

Dev Ops Engineer Task

Problem statement:

There are 5 node.js based microservices. Make a deployment strategy for the same with auto-scaling and auto deployment. (Create AWS account for testing or use an existing one if you have one). Use service instance per container pattern.

- Trigger deployment whenever someone pushes to the production branch on gitlab.
- The deployment process should update the code on all existing production servers in batches of 30% of the running servers at a time.

Solution:

Cloud Services use: AWS codepipeline, AWS beanstalk,S3,EC2,ASG,ELB

Other tool: Jenkins codepipeline, jq, shell scripting,git hub,nginx,Docker

Strategy: To achieve that changes I will create a CI/CD strategy by using **Jenkins** and **AWS** services.

I will create per microservices as single container so will created a pipeline in such a way that I can build application and after building that I will create a docker image with latest version of application and will push them into docker repo for that we must use api url of docker if you use docker hub url it will not work.

Then after pushing image into the docker hub account I need to create a Dockerrun.aws.json file from a template. So, for that will use jq or sed and update the image name with latest build number tag.

After building the template I will push those changes into s3 bucket. S3 bucket contain the **.dockercfg.json** file which contain my Docker hub authentication file and

Dockerrun.aws.json file which contain my image details S3 bucket should have version enabled to use with awscodepipeline and bucket should have in the same region.

Then I will create a awscodepipe line for which I will use my s3 bucket as a source and deploy my latest version of Dockerrun.aws.json file to elastic beanstalk (**Dockerrun.aws.json file is same like a docker-compose.yaml**)

As I am not aware of implementation of BLUE GREEN Deployment so that is actual reason I choose elastic bean stalk so I will create an Environment and Application and select the other configuration option. I will install my nginx on my ec2 instance and create a AMI of that and then add it under my ELB.

Nginx will work as application load balancer which will route my request to my different docker containers. We can do one more thing we can add nginx container in our Dockerrun.aws.json file and copy the nginx.conf configuration .ebextension folder and use the volume feature of docker and also need to links all other containers to nginx container

Steps for implementation:

Steps in Git:

- Create a nodejs microservice which contain 5 microservice under one master branch.
- Created a hook for Jenkins job invoke.

The screenshot shows a GitHub repository interface. At the top, there are tabs for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Insights, and Settings. Below the tabs, it says "No description, website, or topics provided." with an "Edit" button. Underneath, it says "Add topics". A progress bar shows 64 commits, 1 branch, 0 releases, and 1 contributor. Below the progress bar, there are buttons for "Branch: master", "New pull request", "Create new file", "Upload files", "Find file", and "Clone or download". The main content area shows a commit by "snjy991" titled "Update server.js" with the latest commit hash "2806fdb" from 3 hours ago. Below the commit header is a table of files changed in this commit:

File	Change	Time
nodeservice1	changed	a day ago
nodeservice2	Update server.js	3 hours ago
nodeservice3	Update server.js	3 hours ago
nodeservice4	Update server.js	3 hours ago
nodeservice5	Update server.js	4 hours ago
Dockerrun.aws.template.json	Update Dockerrun.aws.template.json	4 days ago
Jenkinsfile	Update Jenkinsfile	a day ago
buildjson.sh	file permisn	4 days ago

Steps in Jenkins:

- Created a new pipeline job in Jenkins.
- Created a Jenkinsfile and pushed it in git repo

Jenkinsfile contain following steps:

