Table of Contents

Learn DevOps

A bunch of devops tutorial. Need more example or have some suggestions? Email me at caknopal@gmail.com, looking forward to it.

AWS - Create Individual IAM User

In this tutorial we will learn how to create an individual IAM user.

---

1. Definitions

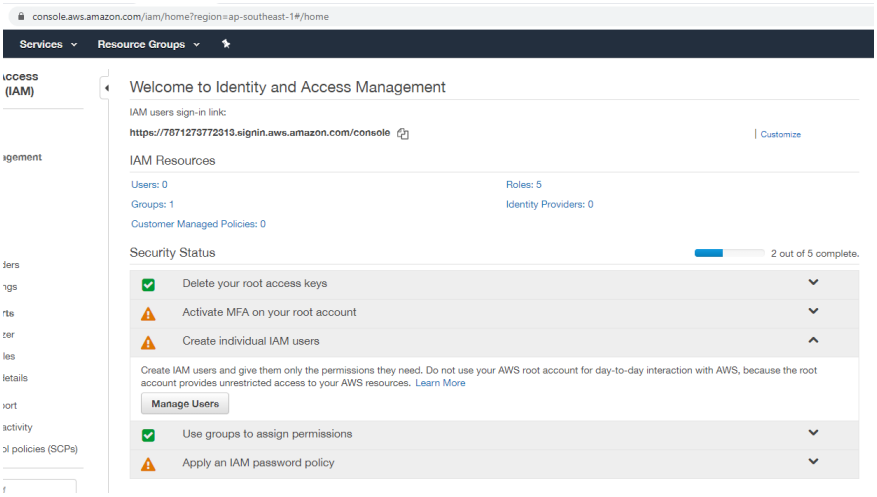
IAM or Identity and Access Management is used to securely control individual and group access to your AWS resources. You can create and manage user identities (IAM users) and grant permissions for those IAM users to access your resources.

The IAM user is similar to a AWS account user, the only differences are IAM user's permission towards AWS resources are controlled (by the AWS account user).

---

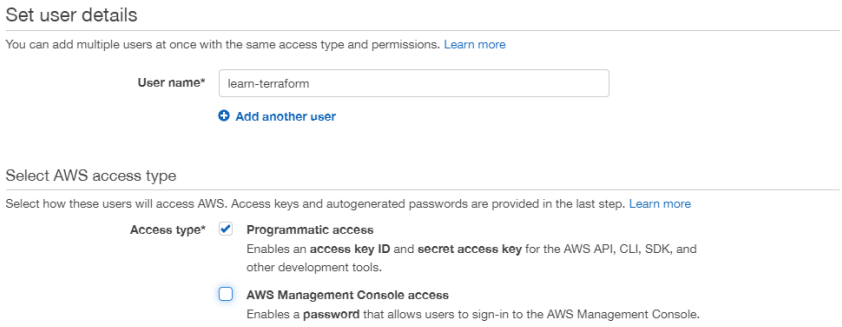
2. Create new IAM User

To create IAM user, you (as the owner of AWS account) need to login to AWS console first. Then do open AWS IAM page and click the **manage users** menu.

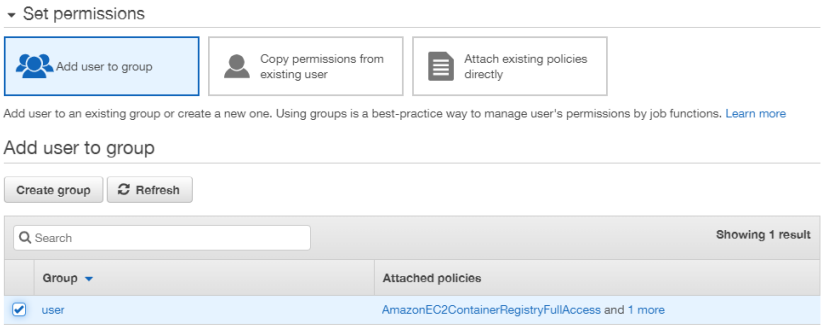


You will directed to new page that show list of created IAM users. Next, click the **Add user**, then fill the name.

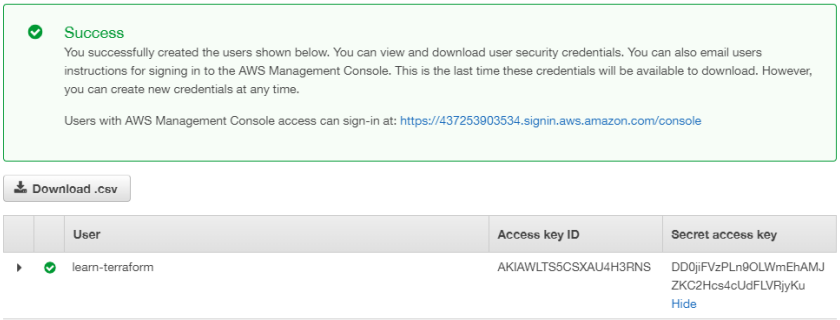
If the particular user will be used on 3rd party or AWS SDK, then do check check the **Programmatic Access** option.



Then click next to open the user group page. In here do create new group with certain access checked. For example in the image below, a new group named `user` is created with full access to EC2 features.



Do click next few times, then the user creation process will be done.



Copy the **access key ID** and **secret access key**, save it into some notes because you won't be able to see the secret key again.

Ok, that's it. The keys are ready to use.

CI/CD - Serverless Ebook using Gitbook CLI, Github Pages, Github Actions CI/CD, and Calibre

In this tutorial we are going to create an ebook instance using Github, then publish it to the Github pages in an automated manner (on every push to upstream) managed by Github Actions, and it will not deploy only the web version, but the ebook files as wall (in `.pdf`, `.epub`, and `.mobi` format).

> The very example of this tutorial is ... this website 😊
> https://devops.novalagung.com/en/

For every incoming push to the upstream, Github Actions (CI/CD) will trigger certain processes (like compiling and generating the ebook), then the result will be pushed to the `gh-pages` branch, make it publicly accessible.

---

1. Prerequisites

1.1. Gitbook CLI

Install gitbook CLI (if you haven't). Do follow the guide on https://github.com/GitbookIO/gitbook-cli.

1.2. Github account

Ensure you have a Github account.

1.3. Git client

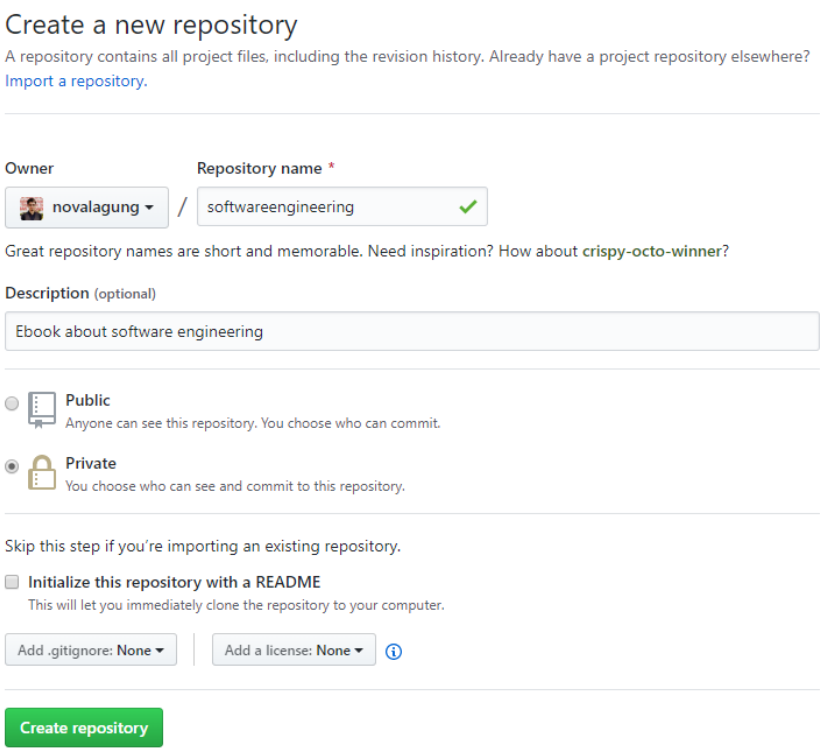
Ensure you have Git client installed in your local machine.

