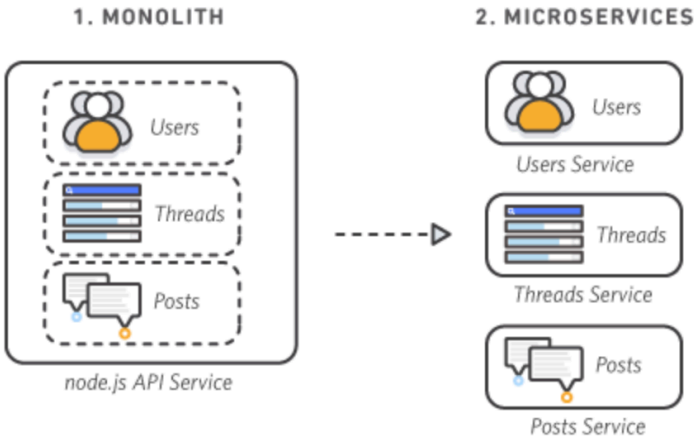
AWS Project : Break a Monolith Application into Microservices

Time spent: 200min

1. Overview

In this tutorial, you will deploy a monolithic node.js application to a Docker container, then decouple the application into microservices without any downtime. The node.js application hosts a simple message board with threads and messages between users.



1 **Monolithic Architecture**

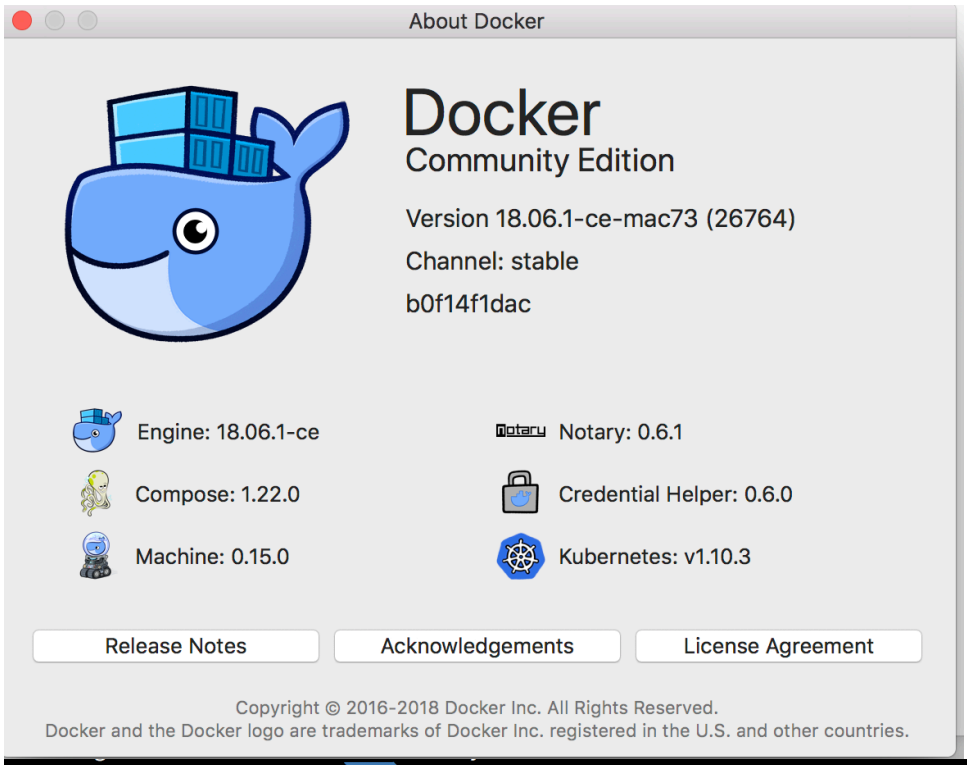
The entire node.js application is run in a container as a single service and each container has the same features as all other containers. If one application feature experiences a spike in demand, the entire architecture must be scaled.

2 **Microservices Architecture**

Each feature of the node.js application runs as a separate service within its own container. The services can scale and be updated independently of the others.

