Todo App - DEVOPS Project

By: Ali Hamouda

Description

This is a NestJs Todo Demo REST app with CI/CD workflow. AWS native solution.

Stack used

NestJS, MySQL, Docker GithubActions, AWS EC2, AWS RDS, AWS ELBv2, AWS ECS, Grafana, Prometheus, Ubuntu

Repo Folders

Infrastructure Folder

Contains the desired infrastructure of the deployed app. This description is made using AWS CDK.

Monitoring Folder

Has scripts for Monitoring Server configuration. Let's user install Grafana and Prometheus on EC2 instance automatically deployed using CDK files.

SRC

Contains the app source code.

Requirements

- AWS account
- AWS CLI installed on host machine
- .aws/credentials configured
- NodeJs installed on machine
- NestJs installed if code is going to run on dev environment
- Python3 installed
- Ubuntu as scripts are made for Ubuntu machine
- Docker and Docker-compose
- Gmail Email for Grafana Alerting

1. Philosophy of the project

This project introduces people and users to NestJs development, testing, containerization using Docker, Continuos Integration using Github Actions Workflows, Continuos Delivery using AWS Elastic Container Service a.k.a ECS and monitoring using Prometheus and Grafana.

The repo contains multiple automation scripts as part of this project.

2. App description

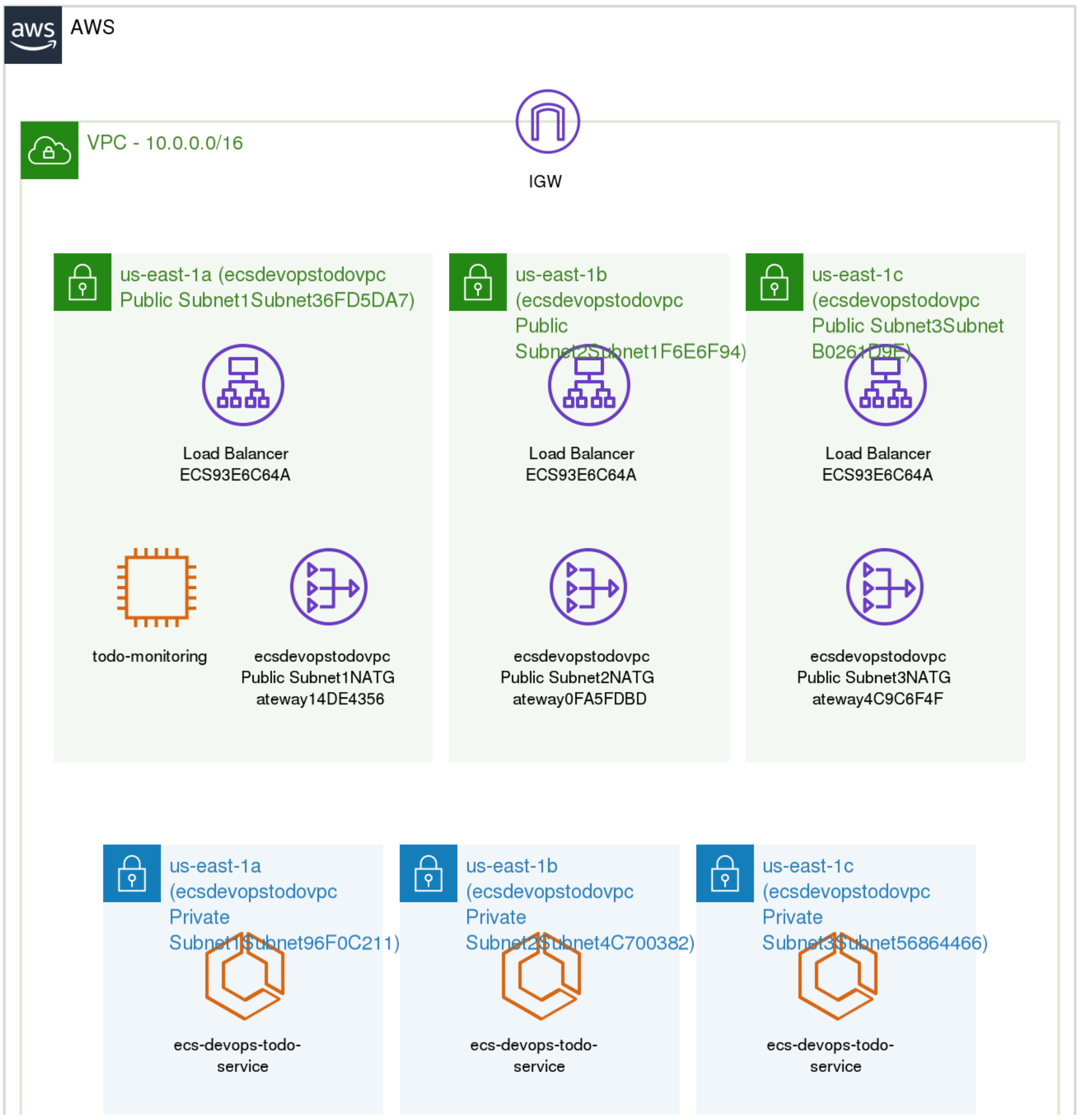
The app is a simple Todo REST app allowing user to **GET, POST, PUT, DELETE** todo. Is exposes a metrics endpoint for Prometheus for requests counter.