---

2. Guide

2.1. Create a Github repo

First, create a new repo in your Github account, it can be a private one or public, doesn't matter. Just for the sake of this tutorial, I am going to pick `softwareengineering` as the repo name.

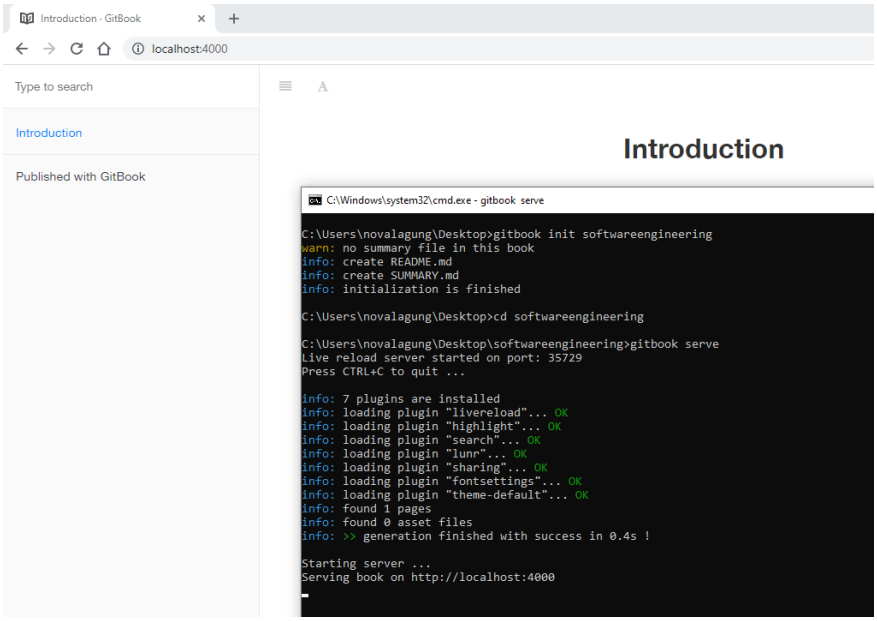


2.2. Create a new Gitbook project

Next, use `gitbook` command line to initialize a new project, name it anyhing. Here I'll use `softwareengineering`, the same one as the git repo name.

After the project setup is finished, try to test it locally.

```
gitbook init softwareengineering
cd softwareengineering
gitbook serve
```



As we can see from image above, the web version of the book is running up.

2.3. Prepare ssh Github deploy key

Next, we are going to use Github Action plugin peaceiris/actions-gh-pages to automate pushing resources from git repo server to the `gh-pages`.

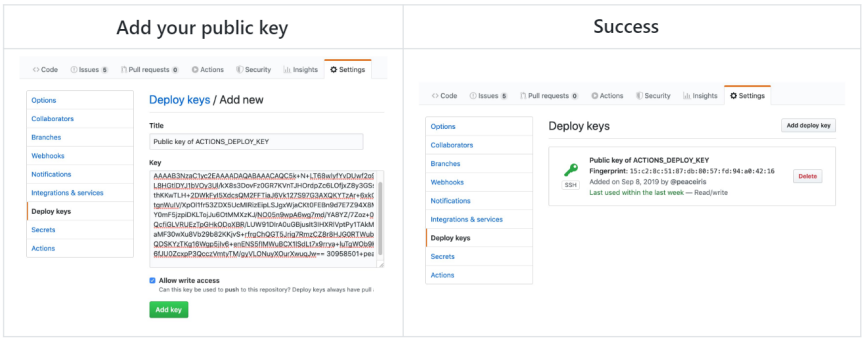
To make this scenario happen, first, generate new key pair using `ssh-keygen` command below. We will use the keys as Github deploy key.

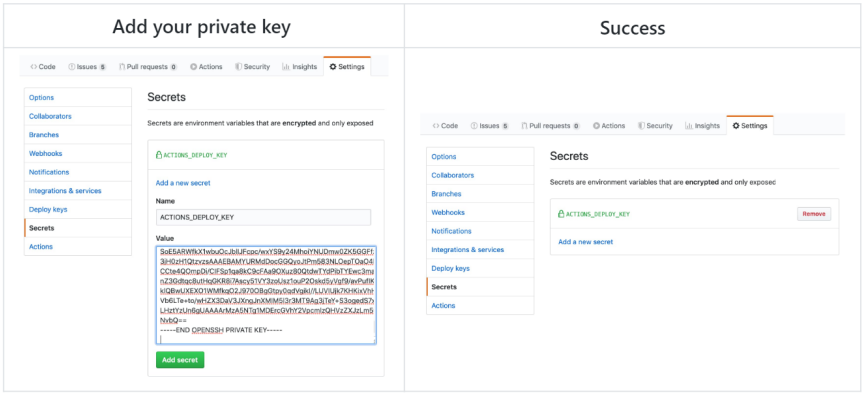
```
ssh-keygen -t rsa -b 4096 -C "$(git config user.email)" -f gh-pages -N ""
# You will get 2 files:
#   gh-pages.pub (public key)
#   gh-pages     (private key)
```

The above command generates two files:

- `gh-pages.pub` file as the public key
- `gh-pages` file as the private key

Upload these two files into repo's project keys and secret menu respectively. To do that, open the repo, click **Settings**, then do follow the steps below:





2.4. Create Github workflow CI/CD file for generating the web version of the ebook

Now we are going to make Github able to automatically deploy the web version of the ebook on every push. And we want that to be applied into the first push as well.

Create a new workflow file named `deploy.yml`, place it in `<yourproject>/.github/workflows`, then fill it with the configuration below:

```
# file ./softwareengineering/.github/workflow/deploy.yml

name: 'deploy website and ebooks'

on:
  push:
    branches:
      - master

jobs:
  job_deploy_website:
    name: 'deploy website'
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v1
    - uses: actions/setup-node@v1
      with:
        node-version: '10.x'
    - name: 'Installing gitbook cli'
      run: npm install -g gitbook-cli
    - name: 'Generating distributable files'
      run: |
        gitbook install
        gitbook build
    - uses: peaceiris/actions-gh-pages@v2.5.0
      env:
        ACTIONS_DEPLOY_KEY: ${{ secrets.ACTIONS_DEPLOY_KEY }}
        PUBLISH_BRANCH: gh-pages
        PUBLISH_DIR: ./_book
```

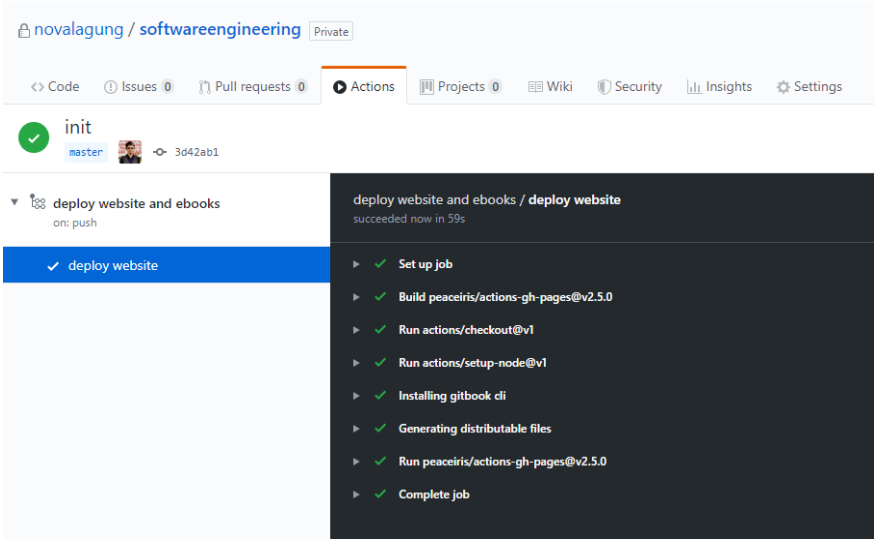
In summary, the workflow above will do these things sequentially:

- Trigger this workflow on every push happens on `master` branch.
- Install `nodejs` .
- Install `gitbook` CLI.
- Build the project.
- use `peaceiris/actions-gh-pages` plugin to deploy the built result to `gh-pages` branch. The Github deploy key that we just uploaded is used by this plugin.