2. Containerize the Monolith

1) Environment set up
   AWS CLI, Docker




```
 expo@lishibodeMacBook-Pro  ~  pip install awscli --upgrade --user
Collecting awscli
  Downloading https://files.pythonhosted.org/packages/8d/d6/12edb1740e19c174bf71b7e58da13beca505db
c679612ae44bc87f9464f9/awscli-1.16.46-py2.py3-none-any.whl (1.4MB)
    100% |████████████████████████████████| 1.4MB 1.1MB/s
Collecting docutils>=0.10 (from awscli)
  Downloading https://files.pythonhosted.org/packages/50/09/c53398e0005b11f7ffb27b7aa720c617aba53b
e4fb4f4f3f06b9b5c60f28/docutils-0.14-py2-none-any.whl (543kB)
    100% |████████████████████████████████| 552kB 2.6MB/s
Collecting botocore==1.12.36 (from awscli)
  Downloading https://files.pythonhosted.org/packages/8a/35/3dfbd6fac89abd8cd3f8090a892fd941e97ec5
34e781b39c5bb804940815/botocore-1.12.36-py2.py3-none-any.whl (4.7MB)
    100% |████████████████████████████████| 4.7MB 305kB/s
Collecting rsa<=3.5.0,>=3.1.2 (from awscli)
```

2) Get project

```
 expo@lishibodeMacBook-Pro  ~/Desktop/Class/Third/6421/AWS_dockerPJ  git clone https://github.co
m/awslabs/amazon-ecs-nodejs-microservices.git
Cloning into 'amazon-ecs-nodejs-microservices'...
remote: Enumerating objects: 79, done.
remote: Total 79 (delta 0), reused 0 (delta 0), pack-reused 79
```

3) Get repository

Build, tag, and push Docker image

Now that your repository exists, you can push a Docker image by following these steps:

Successfully created repository
238665400611.dkr.ecr.us-east-2.amazonaws.com/shibo