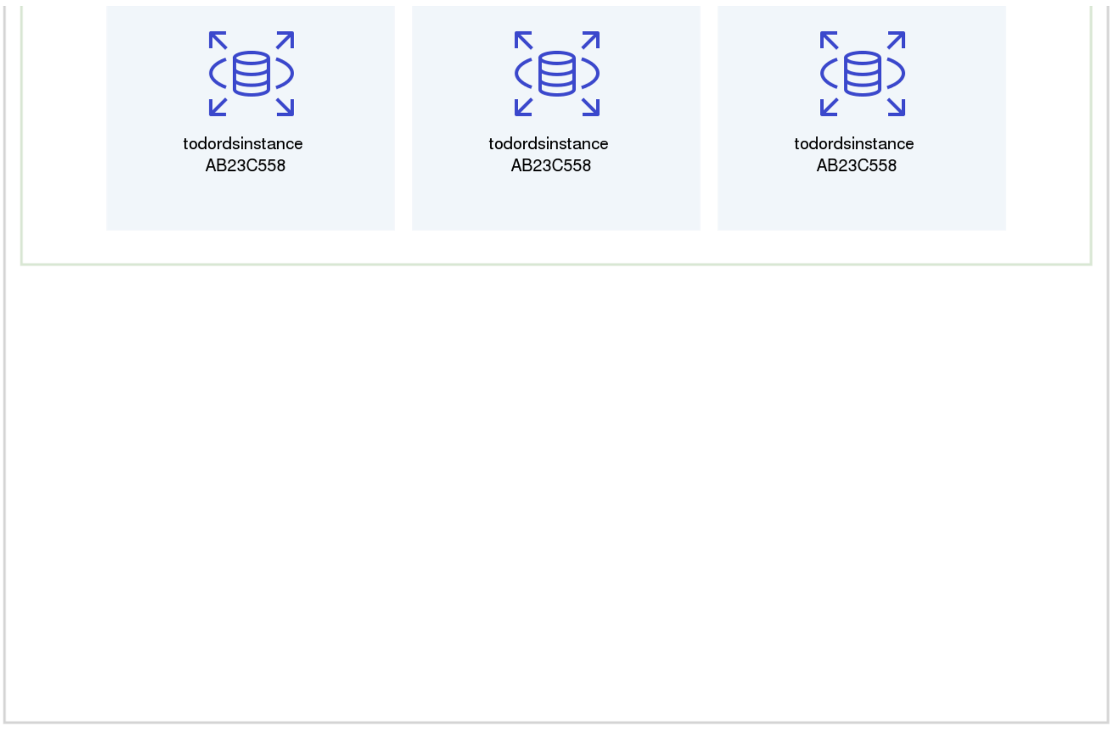
Route	Method	Functionality	Body
/api/todo	GET	Get todos from DB	-

/api/todo	POST	Add new todo object	{title: string, body: string}
/api/todo/:id	PUT	Update todo Status	{status: DONE | PENDING | STARTED}
/api/todo/:id	DELETE	Delete a todo from db by id	-
/metrics	GET	Get metrics: http_requests_total for prometheus	-

3. Solution Architecture

- I choose to set 3 Avaibility zones
- We find 1 public subnet and one private subnet (with High Availability)
- Loadbalancers get requests from InternetGateway and forward them on port 3000 to ECS instances
- EC2 Instance is facing internet through IGW on ports 80,8080,22.

