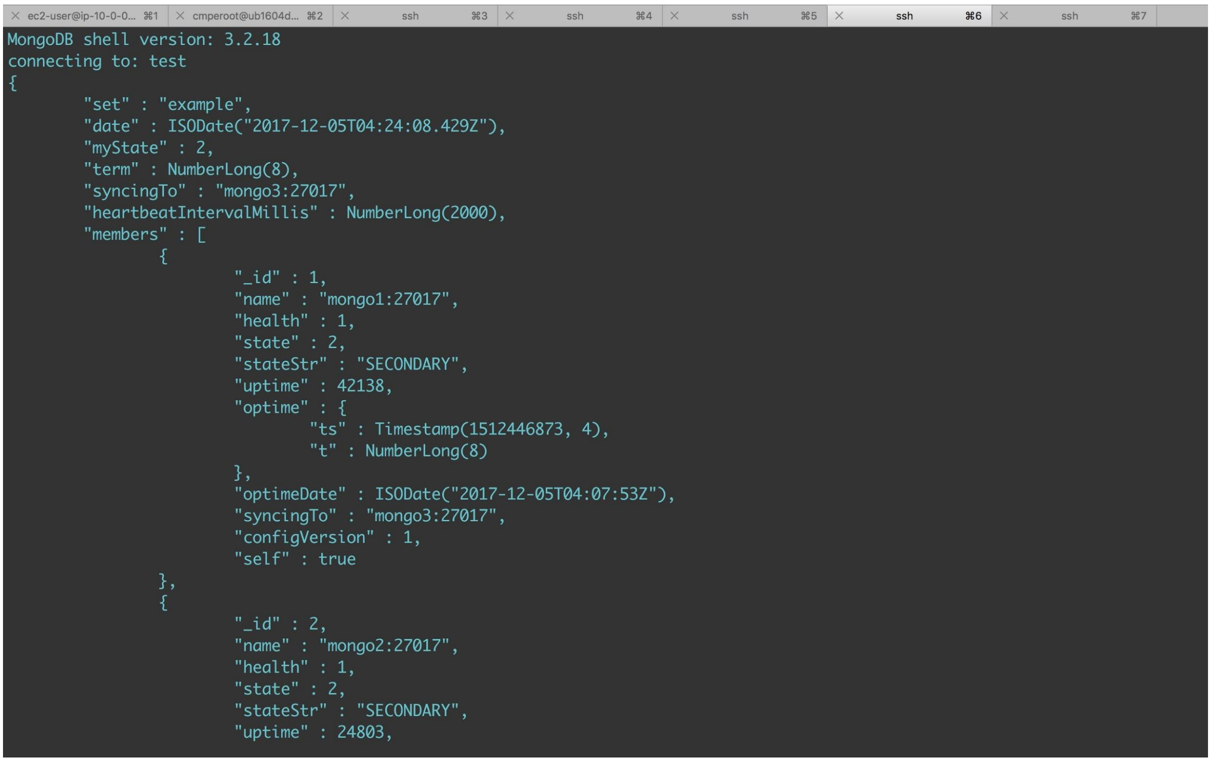


```
~ $ docker ps
CONTAINER ID        IMAGE                                       COMMAND                  CREATED             STATUS
  PORTS                         NAMES
3ce2476e392b        gateway_microservice:latest                 "java -Djava.secur..."   4 hours ago         Up 4 hours
  8080/tcp                      gateway_microservice.1.u5ib19g17wj5j3fl964t2f19s
bc99b269bac0        mongo:3.2                                   "docker-entrypoint..."   12 hours ago        Up 12 hours
                                mongo1.1.pwsl9nlt34rp99vuxi7du7iqm
e55817e0a691        docker4x/l4controller-aws:17.09.0-ce-aws1   "loadbalancer run ..."   21 hours ago        Up 21 hours
                                l4controller-aws
5df5d5385811        docker4x/meta-aws:17.09.0-ce-aws1           "metaserver -iaas_..."   21 hours ago        Up 21 hours
  172.31.27.44:9024->8080/tcp   meta-aws
4a7c1779914f        docker4x/guide-aws:17.09.0-ce-aws1          "/entry.sh"              21 hours ago        Up 21 hours
                                guide-aws
54e0fa3cae80        docker4x/shell-aws:17.09.0-ce-aws1          "/entry.sh /usr/sb..."   21 hours ago        Up 21 hours
  0.0.0.0:22->22/tcp            shell-aws
~ $ 
```