4) Get docker login

```
expo@lishibodeMacBook-Pro ~/Desktop/Class/third/6421/AWS_dockerPJ/amazon-ecs-nodejs-microservices/2-containerized/services/api master docker login -u AWS -
p eyJwYXlsb2FkIjoiK0t3WkMvWkM3WW1URzBSaTQrbUhWNHZRVitjK0NmVTdUYVVmMmRjVFp3NDY2QVcyU1JJZEFIR2U2VjV5N0s1UHptMUpDelB5My9GbStNTWl4VlNlT3hxSWlrOWJuZDNtZG5pM1prS3F3SUxxWD
BHakNQam4yVGdZN2k2bUNNM0VaNU1pNFZSUHg2QnA2alhXTUNrQkFOVTR2MDVJd3pVUllNSmJLY0UyTGZXL0xRZ2ZNY1lsMks1NVR5Y0xvN3pXTWFwYm95SDdlQWtvdUFGeDJNVWFaa2ZxMEN5ZlVBekVwdEIvcExxcm
xGd01mNVd0aERrK3pGZW5PNW9JNUlOZXpPbjNMaTN6WFVIRGpCZ2p3UFU1bVF2NC9tcThaWE9OV2grdWQxRjcvc1FaaXM0S0Z3dGRubGpIU3NjSk1kTCtGdS9lNjJJejVFMzBLcnkzLzZnRTROOFZUL09jcmU1SUNmbj
FncWxKUEg5OVB4dlhzOCtxc0hYelFzakVoeFgwQTRsRFhVbG84YytiM0diVVlxblgwaHEzU0NVWHpHaUFmd1ZNNURnUVVLUXhEVUY5N0Zielh5ZHMzT3NtQ3Ztb2RtRTg3czBmcUdEQmlROFk0a2p0MERSS3J6elRseF
ZQa2pFTVIrQVVNZFF6V0JYb1dDVklFK0dQRjlWRnZhbkFxQ3FIYnFYZEhRakt3cDQ3ZHZoajU4ZjBOTnFwaGE1OFV1cmhramI1cDR4RUdIemxtZlA1aWZtNndaaFNBUkNZTUJHZEYzTTZPeFd2S3hPaUlXM0NadVlaUE
UxczFMS2RMUjZvbmNZWkg1U25kZEJkenVreEFkQmUwdXVGMWFQQ04yTnNrd1JmZUVaS3lBdi84cCtRWERwTjRXNVFkRXZJd0FYY0pEelBsdGs4V3QwUUdLUmVwNG5JVUEyZDBRYmRibzRuWnB0WGphOFJuYU8vQkpPOX
h5UEVLa3ZFV1Z0d1pqZUcreitUdVNVVUx4SUVORml1Y2RTc204SmV2U2x5YkFreGozandubUlPTkJuUk5ZbWg5bGxnUEdwNUFTaWJ5Y3RHNVZwS2hrdnd0dEVmLzU4blk2YWdycnppbjhkaG5iU1ZFY1N0ZDV1bGZZby
tHdUNLYllmWVd5SUFFKyttcFJHY0c0WC9HdVdSbnRwZmxvQlFyWlc1aklSeFArZDdjTkoyUUFiTkw3Q0xjMWNiR1FDQkR0QlBiYnduS1lDOUpWekxsV0xNeGJvdTlIYUlNUXM3VEN5MGVsQUx3aGdLZmVtYm9OeENXdF
gybzh2WVdaT29GTTNNOC9lS3dCSnU4TExuYk1oV0JMckhRMmdVZ1oxYm5BczclDSkVnQURqZTBUc0cyaUVtT3l5R1QzWC8zelNac2cxT2ZDOHJGTmc1WnZYbTNPWkVBNTNidGhUOFp0VTEzdVhXQnpkOEVIVmREVS91Sm
NCK3JsNzZ0bHdaNUVVNkZFeitndDdNTnRvRzQzbU9RbEM0NVdkUnpRPT0iLCJkYXRha2V5IjoiQVFFQkFIakI3L2lnd01nNE5Qd2F1cnhTSVl4NEhmbnh1R2MvNDhiRHd2d0RwTllXWmdBQUFINHdmQVlKS29aSWh2Y0
5BUWNHb0c4d2JRSUJBREJvQmdrcWhraUc5dzBCQndFd0hnWUpZSVpJQVdVREJBRXVNQkVFREFGL0ZVWXUzZXpxbWRQcmxnSUJFSUE3eGFzYzFmV2gyNTJXb2NsSzh4eTBHeldWUVR5blpOMmh4U1ROVkVkMmM3azFocV
lJeDlKWkI1L0xjRHlRbFEzS0JDOC8vblgrdElIRk4vMD0iLCJ2ZXJzaW9uIjoiMiIsInR5cGUiOiJEQVRBX0tFWSIsImV4cGlyYXRpb24iOjE1NDExODg4ODd9 https://238665400611.dkr.ecr.us-east-2.am
azonaws.com
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded
expo@lishibodeMacBook-Pro ~/Desktop/Class/third/6421/AWS_dockerPJ/amazon-ecs-nodejs-microservices/2-containerized/services/api master
```

5) Push the Image

```
expo@lishibodeMacBook-Pro ~/Desktop/Class/third/6421/AWS_dockerPJ/amazon-ecs-nodejs-microservices/2-containerized/services/api master docker tag 19263a9dde92 238665400611.dkr.ecr.us-east-2.a
mazonaws.com/shibo:latest
expo@lishibodeMacBook-Pro ~/Desktop/Class/third/6421/AWS_dockerPJ/amazon-ecs-nodejs-microservices/2-containerized/services/api master docker push 238665400611.dkr.ecr.us-east-2.amazonaws.com/s
hibo:latest
The push refers to repository [238665400611.dkr.ecr.us-east-2.amazonaws.com/shibo]
fce9d17b7576: Pushed
bf1327a05d73: Pushed
8e893534526a: Pushed
940fd7841192: Pushed
latest: digest: sha256:5feb84d659ced1b94ffcccaa7886a16766a79213543ed9d92587a68f4613d7c0 size: 1158
```

< All repositories : shibo

Repository ARN	arn:aws:ecr:us-east-2:238665400611:repository/shibo
Repository URI	238665400611.dkr.ecr.us-east-2.amazonaws.com/shibo

View Push Commands

Images | Permissions | Dry run of lifecycle rules | Lifecycle policy

Amazon ECR limits the number of images to 1,000 per repository. Request a limit increase.