2.5. Push project to Github repo

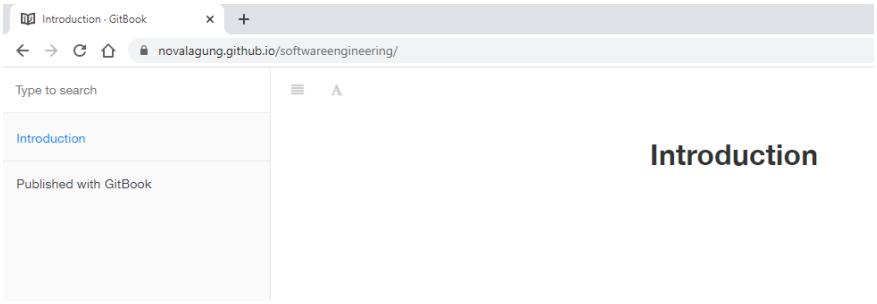
```
cd softwareengineering

# ignore certain directory
touch .gitignore
echo '_book' >> .gitignore

# init git repo
git init
git add .
git commit -m "init"
git remote add origin git@github.com:novalagung/softwareengineering.git

# push
git push origin master
```

Navigate to browser, open your Github repo, click `Actions`, watch a workflow process that currently is running.

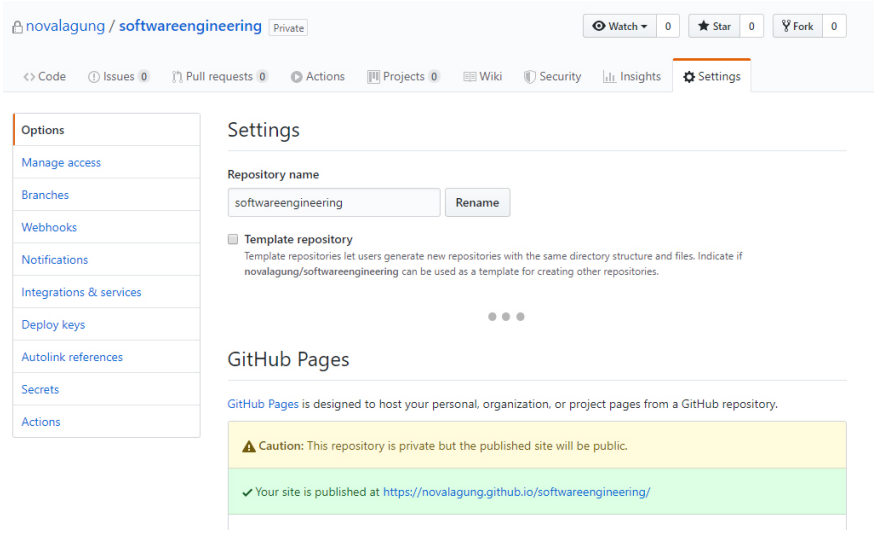


After the workflow is complete, then try to open in the browser the following URL.

```
# https://<github-username>.github.io/<repo-name>
https://novalagung.github.io/softwareengineering/
```



If you are still not sure about what is the valid URL, open **Settings** menu of your Github repo then scrolls down a little bit until **Github Pages** section appears. The Github Pages URL will appear there.



2.6. Modify the workflow file to be able to generate the ebook files

Ok, now we will modify the workflow so it will be able to generate the ebook files (`.pdf` , `.epub` , and `.mobi`), not just the web version.

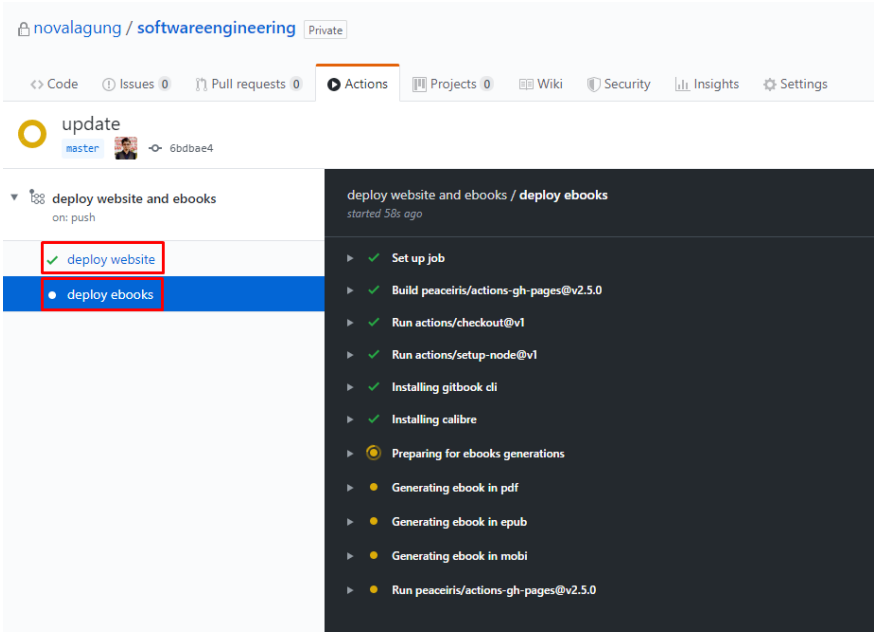
Do open the previous `deploy.yml` file, add a new job called `job_deploy_ebooks` .

```
# file ./softwareengineering/.github/workflow/deploy.yml

name: 'deploy website and ebooks'

on:
  push:
    branches:
      - master

env:
  ebook_name: 'softwareengineeringtutorial'

jobs:
  job_deploy_website:
    # ...
  job_deploy_ebooks:
    name: 'deploy ebooks'
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v1
    - uses: actions/setup-node@v1
      with:
        node-version: '10.x'
    - name: 'Installing gitbook cli'
      run: npm install -g gitbook-cli
    - name: 'Installing calibre'
      run: |
        sudo -v
        wget -nv -O- https://download.calibre-ebook.com/linux-installer.sh | su
    - name: 'Preparing for ebooks generations'
      run: |
        gitbook install
        mkdir _book
    - name: 'Generating ebook in pdf'
      run: gitbook pdf ./ ./_book/${{ env.ebook_name }}.pdf
    - name: 'Generating ebook in epub'
      run: gitbook epub ./ ./_book/${{ env.ebook_name }}.epub
    - name: 'Generating ebook in mobi'
      run: gitbook mobi ./ ./_book/${{ env.ebook_name }}.mobi
    - uses: peaceiris/actions-gh-pages@v2.5.0
      env:
        ACTIONS_DEPLOY_KEY: ${{ secrets.ACTIONS_DEPLOY_KEY }}
        PUBLISH_BRANCH: ebooks
        PUBLISH_DIR: ./_book
```

The previous `job_deploy_website` is responsible for generating the web-based version of the ebook. This newly created `job_deploy_ebooks` has different purpose, which is to generate the files version of the ebook (`.pdf` , `.epub` , `.mobi`). The generated files later will be pushed to a branch named `ebooks` . The processes will be done by **Calibre**.