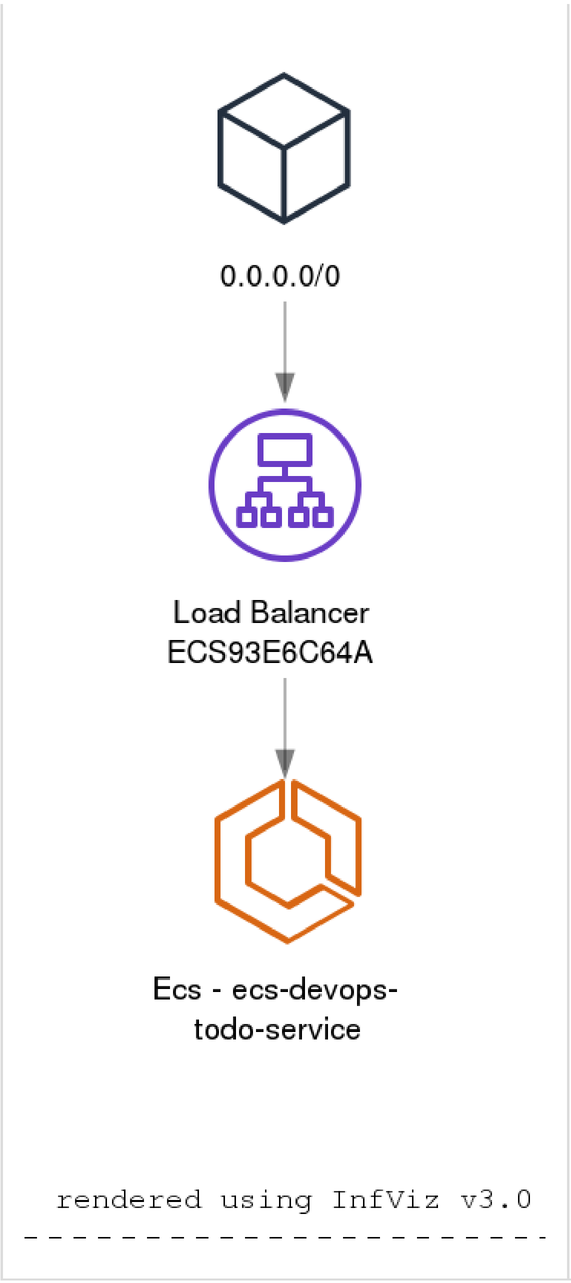

AWS
VPC - 10.0.0.0/16
IGW
us-east-1a (ecsdevopstodovpc Public Subnet1Subnet36FD5DA7)
us-east-1b (ecsdevopstodovpc Public Subnet2Subnet1F6E6F94)
us-east-1c (ecsdevopstodovpc Public Subnet3Subnet B0261D9E)
Load Balancer ECS93E6C64A
Load Balancer ECS93E6C64A
Load Balancer ECS93E6C64A
todo-monitoring
ecsdevopstodovpc Public Subnet1NATG ateway14DE4356
ecsdevopstodovpc Public Subnet2NATG ateway0FA5FDBD
ecsdevopstodovpc Public Subnet3NATG ateway4C9C6F4F
us-east-1a (ecsdevopstodovpc Private Subnet1Subnet96F0C211)
us-east-1b (ecsdevopstodovpc Private Subnet2Subnet4C700382)
us-east-1c (ecsdevopstodovpc Private Subnet3Subnet56864466)
ecs-devops-todo-service
ecs-devops-todo-service
ecs-devops-todo-service