Image sizes may appear compressed. Learn more

Delete

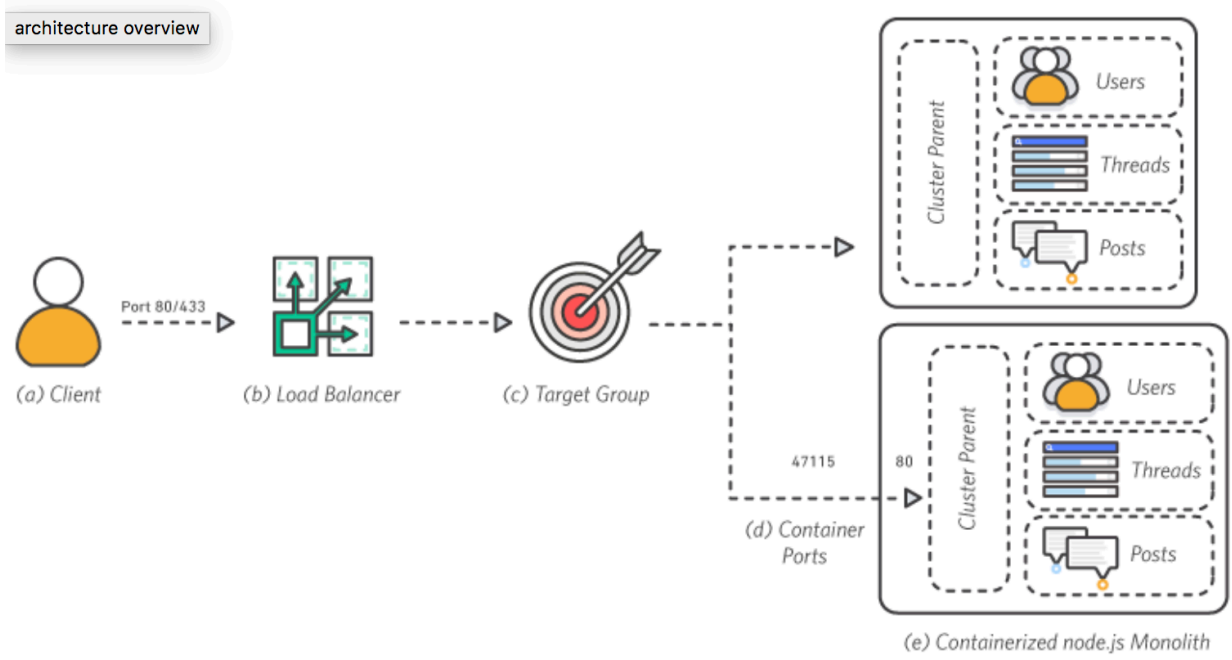
Last updated on November 2, 201

Filter in this page

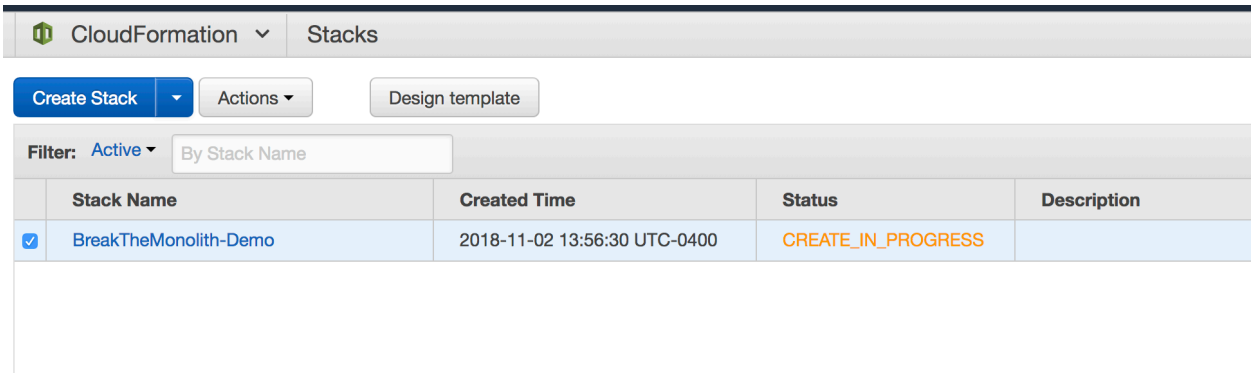
Image tags		Digest	Size (MiB)
latest	view all	sha256:5feb84d659ced1b94ffcccaa7886a16766a79213543ed9d92587a68f...	19.18