Ok, now let's push recent changes into upstream.

```
git add .
git commit -m "update"
git push origin master
```



After the process complete, the ebooks will be available for download in these following URLs. Please adjust it to follow your Github profile and repo name.

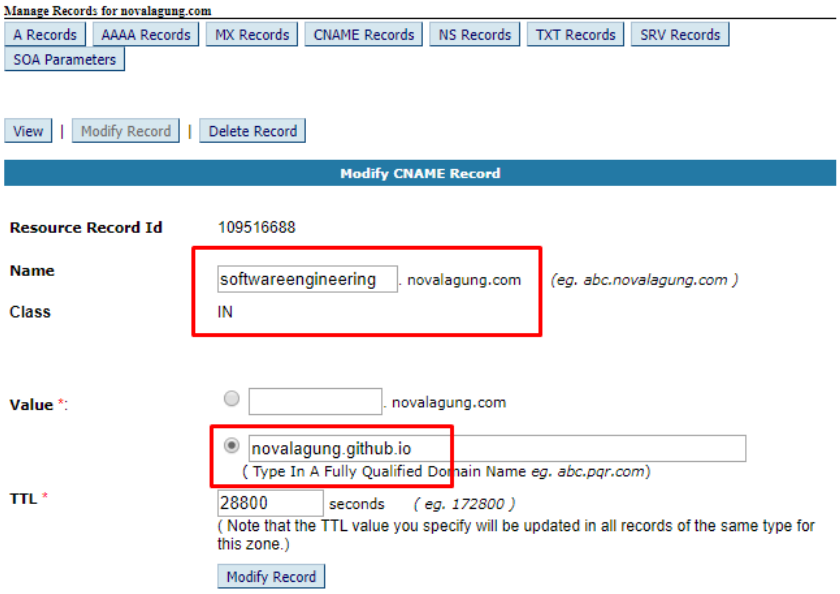
```
https://github.com/novalagung/softwareengineering/raw/ebooks/softwareengineerin
https://github.com/novalagung/softwareengineering/raw/ebooks/softwareengineerin
https://github.com/novalagung/softwareengineering/raw/ebooks/softwareengineerin
```

FYI! Since the ebook files are accessible through Github direct link, this means the visibility of the repo needs to be public (not private). If you want the repo to be in private but keep the files accessible, then do push the files into `gh-pages` branch.

2.7. Add custom domain

This one is optional, but probably important. We are going to apply custom domain to our Github Page.

Let's do it. Navigate to your domain control panel, add a new **CNAME** record that points to your Github page domain `<github-username>.github.io` .



FYI, In this example, we pick subdomain `softwareengineering.novalagung.com`. So for every incoming request to this domain, it will be directed to our Github Pages.

Next, in the Gitbook project, create a new file called `CNAME` then fill it with the domain/subdomain URL.

```
echo 'softwareengineering.novalagung.com' > CNAME
```

This `CNAME` file needs to be copied into the `_book` directory during the build process, it is because that folder is the one that will be pushed to the `gh-pages` branch.

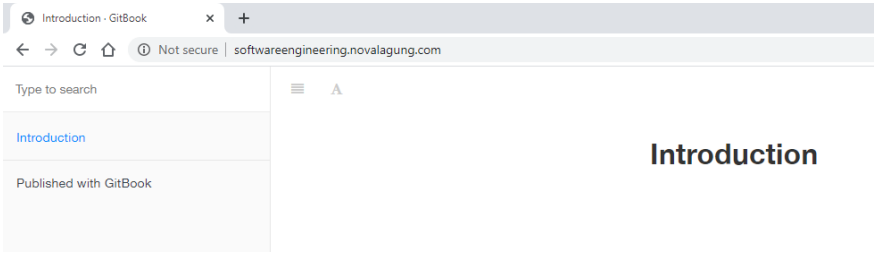
Ok, now let's put a little addition in the workflow file. In the `Generating distributable files` block, add the copy statement.

```
jobs:
  job_deploy_website:
    # ...
    - name: 'Generating distributable files'
      run: |
        gitbook install
        gitbook build
        cp ./CNAME _book/CNAME
```

Now push the update into upstream.

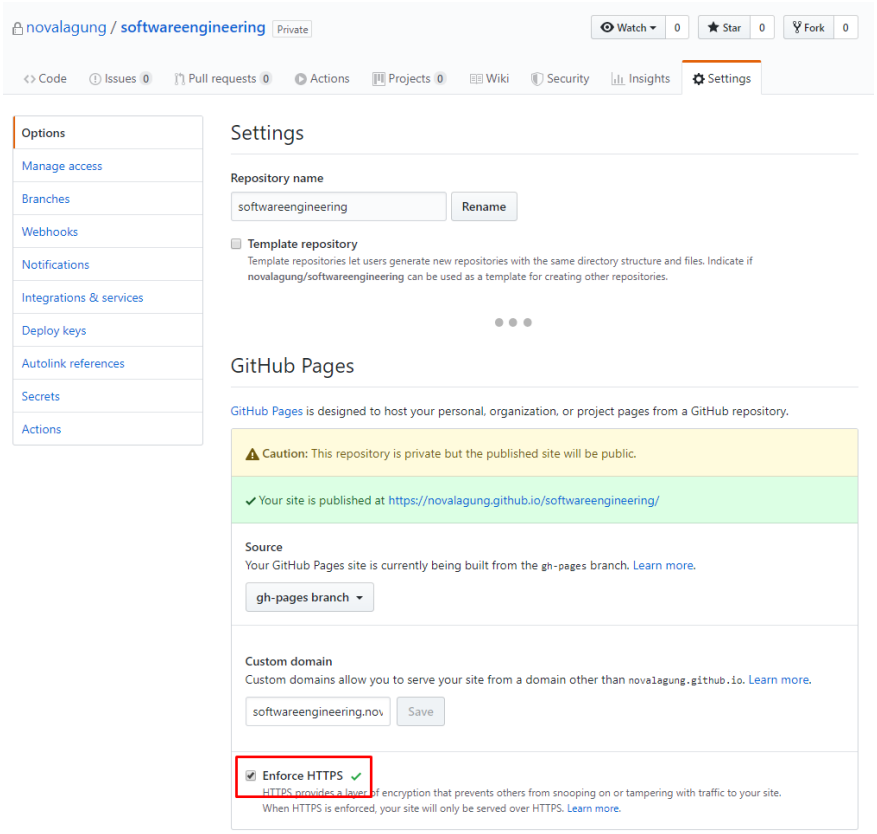
```
git add .
git commit -m "update"
git push origin master
```

Watch the workflow progress in the repo **Actions** menu. After it is finished, try to test the custom domain.



2.8. Enable `SSL/HTTPS` to our Github Pages

Lastly, before we end this tutorial, let's enable `SSL/HTTPS` into our page. No need to generate a SSL certificate file and etc, since Github will handle the setup. We just need to navigate to the **Settings** menu on the the repo, then scroll down a little bit until **GitHub Pages** section appears. Do check the **Enforce HTTPS** option. After that, wait for a few minutes, then try the custom domain again.



Have a go! https://softwareengineering.novalagung.com

Docker - Push Image to hub.docker.com

In this post, we are going to learn about how to push a Docker image to Docker Hub.

---

1. Prerequisites

1.1. Docker engine

Ensure the Docker engine is running. If you haven't installed it, then install it first.

1.2. Docker Hub account

Prepare a Docker Hub account. If you don't have it, then follow a guide on Create Docker Hub Account.