1. Clone repository: Checkout the repo from git
2. Build app : building the app for every service npm install
3. Build image : building docker image from Dockerfile exists in each microservices
4. Test Image : Testing all build container
5. Push Image : Pushing docker images with buildnumber tag in hub.docker.com
6. Changefile : changing the Dockerrun.aws.json v2 for elastic beanstalk deployment by using shell script in that I have used jq to replace the json content every time
7. Copying Dockerrun.aws.json file in S3 Bucket

```
node {
    def app1, app2, app3, app4, app5

    stage('Clone repository') {
        /* Let's make sure we have the repository cloned to our
        workspace */

        checkout scm
    }
}
```

```

    }

    stage('Build app') {
        dir ('nodeservice1') {
            sh 'npm install'
        }
        dir ('nodeservice2') {
            sh 'npm install'
        }
        dir ('nodeservice3') {
            sh 'npm install'
        }
        dir ('nodeservice4') {
            sh 'npm install'
        }
        dir ('nodeservice5') {
            sh 'npm install'
        }
    }

    stage('Build image') {
        /* This builds the actual image; synonymous to
        * docker build on the command line */
        dir ('nodeservice1') {
            app1 = docker.build("snjy991/nodeservice1")
        }
        dir ('nodeservice2') {
            app2 = docker.build("snjy991/nodeservice2")
        }
        dir ('nodeservice3') {
            app3 = docker.build("snjy991/nodeservice3")
        }
        dir ('nodeservice4') {
            app4 = docker.build("snjy991/nodeservice4")
        }
        dir ('nodeservice5') {
            app5 = docker.build("snjy991/nodeservice5")
        }
    }

    stage('Test image') {
        /* Ideally, we would run a test framework against our image.
        * For this example, we're using a Volkswagen-type approach ;-)
        */

        app1.inside {
            sh 'echo "Tests passed"'
        }
        app2.inside {
            sh 'echo "Tests passed"'
        }
        app3.inside {
            sh 'echo "Tests passed"'
        }
        app4.inside {
            sh 'echo "Tests passed"'
        }
        app5.inside {

```

```

        sh 'echo "Tests passed"'
    }
}

stage('Push image') {
    /* Finally, we'll push the image with two tags:
    * First, the incremental build number from Jenkins
    * Second, the 'latest' tag.
    * Pushing multiple tags is cheap, as all the layers are
reused. */
    docker.withRegistry('https://index.docker.io/v1/', 'docker-hub-
credentials') {
        app1.push("${env.BUILD_NUMBER}")
        app2.push("${env.BUILD_NUMBER}")
        app3.push("${env.BUILD_NUMBER}")
        app4.push("${env.BUILD_NUMBER}")
        app5.push("${env.BUILD_NUMBER}")

    }
}

stage('change file') {
    sh './buildjson.sh snjy991/nodeservice:${BUILD_NUMBER}'
}

withAWS(region:'us-east-1',credentials:'nameOfSystemCredentials') {
    s3Upload(file:'Dockerrun.aws.json',
bucket:'mysamplenodejs1', path:'Dockerrun.aws.json')
}

}

```

Steps in AWSCodepipeline:

I used aws pipeline because I am changing my Dockerrun.aws.json file for each build and invoking elastic beanstalk from there. Because that will pull my correct docker image which I pushed with BuildNumber tag in docker hub. So for that I used S3 bucket with versioning enabled so I can track the changes.

So my awscode pipeline is invoking every time when the content of Dockerrun.aws.json is changing inside S3 and deploying my new version of code through awsbeanstalk



testnodeapp

[View pipeline history](#)

View progress and manage your pipeline.

Edit

Release change

Source

Source



Amazon S3

✓ Succeeded just now



Source: Amazon S3 versi...



Staging

snjynode



AWS Elastic Beanstalk

✓ Succeeded just now



Source: Amazon S3 versi...

Dockerrun.aws.json file contains a configuration like

```
{
  "AWSEBDockerrunVersion": 2,
  "containerDefinitions": [
    {
      "name": "mywebapp1",
      "image": "snjy991/nodeservice1:49",
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 8081,
          "containerPort": 9090
        }
      ],
      "links": [
        "mywebapp2","mywebapp3","mywebapp4","mywebapp5"
      ]
    },
    {
      "name": "mywebapp2",
      "image": "snjy991/nodeservice2:49",
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 8082,
          "containerPort": 9090
        }
      ]
    },
    {
      "name": "mywebapp3",
      "image": "snjy991/nodeservice3:49",
      "essential": true,
      "memory": 128,
      "portMappings": [
        {
          "hostPort": 8083,
          "containerPort": 9090
        }
      ]
    },
    {
      "name": "mywebapp4",
      "image": "snjy991/nodeservice4:49",
      "essential": true,
```

```

    "memory": 128,
    "portMappings": [
      {
        "hostPort": 8084,
        "containerPort": 9090
      }
    ]
  },
  {
    "name": "mywebapp5",
    "image": "snjy991/nodeservice5:49",
    "essential": true,
    "memory": 128,
    "portMappings": [
      {
        "hostPort": 8085,
        "containerPort": 9090
      }
    ]
  }
]
}

```

Steps in ElasticBeanStalk :

- Create a new environment and application.
- Create a web server environment and select a Multi-Container Docker platform
- Given a source as my s3 bucket url from which I will receive Dockerrun.aws.json for that I have to create a S3 bucket policy and added that policy to my elastic bean stalk roles.
- Select a load balancing as we using classic load balancer which are listening to my http port 80
- Select a Deployment preference as we have asked update the code in batches of 30%.
- Select a Ec2 instance type and keypair and I have installed my nginx server on that and which is working as application load balancer.