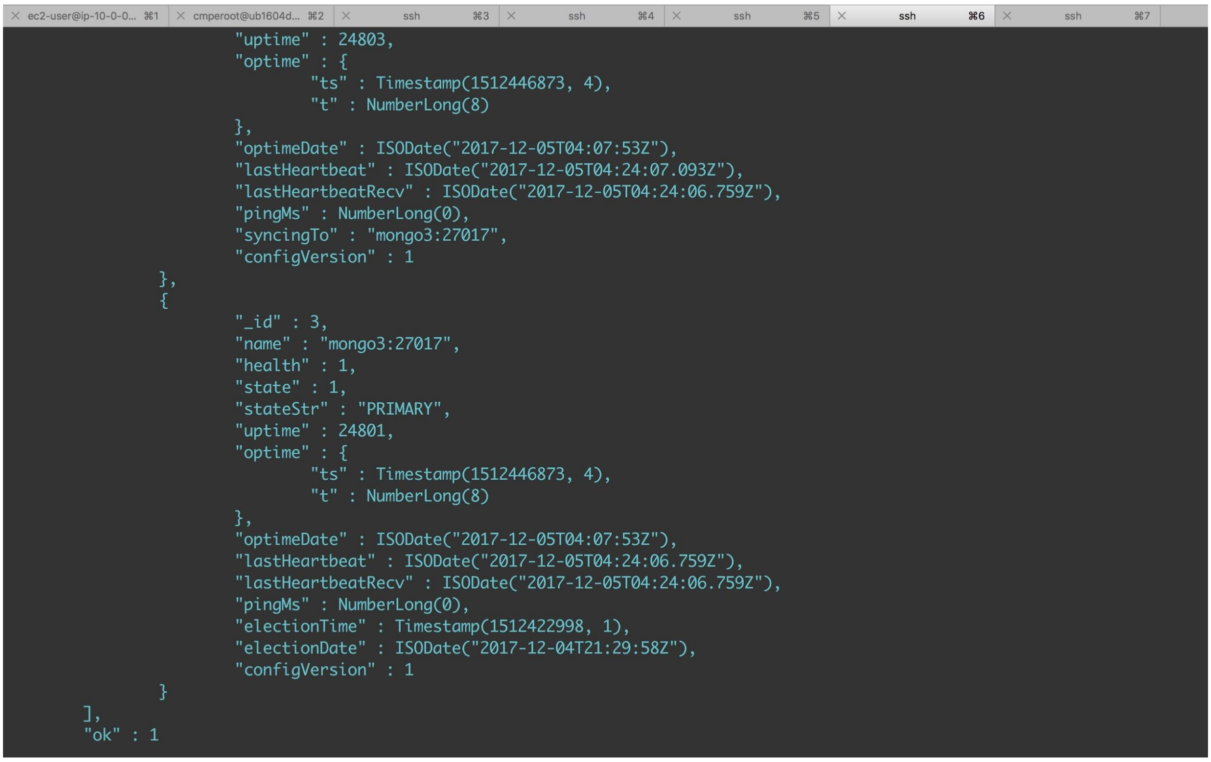


```
~ $ docker ps
CONTAINER ID        IMAGE                                       COMMAND                  CREATED             STATUS
  PORTS                         NAMES
e411c5b9cec5        customer-microservice:latest                "java -Djava.secur..."   6 minutes ago       Up 6 minutes
  8081/tcp                      customer_microservice.1.w7y46jz3ugnsanw7pl4ep5iz1
b182a07b0216        docker4x/l4controller-aws:17.09.0-ce-aws1   "loadbalancer run ..."   21 hours ago        Up 21 hours
                                l4controller-aws
423e4236d445        docker4x/meta-aws:17.09.0-ce-aws1           "metaserver -iaas_..."   21 hours ago        Up 21 hours
  172.31.3.126:9024->8080/tcp   meta-aws
4041cc2303d1        docker4x/guide-aws:17.09.0-ce-aws1          "/entry.sh"              21 hours ago        Up 21 hours
                                guide-aws
1898993a20f1        docker4x/shell-aws:17.09.0-ce-aws1          "/entry.sh /usr/sb..."   21 hours ago        Up 21 hours
  0.0.0.0:22->22/tcp            shell-aws
~ $ 
```