1.3. Login to Docker Hub on the local machine

Do log in to Docker Hub via CLI command below:

```
docker login --username=novalagung --password=<your-password>
```

Or use the UI menu. It is available by doing a right-click on the docker menu → sign in.

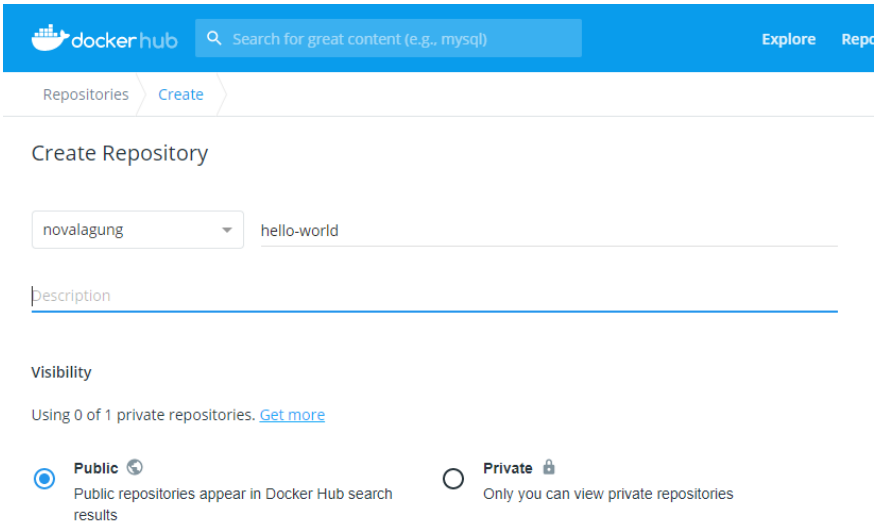
---

2. Guide

2.1. Create repo at Docker Hub

First of all, we need to book a repo on Docker Hub. Later we will push the image to that particular repo.

Go to https://hub.docker.com/repository/create, create a new repo (under your account), name it `hello-world` (or anything).



2.2. Clone the example app then build as Docker image

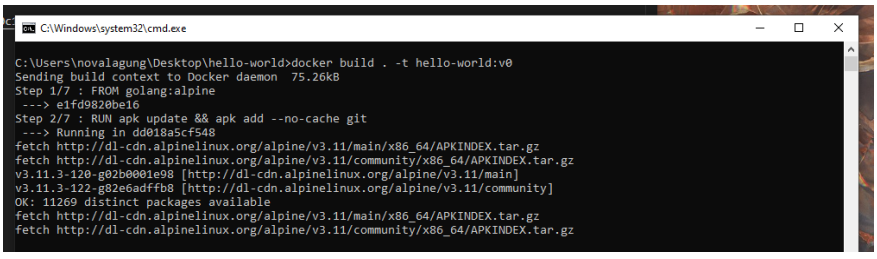
Next, we need to create a simple dockerized hello world app. But to make the thing faster, we will use a ready-to-deploy-dockerized hello world app crafted using Go. It's available on Github (via Github token), just run the command below.

```
git clone https://30542dd8874ba3745c55203a091c345340c18b7a:x-oauth-basic@github
```

After the cloning process is finished, build the app as Docker image with a name in this format `<your-docker-username>/<your-repo-name>:<tag-name>`. Adjust the value of `<your-docker-username>` to use your actual Docker Hub username.

```
cd hello-world

# docker build . -t <username>/<repo-name>:<tag>
docker build . -t novalagung/hello-world:v0
```

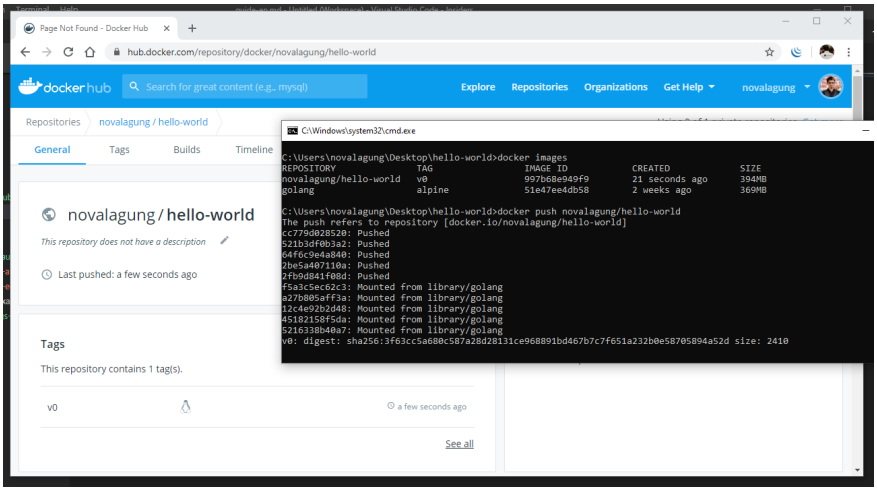
As we can see from the command above, the tag `v0` is used on this image.



2.3. Push image into Docker Hub

Next, use `docker push` command below to push the image that we just built.

```
# docker push <username>/<repo-name>[:<tag>]
docker push novalagung/hello-world
```



Ok, done.

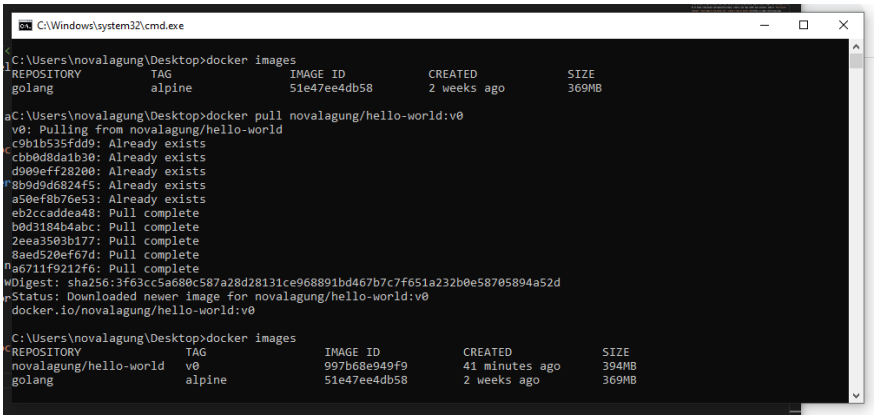
---

3. Test - Pull the Image from Docker Hub

> This step is optional.

We have pushed the image into Docker Hub. To pull it, use the `docker pull` command.

```
# docker pull <username>/<repo-name>[:<tag>]
docker pull novalagung/hello-world:v0
```



---

4. The `latest` tag

By default, when we pull a certain image from the Hub without a tag specified, then the `latest` tag of the particular image will be pulled.

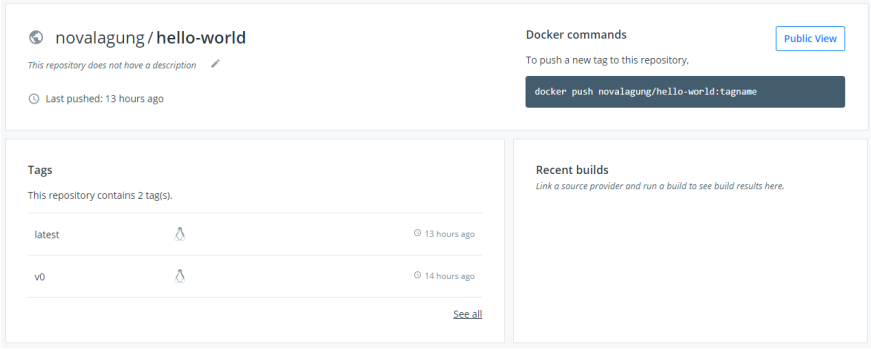
Take a look at two commands below, they are equivalent.

```
docker pull novalagung/hello-world
docker pull novalagung/hello-world:latest
```

The funny thing about this what-so-called `latest` tag is, it is actually not referring to the latest tag pushed to the Hub, it'll look for a tag with explicit name `latest`.