3. Deploy the Monolith

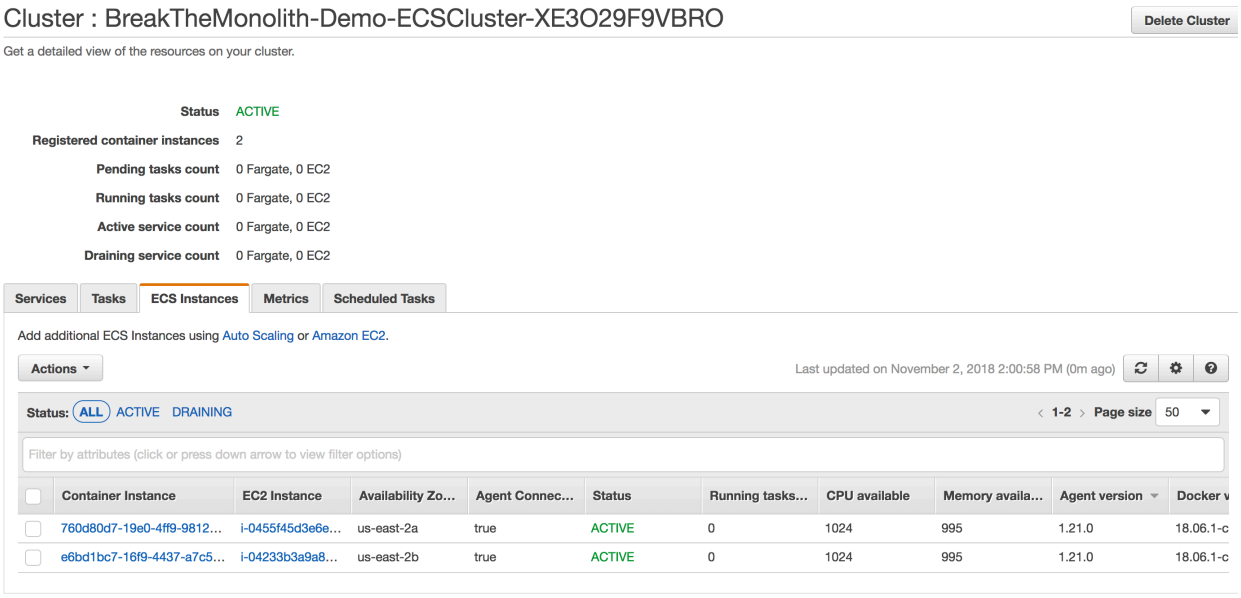
1) Overview



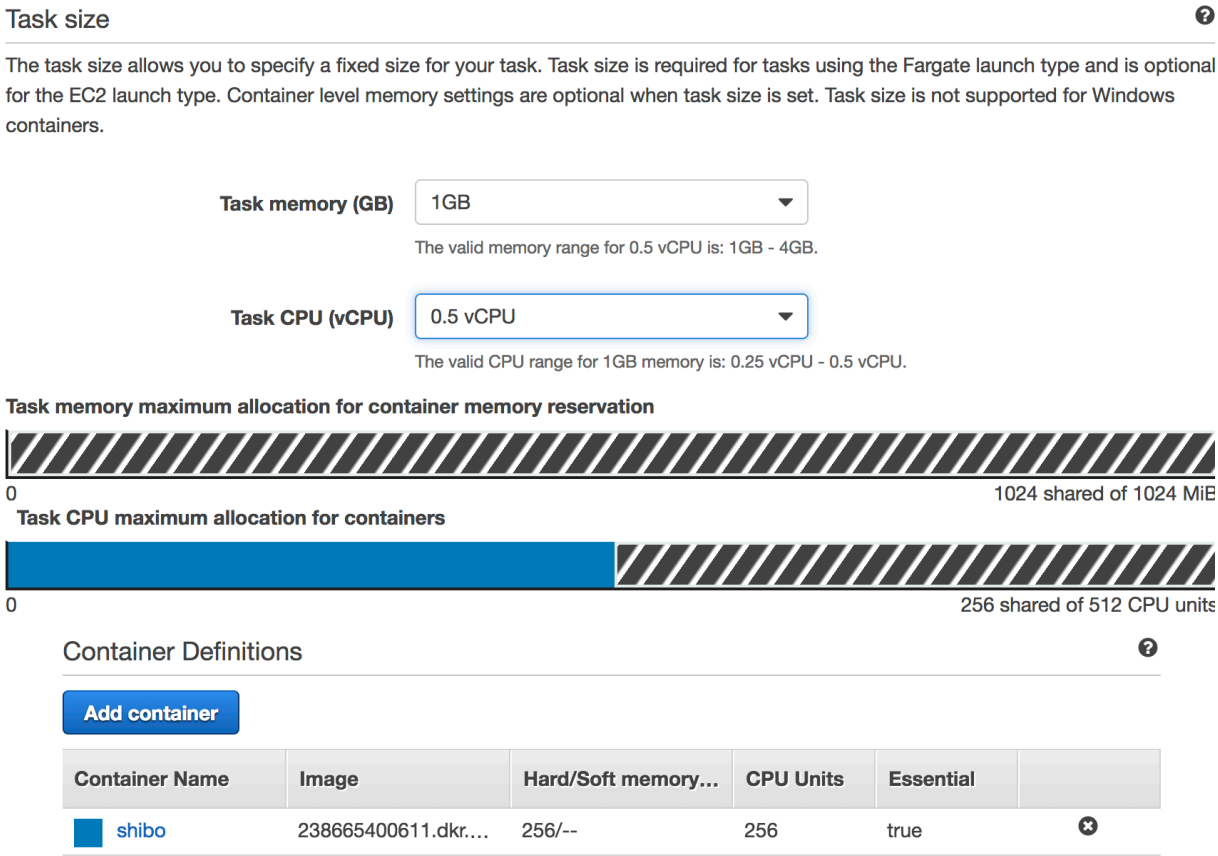
2) Create the Stack



3) Check cluster is running

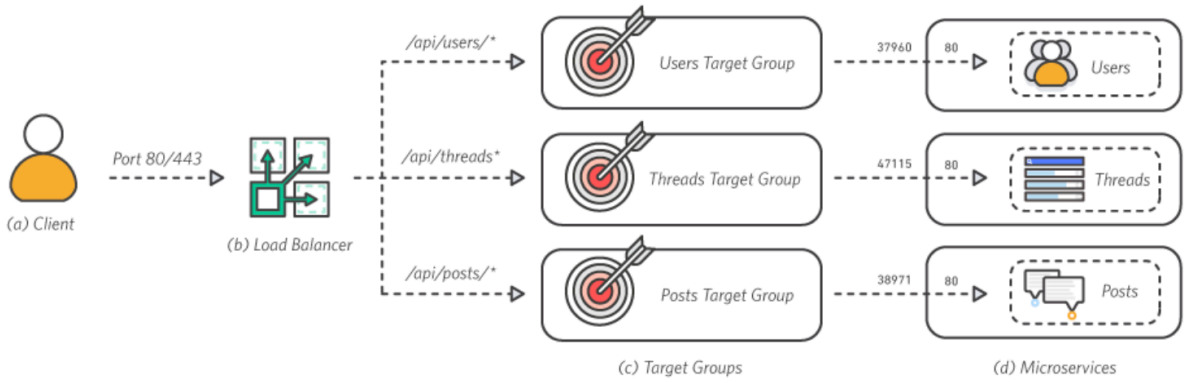


4) Define the task



4. Break the Monolith

1) Overview



2) Create 4 repositories

Filter by repository name

Repository name	Repository URI
posts	238665400611.dkr.ecr.us-east-2.amazonaws.com/posts
shibo	238665400611.dkr.ecr.us-east-2.amazonaws.com/shibo
thread	238665400611.dkr.ecr.us-east-2.amazonaws.com/thread
user	238665400611.dkr.ecr.us-east-2.amazonaws.com/user