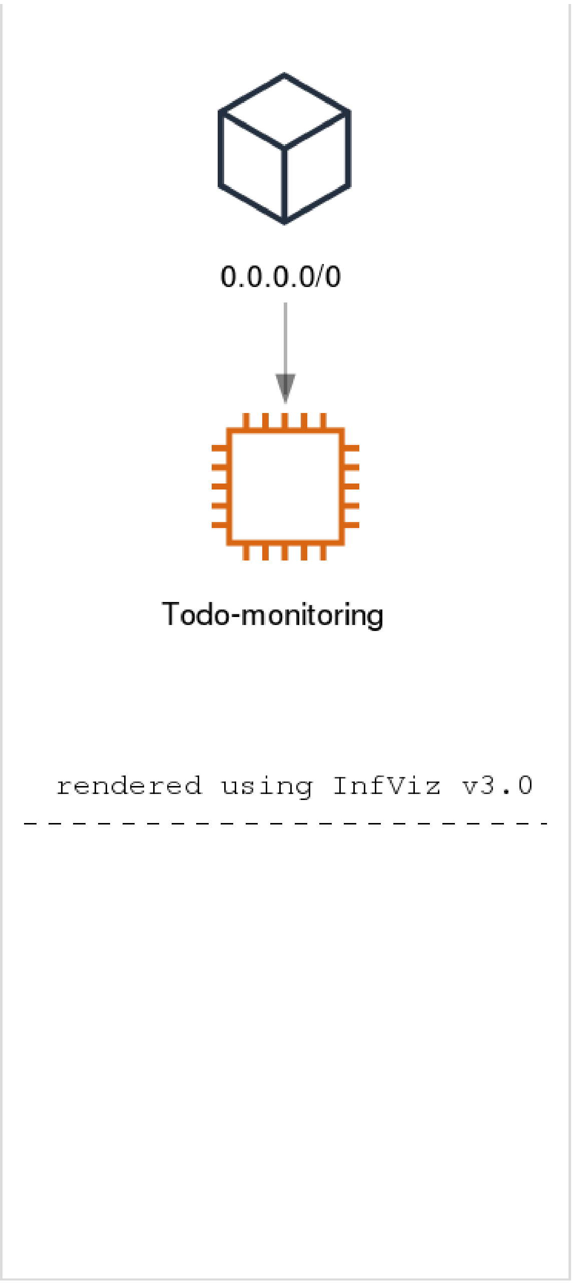


rendered using InfViz v3.0

Connectivity:

ECS	EC2


0.0.0.0/0
Load Balancer ECS93E6C64A
Ecs - ecs-devops-todo-service
rendered using InfViz v3.0



EC2 Open Ports

Ports	Description
80	Grafana is set to serve over 80
8080	Prometheus is set to serve over 8080
22	for SSH

Elastic Container Service ECS Ports

Ports	Description
3000	Container serving over 3000

Elastic Load Balancer Ports

Ports	Description
80	Listening port
3000	Forward port to ECS

4. Monitoring

The infrastructure contains an EC2 ubuntu instance for monitoring. Thi repository contains scripts to deploy Prometheus and Grafana (Configufration, Dashboard and Alerting) automatically.

Prometheus is a time series data and monitoring solution. It scrapes every **scrape_interval** data from (by default) */metrics/* of set target. No agent is needed on remote server.

Grafana is a dashboarding and alerting solution. It supports different datasources such as Prometheus, AWS Cloudwatch and others.

The script:

- Installs Prometheus
- Copy configuration files
- Installs Grafana
- Copy Grafana Files
- Send POST Requests to Grafana to save dashboard, datasource and alerting scripts

Ports are configured as below: |Ports|Service| |---|--| |80|Grafana| |8080|Prometheus|

Grafana Auto configuration

The folder **/infrastructure/grafana/** contains 3 JSON files as Datasource, Dashboard and Alert Configuration descriptions.

The script automatically deploys them using post requests to Grafana endpoints.