The previous `v0` tag won't be treated as the latest tag. To have the latest tag, we shall rebuild our project into another image then push it to the Hub, but this time during the build we will do it using `latest` as the tag.

```
cd hello-world
docker build . -t novalagung/hello-world:latest
docker push novalagung/hello-world:latest
```



Kubernetes - Deploy App into Minikube Cluster using Deployment controller, Service, and Horizontal Pod Autoscaler

In this post, we are going to learn about how to deploy a containerized app into the Kubernetes (minikube) cluster, enable the horizontal autoscaling on it, and create a service that makes the application accessible from outside the cluster.

The application that we are going to use on the tutorial is a simple hello world app written in Go. The app is dockerized and the image is available on Docker Hub.

You can also deploy your app, just do push it into Docker Hub. This guide might help you Push Image to hub.docker.com.

---

1. Prerequisites

1.1. Docker engine

Ensure the Docker engine is running. If you haven't installed it, then do install it first.

1.2. Minikube

Ensure the Minikube is running. Run `minikube start` command on PowerShell (opened with an administrator privilege). If you haven't installed it, then do install it first.

1.3. Kubernetes CLI tool

Ensure the `kubectl` command is available. If you haven't installed it, then do install it first.

1.4. The `hey` tool (an HTTP load generator)

Install this tool in your local machine https://github.com/rakyll/hey. It's similar to the Apache Benchmark tool. We are going to use this to perform the stress test to our app to check whether the auto-scaling capability is working or not.

---

2. Preparation

2.1. For Windows user only, run PowerShell with admin privilege

CMD won't be helpful here. Run the PowerShell as an administrator.

2.2. Create the Kubernetes objects configuration file (in `.yaml` format)

We are going to create three Kubernetes objects: the deployment, horizontal pod auto scaler, and service. But to make things easier, we will do the creation by using the config file.

So the three objects mentioned above will be defined in a `.yaml` file. One object usually represented by one config file, however, in this tutorial, we will write all configs in a single file.

Now create a file called `k8s.yaml` (or use another name, it is fine). Open the file using your favorite editor. Next, we shall begin the config definition.

---

3. Object Definitions

3.1. Deployment Object

Deployment is a controller used to manage both pod and replica sets. In this section, we are going to create the object.

On the `k8s.yaml`, write the following config below. Each part of the script has some remark that explains what it does.

```
---
# there is a lot of APIs available in Kubernetes (try `kubectl api-versions` to
# for this block of deployment code, we will use `apps/v1`.
apiVersion: apps/v1

# book this block of YAML for Deployment.
kind: Deployment

# name it `my-app-deployment`.
metadata:
  name: my-app-deployment

# specification of the desired behavior of the Deployment.
spec:

  # selector.matchLabels basically used to determine which pods are managed by
  # this deployment will manage all pods that have labels matching the selecto
  selector:
    matchLabels:
      app: my-app

  # template describes the pods that will be created.
  template:

    # put a label on the pods as `my-app`.
    metadata:
      labels:
        app: my-app

    # specification of the desired behavior of the `my-app` pod.
    spec:

      # list of containers belonging to the `my-app` pod.
      containers:

          # allocate a container, name it as `hello-world`.
        - name: hello-world

          # the container image is on docker hub repo `novalagung/hello-world`
          # if the particular image is not available locally, then it'll be pul
          image: novalagung/hello-world

          # set the env vars during container build process.
          # for more details take a look at
          # https://hub.docker.com/repository/docker/novalagung/hello-world.
          env:
            - name: PORT
              value: "8081"
```

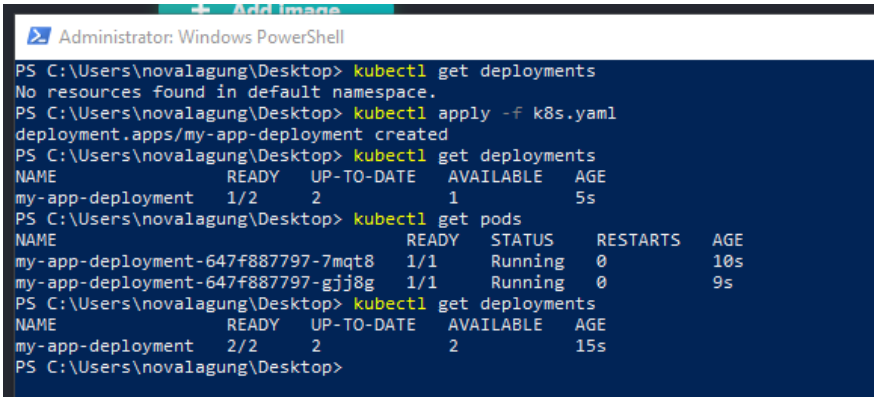
```
            - name: INSTANCE_ID
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name

          # this pod only have one container (`hello-world`),
          # and what this container does is start a webserver that listens to p
          # the port need to be exposed,
          # to make it accessible between the pods within the cluster.
          ports:
            - containerPort: 8081

          # compute resources required by this container `hello-world`.
          resources:
            limits:
              cpu: 250m
              memory: 32Mi
```

In summary, the above deployment config will do these things:

- Create a deployment object called `my-app-deployment`.
- The pod spec (within deployment object) defined with only one container.
- The container is `hello-world` and the image will be pulled from Docker Hub.
- During the container build, port and instance ID specified. The port specifically used by the webserver within the container.
- The web server listens to the port `8081` and it is exposed. Meaning we will be able to access the web server from outside the particular port but within the cluster.

Now, apply the config using the command below.

```
# apply the config
kubectl apply -f k8s.yaml

# show all deployments
kubectl get deployments

# show all pods
kubectl get pods
```



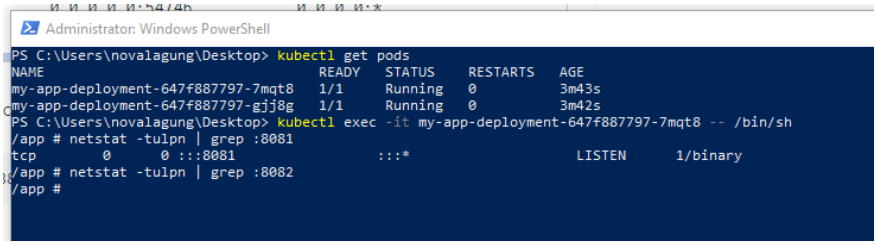
3.1. Testing one of the pod

As we can see from the image above, the deployment is working and two pods are currently running.

> Two pods automatically created. This is because we don't specify the `spec.replicas` value. If we specify some value like `4`, then there will be 4 pods running. The default replicas value is `2`.

Let's do some testing here. We will try to connect into one of the pods and then check whether the app is listening to port `8081` or not.

```
# show all pods
kubectl get pods


# connect to specific pod
kubectl exec -it <pod-name> -- /bin/sh


# check for app that listen to port 8081
netstat -tulpn | grep :8081
```



It's clear from the image above that the app is running on port `8081`.

3.2. Apply changes on the deployment object

Other than deployment, there are some other controllers available in k8s. What makes deployment controller special is whenever there is a change happen in the pod config within deployment resource, when we apply it then the pods will be updated by the controller seamlessly.

