16 MARCH 2022

GOWRI SHANKAR PONNUSAMY

SECURITY ASSESMENT

DELIO GROUP

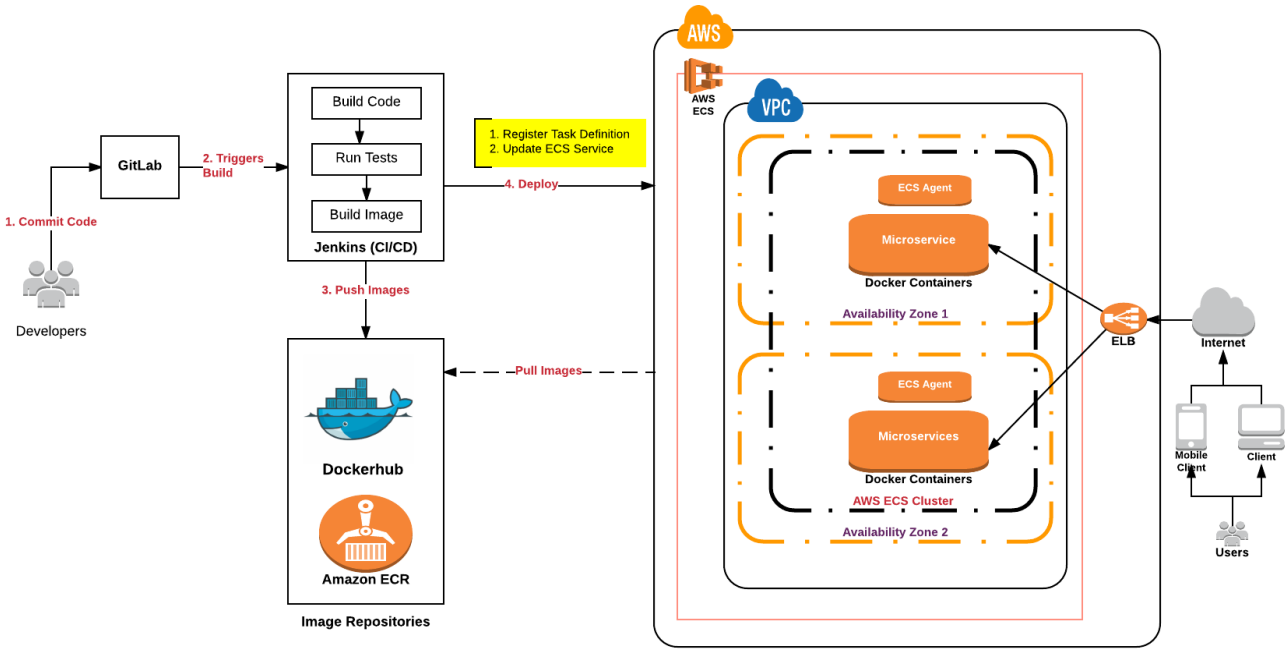
Delio Engineering has decided to start selling "Delio" branded hoodies and coasters. We have developed an API back-end written in Node JS and started building the infrastructure in Terraform. Before the code is deployed the Head of TechOps has asked you to conduct a security audit of the current codebase.

Proposed architecture for deployment:

This is simple web application, we will be deploying it as docker container in AWS. We can using either AWS Elastic Continuer Service (ECS) or AWS Elastic Kubernetes Service (EKS). For testing purpose, I will go with EKS to keep things simple. In the architecture, am going to assume that all vulnerabilities are remediated and we

will be deploy application post remediation (For vulnerabilities identified and remediated, please refer to next section).

Below is the developer architecture for amazon ECS service. As part of Jenkins build pipeline, we will be trigger AWS ECR image vulnerability scanning. Alternatively if we are using some other docker registry, we can use other tools like Stackrox (Red Hat Advanced Cluster Security for Kubernetes), Twistlock, Trend Micro etc.