All Applications > testfinal > snynode (Environment ID: e-ad2hqir8b, URL: [snynode.us-east-1.elasticbeanstalk.com](#)) Actions ▾

Dashboard Overview Refresh

Configuration

Logs

Health Ok Causes

Monitoring

Alarms

Managed Updates


Events

Tags

Running Version

code-pipeline-1521746925515-tEWa5_BguxDE66zwUebhgh23Ni.LWdKd

Upload and Deploy

 Configuration

64bit Amazon Linux 2017.09
v2.9.0 running Multi-container
Docker 17.12.0-ce (Generic)
Newer version available

Change

Recent Events Show All

Time	Type	Details
2018-03-22 12:29:20 UTC-0700	INFO	Environment update completed successfully.
2018-03-22 12:29:20 UTC-0700	INFO	New application version was deployed to running EC2 instances.

All Applications > testfinal > snynode (Environment ID: e-ad2hqir8b, URL: [snynode.us-east-1.elasticbeanstalk.com](#)) Actions ▾

Dashboard

Configuration

Logs

Health

Monitoring

Alarms

Managed Updates

Events

Tags

Application Deployments

The following settings control how application versions are deployed to instances.

Deployment policy: [Learn more](#)

Batch type: Specifies whether to batch deployments to a percentage or fixed number of instances.

Batch size: Specifies the percentage or number of instances on which to deploy an application version at any given time.

Configuration Updates

The following settings control how changes to the environment's instances are propagated.

Rolling update type: [Learn more](#)

Maximum batch size: The maximum number of instances that should be modified at any given time.

Minimum instances in service: The minimum number of instances that should be in service at any given time.

Pause time: Hour Minutes Seconds

Nginx Configuration

Below I configuration I used to handle my request for multiple container.

```
server {
    listen      80 default_server;
    server_name localhost;
    root        /usr/share/nginx/html;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
        proxy_pass 'http://localhost:8081/';
    }
}
```



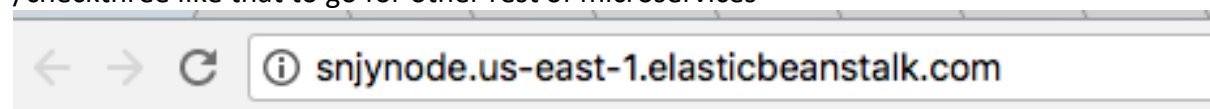
```
location /checktwo {  
    proxy_pass 'http://localhost:8082/checktwo';  
}  
  
location /checkthree {  
    proxy_pass 'http://localhost:8083/checkthree';  
}  
  
location /checkfour {  
    proxy_pass 'http://localhost:8084/checkfour';  
}  
  
location /checkfive {  
    proxy_pass 'http://localhost:8085/checkfive';  
}
```

Final URL of my elasticbeanstalk:

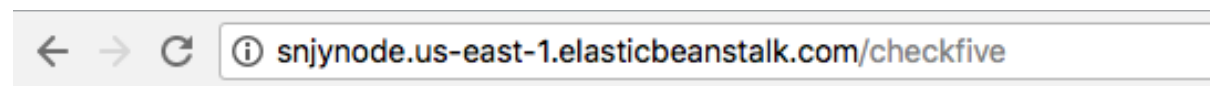
<http://snjynode.us-east-1.elasticbeanstalk.com/>

/checktwo

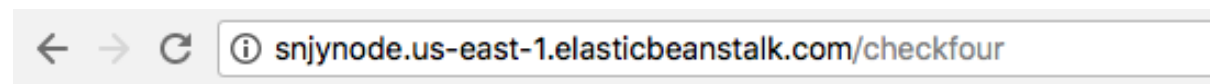
/checkthree like that to go for other rest of microservices



Hello world my changed now final test now



Hello world m from 5th service



Hello world m from 4th service