Ok, now let's prove the above statement by doing some changes on the deployment config. Do the following changes:

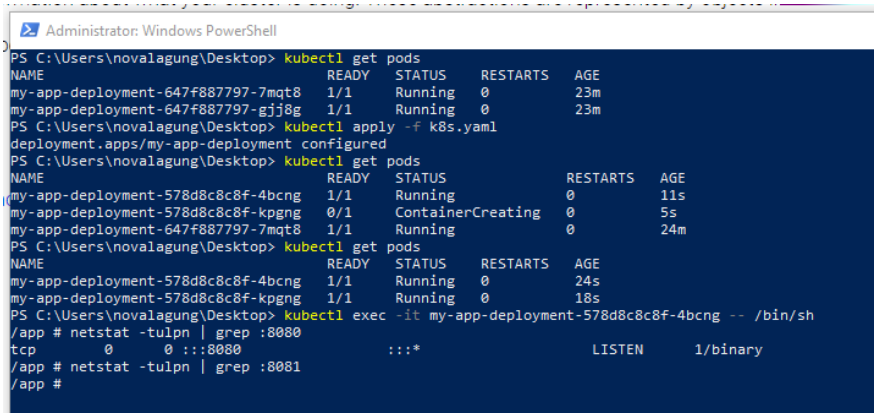
- Change `containers.env.value` of `PORT` env to `8080`. Previously it is `8081`.
- Change `containers.ports.containerPort` to `8080`. Previously it is `8081`.

Below is how the config will look like after the changes.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-deployment
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: hello-world
          image: novalagung/hello-world
          env:
            - name: PORT
              value: "8080" # <--- change from 8081 to 8080
            - name: INSTANCE_ID
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
          ports:
            - containerPort: 8080 # <--- change from 8081 to 8080
          resources:
            limits:
              cpu: 250m
              memory: 32Mi
```

Next, re-apply this config.

```
# apply the config
kubectl apply -f k8s.yaml


# show all pods
kubectl get pods


# connect to specific pod
kubectl exec -it <pod-name> -- /bin/sh


# check for the app that listens to port 8080
netstat -tulpn | grep :8080
```



See, the changes that we made on the pod are applied in a controlled way. And the web server within the newly created pod is listening to port `8080`. This is nice!

> Tips! Use the command below to see the error log on certain pods. Probably useful is something wrong going on, like the webserver not starting, etc.
>
> `kubectl get pods`
>
> `kubectl describe pod <pod-name>`
>
> `kubectl logs <pod-name>`

3.2. Service Object

In this section, we are going to create a new service. This service shall enable access between pod within the cluster, and also enable access for incoming request from external into the cluster pod.

> The `NodePort` service type can be used in our situation as well, not just `LoadBalancer` type

Let's append below config into the `k8s.yaml` file.

```
---
# pick API version `v1` for service.
apiVersion: v1

# book this block of YAML for Service.
kind: Service

# name it `my-service`.
metadata:
  name: my-service

# spec of the desired behavior of service.
spec:

  # pick LoadBalancer as the type of the service.
  # a LoadBalancer service is the standard way to expose a service to the inte
  # this will spin up a Network Load Balancer that will give you a single IP ad
  # that will forward all traffic to your service.
  #
  # on cloud provider this will generate an external IP for public access.
  # in local usage (e.g. minikube), the service will be accessible through mini
  type: LoadBalancer

  # route service traffic to pods with label keys and values matching this sele
  selector:
    app: my-app

  # the list of ports that are exposed by this service.
  ports:

      # expose the service to outside of cluster, make it publicily accessible
      # via external IP or via cluster public IP (e.g minikube IP) using nodePo
      #
      # to get the exposed URL (with IP): `minikube service my-service --url`.
      #   => http://<cluster-public-ip>:<nodePort>
    - nodePort: 32199

      # the incoming external request into nodePort will be directed towards po
      # this particular service, within the cluster.
      #
      # to get the exposed URL (with IP): `kubectl describe service my-service
      #   => http://<service-ip>:<port>
      port: 80

      # then from the service, it'll be directed to the available pods
      # (in round-robin style), to pod IP with port 8080.
      #   => http://<pod-ip>:<targetPort>
      targetPort: 8080
```

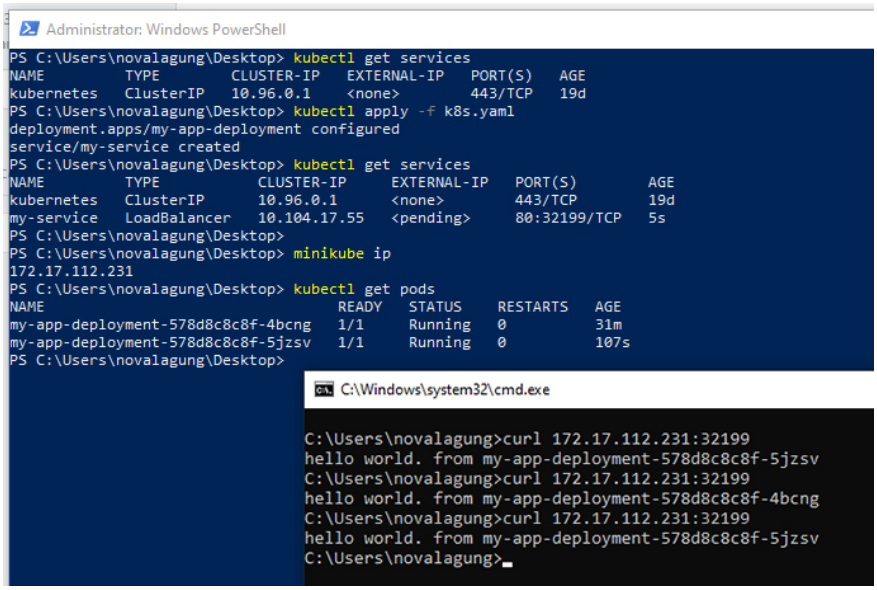
The `LoadBalancer` is chosen as the type of the service. Load balancer service will accept requests from `<publicIP>:<nodePort>` and direct it to port `80` in the service. And then the request on the port `80` will be directed to the `<pod>:<targetPort>` in round-robin style (since it's load balancer after all).

One important note here, since our cluster is within the Minikube environment, so the public IP here refers to the public IP of Minikube. To get the Minikube IP, use command below:

```
# show minikube public IP
minikube ip
```

Ok, let's apply our new `k8s.yaml` file and test the service.

