Deploying Grouper Standalone Container - AWS and Elastic Container Service (ECS)

This is an example of a CI/CD pipeline based on Jenkins and AWS Elastic Container Service (ECS) that will build a Grouper Docker container based on the InCommon Trusted Access Platform containers (https://hub.docker.com/u/i2incommon). If the build is successful, it will push the new image to a private AWS Elastic Container Registry (ECR). In this case, the build host runs with an IAM role that with an IAM policy attached to it that allows it access to push to this repo.

Jenkinsfile

```
import groovy.json.JsonSlurperClassic
// IMPORTANT: to be able to determine the right S3 bucket to pull the config from (dev, staging or prod)
// The branch name MUST end with either -dev, -staging, or -prod

pipeline {
    agent {
        // This is important, it tells Jenkins what worker nodes to use
        // Nodes with this label have an IAM role that is allowed to assume a role
        // in the ICP account that has a very stripped down IAM policy
        label 'icp_ops'
    }
    environment {
        // AWS Docker Repo for Collab Platform
        maintainer = "************.dkr.ecr.us-east-2.amazonaws.com"
        // name of container, uses the FQDN of the collab host
        imagename = "${env.JOB_NAME.tokenize('/')[1]}"
        // tag, should be based on branch and is defined in the build stage below
        tag = ""
        // environment, used to determine which S3 bucket to pull configuration from
        // e.g. dev is s3://****-dev-host-configs, staging is s3://****-staging-host-configs
        // prod is s3://****-prod-host-configs
        shortenv = env.BRANCH_NAME.substring(env.BRANCH_NAME.lastIndexOf("-") + 1, env.BRANCH_NAME.length())

        // temp variable which will be replaced soon
        s3url = "s3://****-${shortenv}-host-configs"
        // The role to assume that is allowed to push to the ICP AWS Docker repo and read s3 bucket
        role_to_assume = "arn:aws:iam::************:role/ti-prod-jenkins"
        // Not super important, just a session name to associate with assuming role, value doesn't seem to
matter
        role_session_name = "***-ops"
        // placeholder for AWS credentials that will be populated when the sts-assume-role
        // command is issued. This uses the IAM role of the running Jenkins, currently "jenkins-agent-icp-ops"
        // in AWS account ************ (also known as TI Prod)
        // This role is able to assume the role "ti-prod-jenkins" in account ************
        AWS_ACCESS_KEY_ID = ""
        AWS_SECRET_ACCESS_KEY = ""
        AWS_SESSION_TOKEN = ""
        region = "us-east-2"
    }
    stages {
        stage('Setting build context') {
            steps {
                script {
                    if (env.BRANCH_NAME == "master") {
                        tag = "latest-test"
                    } else {
                        tag = env.BRANCH_NAME
                    }
                    sh 'mkdir -p tmp && mkdir -p bin'
                    dir('tmp') {
                        git([url: "https://github.internet2.edu/docker/util.git", credentialsId: "jenkins-
github-access-token"])
                        sh 'rm -rf ../bin/*'
                        sh 'mv ./bin/* ../bin/.'
                    }
                }
            }
        }
```

```
        stage('Clean') {
            steps {
                script {
                    try {
                        sh 'bin/destroy.sh >> debug'
                    } catch (error) {
                        def error_details = readFile('./debug');
                        def message = "BUILD ERROR: There was a problem building the Base Image. \n\n
${error_details}"
                        sh "rm -f ./debug"
                        handleError(message)
                    }
                }
            }
        }
        stage('Build') {
            steps {
                script {
                    // This will return a JSON object with credentials that can be parsed and used to carry out
the AWS
                    // operations needed to build and push the container to the Collab AWS account
                    def stsjson= sh (
                        script: "aws sts assume-role --role-arn ${role_to_assume}  --role-session-name
${role_session_name}",
                      returnStdout: true
                      ).trim()
                    def ststoken = new JsonSlurperClassic().parseText(stsjson)
                    AWS_ACCESS_KEY_ID = "${ststoken.Credentials.AccessKeyId}"
                    // print "access key = ${AWS_ACCESS_KEY_ID}"
                    AWS_SECRET_ACCESS_KEY = "${ststoken.Credentials.SecretAccessKey}"
                    // print "secret key = ${AWS_SECRET_ACCESS_KEY}"
                    AWS_SESSION_TOKEN = "${ststoken.Credentials.SessionToken}"
                    // print "session token = ${AWS_SESSION_TOKEN}"
                    try {
                        docker.withRegistry("${maintainer}/${imagename}") {
                            try {
                                baseImg = docker.build("${imagename}", "--no-cache --build-arg
AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID} --build-arg AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY} --build-arg
AWS_SESSION_TOKEN=${AWS_SESSION_TOKEN} --build-arg s3url=${s3url} --build-arg ENVIRON=${shortenv} .")
                            } catch (error) {
                                handleError(error)
                            }
                        }
                     } catch (error) {
                          handleError(error)
                     }
                }
            }
        }
        stage('Test') {
            steps {
                script {
                    try {
                        echo "need to create tests"
                        //sh 'bin/test.sh 2>&1 | tee debug ; test ${PIPESTATUS[0]} -eq 0'
                    } catch (error) {
                        handleError(message)
                    }
                }
            }
        }

        stage('Push') {
            steps {
                script {
                    withEnv(["AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}",
"AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}", "AWS_SESSION_TOKEN=${AWS_SESSION_TOKEN}"  ]) {
                        def ecrlogincmd= sh (
                            script: "aws ecr get-login --no-include-email --region ${region}",
                          returnStdout: true
                        ).trim()
```

```
                        sh "${ecrloginсmd}"
                        docker.withRegistry("https://${maintainer}") {
                            baseImg.push("${tag}")
                        }
                    }
                }
            }
        }
        stage('Notify') {
            steps {
                echo "${imagename}:${tag} pushed to ${maintainer}"
                slackSend color: "good", message: "${maintainer}/${imagename}:${tag} pushed to ECR"
            }
        }
    }
    post {
        always {
            echo "Done Building."
        }
        failure {
            slackSend color: "good", message: "Build failed"
            handleError("BUILD ERROR: There was a problem building ${maintainer}/${imagename}:${tag}.")
        }
    }
}


def handleError(String message) {
    echo "${message}"
    currentBuild.setResult("FAILED")
    slackSend color: "danger", message: "${message}"
    //step([$class: 'Mailer', notifyEveryUnstableBuild: true, recipients: 'chubing@internet2.edu',
sendToIndividuals: true])
    sh 'exit 1'
}
```