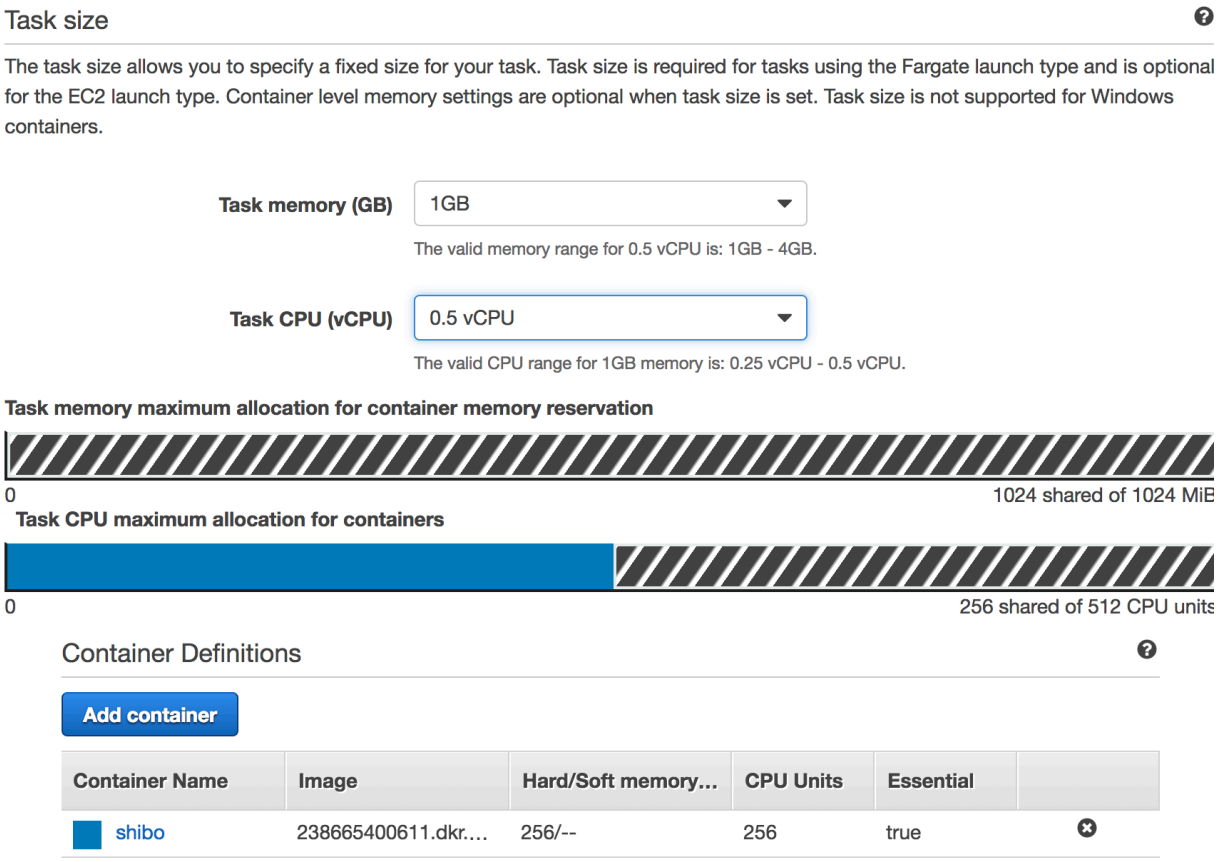
3) Tag and push each image

5. Deploy Microservices

1) Architecture overview

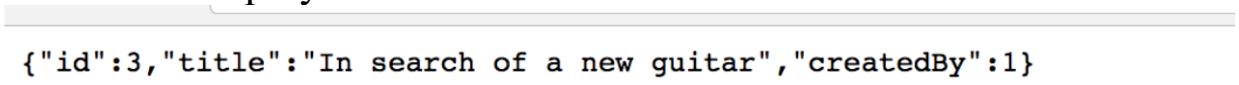
In this module, you will deploy your node.js application as a set of interconnected services behind an Application Load Balancer (ALB). Then, you will use the ALB to seamlessly shift traffic from the monolith to the microservices.

2) Define the task



3) Start Microservices

4) Validate the deployment



Summary

I have harvested a lot of precious experience about docker and containers in this project on AWS. Following the instruction, not only do I got the deeper understanding of dockers and containers, but also I connected the knowledge with the real usage.