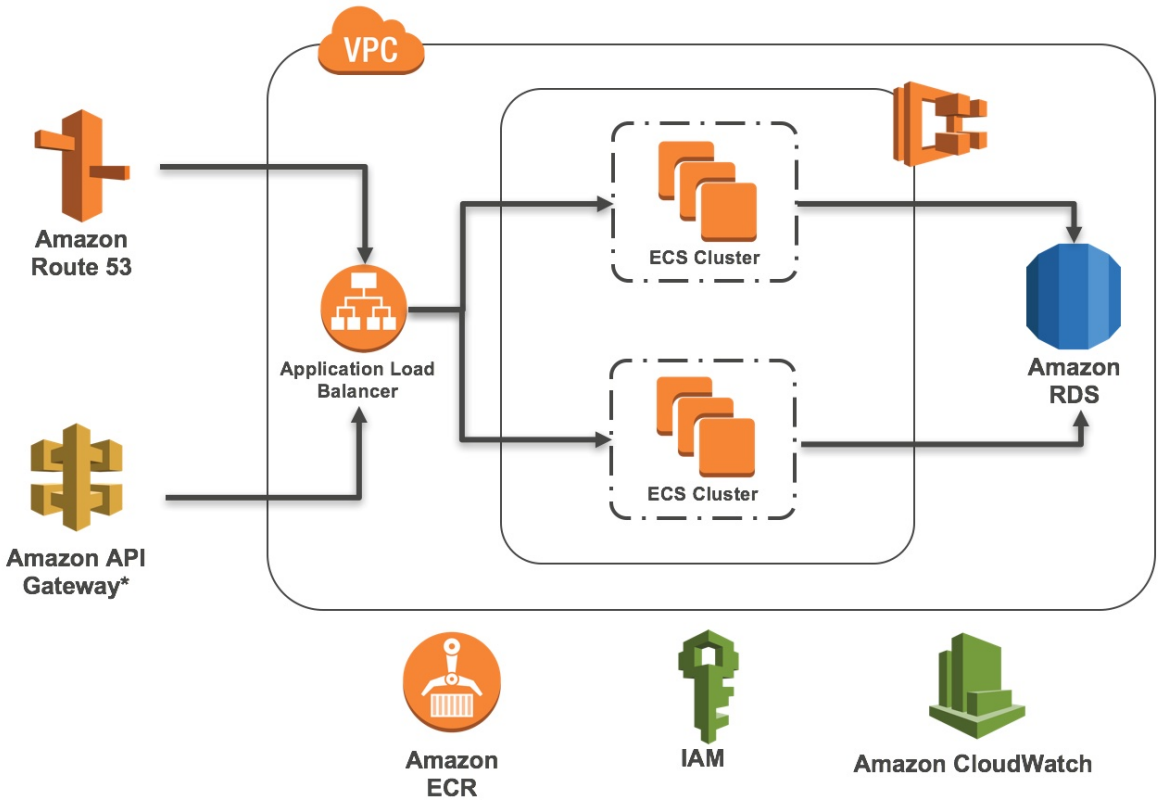


AWS ECR provides both static/build scanning and also dynamic image scanning. Unfortunately dynamic scanning of images has to be enabled by deploying addition utilities (https://github.com/aws-samples/amazon-ecr-continuous-scan) to schedule docker image scanning.

While products like stackrox, twist lock provides both build/static and dynamic scanning of images and more advanced features runtime protection like network segmentation, risk profiling, runtime detection and response etc.

Overall Architecture Set up:

Below is the overall architecture of the application. I would recommend enabling AWS Shield Advanced if below will evolve into a critical application for business.



GIT REPO SCAN ANALYSIS:

NODE JS LIBRARY VULNERABILITIES:

There are lot of known vulnerabilities that are part of the json libraries used by the application. Most of them can be remediated by upgrading libraries as mentioned below

knex -> upgrade to 1.0.4

express -> Upgrade to 4.17.3

Json library vulnerabilities can be identified and fixed as part of CI/CD pipe line using npm audit fix --force

```
# npm audit report

connect  <=2.8.0
methodOverride Middleware Reflected Cross-Site Scripting in connect -
https://github.com/advisories/GHSA-3fw8-66wf-pr7m
fix available via `npm audit fix --force`
Will install express@4.17.3, which is a breaking change
node_modules/connect
  express  <=3.10.5 || 4.0.0-rc1 - 4.13.4 || 5.0.0-alpha.1 - 5.0.0-
alpha.2
  Depends on vulnerable versions of connect
  Depends on vulnerable versions of mime
  Depends on vulnerable versions of qs
  node_modules/express

express  <=3.10.5 || 4.0.0-rc1 - 4.13.4 || 5.0.0-alpha.1 - 5.0.0-alpha.2
Severity: high
No Charset in Content-Type Header in express - https://github.com/
advisories/GHSA-gpvr-g6gh-9mc2
Depends on vulnerable versions of connect
Depends on vulnerable versions of mime
Depends on vulnerable versions of qs
fix available via `npm audit fix --force`
Will install express@4.17.3, which is a breaking change
node_modules/express

knex  <=0.19.4
Severity: critical
SQL Injection in knex - https://github.com/advisories/GHSA-58v4-qwx5-7f59
Depends on vulnerable versions of underscore
fix available via `npm audit fix --force`
Will install knex@1.0.4, which is a breaking change
node_modules/knex

mime  <1.4.1
Severity: moderate
Regular Expression Denial of Service in mime - https://github.com/
advisories/GHSA-wrvr-8mpx-r7pp
fix available via `npm audit fix --force`
Will install express@4.17.3, which is a breaking change
node_modules/mime
  express  <=3.10.5 || 4.0.0-rc1 - 4.13.4 || 5.0.0-alpha.1 - 5.0.0-
alpha.2
  Depends on vulnerable versions of connect
  Depends on vulnerable versions of mime
  Depends on vulnerable versions of qs
  node_modules/express

qs  <=6.0.3
```

```
Severity: high
Prototype Pollution Protection Bypass in qs - https://github.com/
advisories/GHSA-gqgv-6jq5-jjj9
Denial-of-Service Extended Event Loop Blocking in qs - https://
github.com/advisories/GHSA-f9cm-p3w6-xvr3
fix available via `npm audit fix --force`
Will install express@4.17.3, which is a breaking change
node_modules/qs
  express  <=3.10.5 || 4.0.0-rc1 - 4.13.4 || 5.0.0-alpha.1 - 5.0.0-
alpha.2
  Depends on vulnerable versions of connect
  Depends on vulnerable versions of mime
  Depends on vulnerable versions of qs
  node_modules/express

underscore  1.3.2 - 1.12.0
Severity: high
Arbitrary Code Execution in underscore - https://github.com/advisories/
GHSA-cf4h-3jhx-xvhq
fix available via `npm audit fix --force`
Will install knex@1.0.4, which is a breaking change
node_modules/underscore
  knex  <=0.19.4
  Depends on vulnerable versions of underscore
  node_modules/knex

6 vulnerabilities (1 low, 1 moderate, 3 high, 1 critical)
```