The following is a Dockerfile that is used by the above build script that layers in the necessary local configurations for running Grouper needed by the Internet2 Collaboration Platform that houses spaces.internet2.edu. The files that contain secrets are located in an encrypted S3 bucket and the build host runs with an IAM role that allows it to access this bucket. This example is for a Grouper UI container. The AWS RDS (Relational Database Service) is used by this service.

Dockerfile

```
FROM i2incommon/grouper:4.1.6

ARG s3url
ARG ENVIRON
ARG AWS_ACCESS_KEY_ID
ARG AWS_SECRET_ACCESS_KEY
ARG AWS_SESSION_TOKEN

ENV ENV=$ENVIRON
ENV USERTOKEN=ICP
ENV S3_BUCKET=$s3url/grouper.at.internet2.edu

# Bring in public IdP signing cert from AWS
ADD https://s3.us-east-2.amazonaws.com/****-metadata-public/certs/icmp_signing.crt /etc/pki/tls/certs/

# Install and Upgrade pip, install awscli for getting making calls to AWS.
RUN  yum -y install python3-pip && \
     pip3 install --upgrade pip && \
     pip3 install awscli

# Bring in sp cert and private key from AWS S3
RUN aws s3 cp $S3_BUCKET/shib/sp-cert.pem /etc/shibboleth && \
    aws s3 cp $S3_BUCKET/shib/sp-key.pem /etc/shibboleth

# Clears the Default SSL config for Apache
RUN cp /dev/null /etc/httpd/conf.d/ssl-enabled.conf

# Copy in the files that congigure SAML, Apache, and Tomcat Catalina
COPY container_files/shibboleth/ /etc/shibboleth/
COPY container_files/httpd/grouper.conf /etc/httpd/conf.d
COPY container_files/tomee/server.xml /opt/tomee/conf/

# Copy in mariadb client for reading external DBs
COPY container_files/lib/ /opt/grouper/grouperWebapp/WEB-INF/lib/

# Copy in the Grouper properties files in Git
COPY container_files/grouper/ /opt/grouper/grouperWebapp/WEB-INF/classes/

# Update configuration variables based on ENV
COPY container_files/bin/setenvironment.sh /usr/local/bin/
RUN chmod 755 /usr/local/bin/setenvironment.sh && /usr/local/bin/setenvironment.sh
```