MongoDB replica setup



```
MongoDB shell version: 3.2.18
connecting to: test
{
        "set" : "example",
        "date" : ISODate("2017-12-05T04:24:08.429Z"),
        "myState" : 2,
        "term" : NumberLong(8),
        "syncingTo" : "mongo3:27017",
        "heartbeatIntervalMillis" : NumberLong(2000),
        "members" : [
                {
                        "_id" : 1,
                        "name" : "mongo1:27017",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 42138,
                        "optime" : {
                                "ts" : Timestamp(1512446873, 4),
                                "t" : NumberLong(8)
                        },
                        "optimeDate" : ISODate("2017-12-05T04:07:53Z"),
                        "syncingTo" : "mongo3:27017",
                        "configVersion" : 1,
                        "self" : true
                },
                {
                        "_id" : 2,
                        "name" : "mongo2:27017",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 24803,
```



```
                        "uptime" : 24803,
                        "optime" : {
                                "ts" : Timestamp(1512446873, 4),
                                "t" : NumberLong(8)
                        },
                        "optimeDate" : ISODate("2017-12-05T04:07:53Z"),
                        "lastHeartbeat" : ISODate("2017-12-05T04:24:07.093Z"),
                        "lastHeartbeatRecv" : ISODate("2017-12-05T04:24:06.759Z"),
                        "pingMs" : NumberLong(0),
                        "syncingTo" : "mongo3:27017",
                        "configVersion" : 1
                },
                {
                        "_id" : 3,
                        "name" : "mongo3:27017",
                        "health" : 1,
                        "state" : 1,
                        "stateStr" : "PRIMARY",
                        "uptime" : 24801,
                        "optime" : {
                                "ts" : Timestamp(1512446873, 4),
                                "t" : NumberLong(8)
                        },
                        "optimeDate" : ISODate("2017-12-05T04:07:53Z"),
                        "lastHeartbeat" : ISODate("2017-12-05T04:24:06.759Z"),
                        "lastHeartbeatRecv" : ISODate("2017-12-05T04:24:06.759Z"),
                        "pingMs" : NumberLong(0),
                        "electionTime" : Timestamp(1512422998, 1),
                        "electionDate" : ISODate("2017-12-04T21:29:58Z"),
                        "configVersion" : 1
                }
        ],
        "ok" : 1
```

Test Plan Execution

No	Gateway API Test	Description	Status
1	List all product	List of product information	PASS
2	Search product	Search product by keyword	PASS
3	List product information	List product detailed information	PASS
4	Add Review for product	Add rating for product	PASS
5	Get product average review	Show the average rating for product	PASS
6	Register new customer	Register a new customer	PASS
7	Register email notification	Send register email to customer	FAIL

Any major modification from proposal and why

To our project, we tried to focus on e-commerce micro-service architecture, and cloud service leverage. We simplify our original functionalities with the following major change. First, we remove the seller role, only keep customer user role. Second, we merge the inventory to product, with extra stock field in product scheme. The revised functionalities as follows:

- Login service for user – implemented
- Email service for new registerd user - implemented
- Search products for user with keyword - implemented
- Customer review for product – implemented
- Checkout and Past Shopping History data for users - not implemented

Also, initially we planned to create a separate database for separate micro service but we then decided to keep one MongoDB setup to talk with all micro services. We modified this implementation because we had all service using similar fields like product ID, user, etc and hence we wanted to keep database consistent. We implemented the database with 3 replicas. One container which is primary and 2 container in secondary replica mode.

For message queue service, the original design is to AWS SQS service, however, there some practical issue while developing the service, it cannot test the SQS service without Internet connection. Although SQS message service is very cheap, we can have other free option like host a message queue instance with RabbitMQ and can easily test and development on localhost.

Any uniqueness (design, implementation, etc) that you proud of

Using Cloudformation template of Docker Swarm for deployment of micro services. Because of the template, I was able to get EC2 instances which are specially optimized and built for Docker. Also, template automatically put ELB in front of the Swarm. Hence all the request to

ELB DNS will be routed to different containers on round by round basis. Also, using MongoDB in 3 replica mode has been a great design idea. Using Docker Swarm instead of ECS has been really great and simple. All micro-service externalize the configuration such as MongoDB connection, RabbitMQ connection, gateway route to end point, these prevent the hard-coding, can more flexible config with external config file while deployed the service.

Final major areas/components/task of each member, when each was started/completed

Task	Member	Start	End
Customer service	Tsung-Min, Arpeet	11/20	11/20
Product service	Tsung-Min	11/21	11/21
Review service	Tsung-Min	11/22	11/22
Email service	Tsung-Min	11/23	11/27
Gateway service	Tsung-Min	11/27	11/29
Docker image build	Arpeet	11/22	11/29
AWS Cloudformation template	Arpeet	11/24	11/29
AWS Docker Swarm deploy	Arpeet	11/25	11/29
Test execution	Tsung-Min, Arpeet	11/29	11/30
Project report	Arpeet, Tsung-Min	12/02	12/04
Project slide	Arpeet	12/02	12/03

Future-work and improvements:

- Continue Integration test with CI service on cloud
- Auto deployment integration with cloud service
- Using LocalStack that does localhost emulation of AWS cloud stack