WEB SERVER SETUP:

Nodejs express web server listens on http protocol. We should configure it to use https. Alternate option will to do ssl offloading at load balancers like F5. But in public, it's always good to ensure data is secure till it reaches web server. I have made code change for the same

HARD CODED PASSWORD:

Hard coded passwords in source code is always bad security practice. Passwords should be encrypted and shared only on need basis. This is critical to prevent any accidental leakage and also prevent insider attack.

There are 3 places, where passwords has been hard coded.

- app.js - Database and password information is hardcoded
- .env file (Authentication information for application is hard coded)
- terraform/modules/rds/variable.tf

There are many tools that can be used by enterprise to perform hard coded password or secrets canning in a git repo. There are few open source tools like git-secret, wisher that can be integrated into CI/CD pipeline to scan for hardcoded passwords. Otherwise, we can use GitHub secret scanning partner program (https://docs.github.com/en/developers/overview/secret-scanning-partner-program)

In addition, we should also consider runtime issues like storing Terraform state file. If we are going to store it locally/repo, in addition to lock issue we will expose secrets. I have modified the terraform variable for password as sensitive to stop it from logged and also added code to store the state file in S3 bucket and version info in dynamodb.

Also, To protect terraform passwords are stored in secured vault, there are multiple solutions available like AWS Secret manager, Harsicorp Valut etc. I have modified rds configuration to use password from AWS secret manager.

```
locals {

  db_creds = sensitive(jsondecode(
    data.aws_secretsmanager_secret_version.creds.secret_string
  ))
}
```

We will add the sensitive data to the EKS pod using Secrets Store Drive in CI/CD pipeline. For more information, please refer to (https://secrets-store-csi-driver.sigs.k8s.io/).

helm repo add secrets-store-csi-driver https://kubernetes-sigs.github.io/secrets-store-csi-driver/charts

helm install -n kube-system csi-secrets-store secrets-store-csi-driver/secrets-store-csi-driver

kubectl apply -f https://raw.githubusercontent.com/aws/secrets-store-csi-driver-provider-aws/main/deployment/aws-provider-installer.yaml

```
spec:
  serviceAccountName: devopsdb-sa
  volumes:
  - name: devops-db
    csi:
      driver: secrets-store.csi.k8s.io
      readOnly: true
      volumeAttributes:
        secretProviderClass: “nodejs-deployment-aws-secrets"
  containers:
  - name: nodejs-deployment
    image: nodejs
    ports:
    - containerPort: 9000
    volumeMounts:
    - name: secrets-store-inline
      mountPath: "/mnt/secrets-store"
      readOnly: true
```

MISSING INPUT VALIDATION:

In the code for /projects, there is no input parameter validation. Input validation is required.

```
start(port) {
    dotenv.config()
    //Test url for kubelet validation
    this.epressApp.get('/',(req,res) => {
        res.status.(200).send()
        return
    }
    )

    this.expressApp.get('/projects', (req, res) => {
        //Bug in the code, !== should be ==
        if (req.headers.authorization == process.env.API_KEY) {
            res.status(200).json({ projects: this.db().raw('SELECT *
FROM projects') })
            return
```

```
            }

            res.status(401).json({error: 'authorization failure'})
        })

        this.expressApp.post('/projects', (req, res) => {
            //It should be == instead of  === bug in code
            if (req.headers.authorization == process.env.API_KEY) {
                //No validation, risk of cross scripting
                this.db().raw(`INSERT INTO projects(name, description)
VALUES (${req.params.name}, ${req.params.description})`)
                res.status(204).send()
                return
            }

            res.status(401).json({error: 'authorization failure'})
        })
```