Prometheus configuration

The folder **/infrastructure/prometheus/** contains 2 files: service file and target configuration.

```
global:
  scrape_interval: 10s

scrape_configs:
  - job_name: 'prometheus_metrics'
    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:8080']
  - job_name: 'ecs-metrics'
    scrape_interval: 5s
    static_configs:
      - targets: ['prometheus-target-1:80']
      #Prometheus-target-1 will be replaced by LB Dns address by the script
```

5. Workflows

Workflow are made to automate solution testing and deployment.

CI Workflow

Has to jobs:

- Check if all tests pass.
- Check if Coverage Test is above a threshold.

  ### CD Workflow

  Has one Job with multiple steps:
- Checks if code is merged (loop over repo status)
- Checkout code
- Fetches for RDS endpoint using aws cli and places it inside the container task definition. (Task Definition is a JSON file that describes the deployment of the containers inside ECS. It includes network driver, volumes, Env. variables, ...)
- Places DB password, saved as secret in Github repo, inside Task Definition Env Vars.
- Log to AWS
- Build image from dockerimage file and places it in Elastic Container Registry ECR.
- Update the Task Definition to insert the new image ID.
- Deploy Task Definition on ECS
- Clean up

6. Steps for running this solution

> *I assume here that awscli is well configured and node is installed*

Deploy Infrastructure on AWS

We need first to store a secret in the AWS. Why? What is it? well it's the database password we gonne use later.

```
aws secretsmanager create-secret --name rds/todo-app  -secret-string <SECRET_PASSWORD>

> OUTPUT
> {
>   "ARN": "arn:aws:secretsmanager:AZ:user-id:secret:name_here-vJIkpB",
>   "Name": "name_here",
>   "VersionId": "50XXX15d-8cb7-XXXX-XXXX-XXXXXe21a4ad"
> }
```

Copy that ARN inside **./infrastructure/infrastructure-builder.sh** line 6 to replace existing ARN.

Then, we need to build the python app for IaC deployment.

```
cd infrastructure
chmod +x infrastructure-builder.sh
./infrastructure-builder.sh
# This will ask you for aws region and account id : a 12-digit number
```

This will create a new folder under infrastructure called *ecs-devops-todo-cdk*

Next we run the project:

```
cd ecs-devops-todo-cdk
cdk deploy
```

After a certain amount of time, the project would normally be deployed.

For Rollback:

To rollback this step, you need to to run

```
cdk destroy
```

Then log in to your AWS account and delete the ECR repository and LogGroup under CloudWatch.

Monitoring Solution Deployment

> *You can start this step even if the **cdk deploy** didn't finish yet.*

On AWS create a keypair and call it ec2nkp.pem. Place it in ~/Downloads/.

On terminal, get back to main directory **Todo-app** :

```
cd ../..
```

And get to *monitoring-scripts* folder:

```
cd monitoring-scripts

# you need to edit ./grafana/alert-conf.json
# change my emails with yours to receive alerts
...

sudo chmod +x installation-trigger.sh
./installation-trigger.sh
# This will ask for a Grafana (Dashboard) Password, email and password
# The email and password will let Grafana send alerts
# SMTP server are already configured for Gmail
```

The script will loop until EC2 Instance (for monitoring) is deployed. Then it will connect and execute a custom script. You can use your browser to connect to EC2 instance on ports for: |ports|service| |-----|-------| |80| Grafana| |8080|Prometheus|

No data is provided for Prometheus, thus Grafana, as container is not deployed yet on ECS.

Prepare AWS Deployement Workflow

Go to your Github Repository of this project. Set secrets by choosing settings-> secrets as belows:

For AWS Educate Accounts:

Secret Name	Content
AWS_ACCESS_KEY_ID	Available in aws credentials
AWS_SECRET_ACCESS_KEY	Available in aws credentials
AWS_SESSION_TOKEN	Available in aws credentials: required by the AWS Educate Account

DB_PASSWORD	DB password set in previous step as secret (not its ARN)

> *For normal accounts: you need to edit the workflow and remove AWS_SESSION_TOKEN Input*

After adding secrets, we need to run the AWS workflow in github actions: Steps will be executed automatically and solution will be deployed.

Get back to Grafana, *GetCount* will be set to 0.

ToDo

- Make scripts more abstract and available for other OSs or use tools such as Ansible.
- Expand the app for more features and why not add a facade.