The following is an example Elastic Container Service (ECS) task definition for running this container. Configuration such as log groups for Cloudwatch, the port to direct traffic from the Elastic Load Balancer (ELB), the memory requirements for the container, as well as the ECR location to pull the container from. Note the "command" value, as this instructs the container as what component to act as. In this case, it is going to run as a Grouper UI. Alternately, the command could be "loader" (to run as a Grouper Loader), or "ws" (to run as a Grouper Web Services container). This task definition is part of an overall Cloudformation template that builds out the environment that houses other TIER containers used by the Internet2 Collaboration Platform.

```
{
    "taskDefinitionArn": "arn:aws:ecs:us-east-2:************:task-definition/dev-grouper-at:51",
    "containerDefinitions": [
        {
            "name": "dev-grouper-at",
            "image": "************.dkr.ecr.us-east-2.amazonaws.com/grouper.at.internet2.edu:v0.6.4-4.1.6-dev",
            "cpu": 0,
            "memory": 2048,
            "portMappings": [
                {
                    "containerPort": 80,
                    "hostPort": 0,
                    "protocol": "tcp"
                }
            ],
```

```
            "essential": true,
            "command": [
                "ui"
            ],
            "environment": [],
            "mountPoints": [],
            "volumesFrom": [],
            "secrets": [
                {
                    "name": "GROUPER_MORPHSTRING_ENCRYPT_KEY",
                    "valueFrom": "arn:aws:secretsmanager:us-east-2:************:secret:
R_GROUPER_MORPHSTRING_ENCRYPT_KEY"
                },
                {
                    "name": "GROUPER_DATABASE_PASSWORD",
                    "valueFrom": "arn:aws:secretsmanager:us-east-2:************:secret:
R_GROUPER_DATABASE_PASSWORD"
                },
                {
                    "name": "GROUPER_LDAP_PASSWORD",
                    "valueFrom": "arn:aws:secretsmanager:us-east-2:************:secret:R_GROUPER_LDAP_PASSWORD"
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-group": "iam-dev",
                    "awslogs-region": "us-east-2",
                    "awslogs-stream-prefix": "grouper"
                }
            }
        }
    ],
    "family": "dev-grouper-at",
    "executionRoleArn": "arn:aws:iam::************:role/ecsRoleSecretsManagerRead",
    "revision": 51,
    "volumes": [],
    "status": "ACTIVE",
    "requiresAttributes": [
        {
            "name": "com.amazonaws.ecs.capability.logging-driver.awslogs"
        },
        {
            "name": "ecs.capability.execution-role-awslogs"
        },
        {
            "name": "com.amazonaws.ecs.capability.ecr-auth"
        },
        {
            "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
        },
        {
            "name": "ecs.capability.secrets.asm.environment-variables"
        },
        {
            "name": "ecs.capability.execution-role-ecr-pull"
        }
    ],
    "placementConstraints": [],
    "compatibilities": [
        "EXTERNAL",
        "EC2"
    ],
    "requiresCompatibilities": [
        "EC2"
    ],
    "memory": "2048",
    "registeredAt": "2023-05-15T19:15:02.832Z",
    "tags": []
}
```

The following is an apache configuration for running the Grouper container in ECS. **Since containers have to sit behind load balancers, the remote IP address is that of the load balancer. However, the load balancer inserts a header (X-Forwarded-For) that contains the IP address of the actual client, so that is logged.** Rather than having the httpd listen on 2 ports, if the request comes in as http (as seen in the X-Forwarded-Proto), it will redirect to https.

grouper.conf

```
ServerName https://grouper.at.internet2.edu:443
UseCanonicalName On

RemoteIPHeader X-Forwarded-For

RewriteEngine On
RewriteCond %{X-Forwarded-Proto}i http
RewriteRule ^ https://%{HTTP_HOST}:443%{REQUEST_URI} [R=302,L,QSA]

RewriteEngine on
RewriteRule   "^/$"  "/grouper/"  [R]

# log the X-Forwarded-For (the real client IP) header if present
LogFormat "httpd;access_log;%{ENV}e;%{USERTOKEN}e;%{X-Forwarded-For}i %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%
{User-Agent}i\"" proxy
SetEnvIf X-Forwarded-For "^.*\..*\..*\..*" forwarded
CustomLog "/tmp/logpipe" combined env=!forwarded
CustomLog "/tmp/logpipe" proxy env=forwarded

ErrorLogFormat "httpd;error_log;%{ENV}e;%{USERTOKEN}e;[%{u}t] [%-m:%l] [pid %P:tid %T] %7F: %E: [client\ %a] %
M% ,\ referer\ %{Referer}i"
```

Note: Since TLS termination typically happens at the ELB, inform Tomcat that it should create URLs of the form https by specifying

secure="true" in the Connector declaration.