```
# apply the config
kubectl apply -f k8s.yaml

# show all services
kubectl get services

# show all pods
kubectl get pods

# test app using curl
curl <minikubeIP>:<nodePort>
curl <minikubeIP>:32199
```



As we can see from the image above, we did dispatch multiple HTTP requests to the service. The result from the `curl` is different from one another, this is because the service will direct incoming request into available pods in round-robin

style (like what load balancer usually do).

> Tips! Rather than find the Service URL using `minikube ip` and then concat it with node port from config, use command below:
>
> ```
> minikube service <service-name> --url
> minikube service my-service --url
> ```

3.3. Horizontal Pod Auto Scaler (HPA) Object

In this section, we are going to make our pods (within deployment object) scalable in an automated manner. So in case, there is a spike in the total number of users that currently accessing the app, then we shall not be worried.

One way to make the pod scaled automatically is by adding HPA or Horizontal Pod Auto Scaler. The Horizontal Pod Autoscaler automatically scales the number of pods in a replication controller, deployment, replica set or stateful set based on observed CPU utilization (or, with custom metrics support, on some other application-provided metrics).

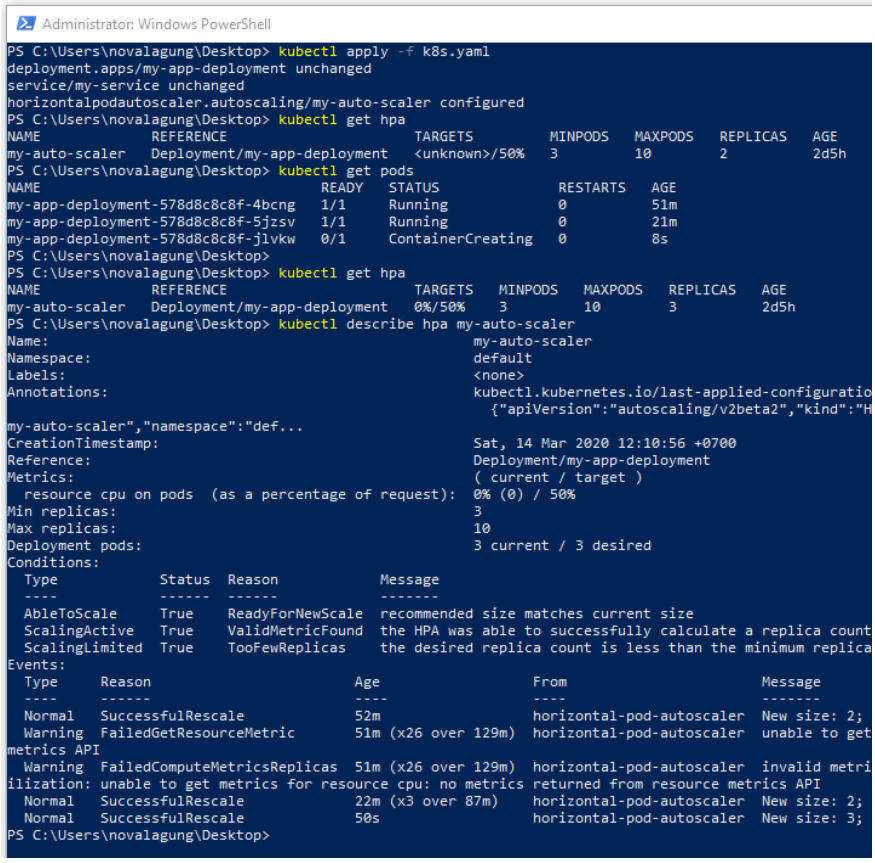
Do append below configuration into `k8s.yaml` file.

```
---
# pick API version `autoscaling/v2beta2` for auto scaler.
apiVersion: autoscaling/v2beta2

# book this block of yaml for HPA (HorizontalPodAutoscaler).
kind: HorizontalPodAutoscaler

# name it `my-auto-scaler`.
metadata:
  name: my-auto-scaler

# spec of the desired behavior of the auto scaler.
spec:

  # min replica allowed.
  minReplicas: 3

  # max replica allowed.
  maxReplicas: 10

  # the deployment that will be scalled is `my-app-deployment`.
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app-deployment

  # metrics contains the specifications for which to use to calculate the desi
  # replica count (the maximum replica count across all metrics).
  # the desired replica count is calculated multiplying the ratio between the
  # target value and the current value by the current number of pods.
  metrics:

      # resource refers to a resource metric known to Kubernetes describing ea
      # in the current scale target (e.g. CPU or memory).
      # in below we define the scaling criteria as, if CPU utilization is chan
      # the amount of 50% utilization, then scalling process shall happen.
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

The remarks on each part of the config above are quite clear. In summary, an HPA will be created attached to `my-app-deployment`, numbers on the replication rules are defined, with scaling criteria is focusing on CPU utilization when average utilization is between 50%.

Ok now let's re-apply our HPA.

```
# apply the config
kubectl apply -f k8s.yaml


# show all HPA
kubectl get hpa


# show describe HPA
kubectl describe hpa <hpa-name>
```



Previously we only have two pods running. After we apply the HPA, the new pod is created, so total there are three pods. This is because in our HPA the `spec.minReplicas` is set to `3`.

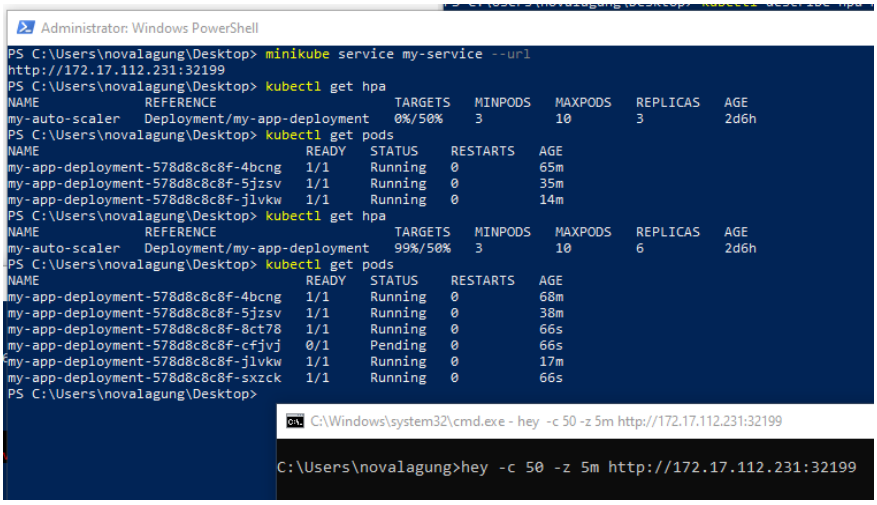
3.3.1. Stress test on Horizontal Pod Auto scaler

Ok, next let's do some stress test! Let's see how the HPA will handle very high traffic coming. The below command will trigger a concurrent 50 request to the service URL for 5 minutes. Run it on a new CMD/PowerShell window.

```
# show service URL
minikube service my-service --url


# start the stress test
hey -c 50 -z 5m <service-URL>
```

And then back to our main PowerShell window, do regularly check the pods.

```
# show all HPA and pods
kubectl get hpa
kubectl get pods
```



After a minute passed, suddenly a total of 6 pods created. This is happening because the CPU utilization is high enough, greater than the threshold that we defined in the config.

HPA is not only able to magically scale the pod during high traffic but on low traffic, the scaling process will happen as well. Do stop the stress test and wait for a few minutes, and check the HPA and pods again, you will see the number of pods reduced to `spec.minReplicas` again.

Ok, that's it.

Setup Oracle Instant Client

In this post, we are going to learn how to setup Oracle instant client on Linux, Windows, and MacOS.

Table of Contents

A. Setup Oracle Instant Client on Linux (Ubuntu 16.04)

First of all, download these three files from https://www.oracle.com/technetwork/topics/linuxx86-64soft-092277.html.

- instantclient-basic-linux.x64-12.2.0.1.0.zip
- instantclient-sdk-linux.x64-12.2.0.1.0.zip
- instantclient-sqlplus-linux.x64-12.2.0.1.0.zip

Then update os package repository, continue with install the required essentials tools/

```
sudo apt-get update
sudo apt-get install build-essential libaio1 unzip git pkg-config
```

Next, we shall setup the oracle client.

```
mkdir -p /home/novalagung/oracle && cd /home/novalagung/oracle
cp /where/instantclient-*.zip .

unzip instantclient-basic-linux.x64-12.2.0.1.0.zip
unzip instantclient-sdk-linux.x64-12.2.0.1.0.zip
unzip instantclient-sqlplus-linux.x64-12.2.0.1.0.zip
rm -rf instantclient-*.zip

echo 'export ORACLE_HOME=/home/novalagung/oracle/instantclient_12_2' >> /home/n
echo 'export PATH=$PATH:$ORACLE_HOME' >> /home/novalagung/.bashrc
source /home/novalagung/.bashrc

cd /home/novalagung/oracle/instantclient_12_2
ln -s libclntsh.so.12.1 libclntsh.so
ln -s libocci.so.12.1 libocci.so

sudo sh -c 'echo '/home/novalagung/oracle/instantclient_12_2' >> /etc/ld.so.con

echo 'export DYLD_LIBRARY_PATH=$ORACLE_HOME' >> /home/novalagung/.bashrc
echo 'export LD_LIBRARY_PATH=$ORACLE_HOME' >> /home/novalagung/.bashrc

sudo ldconfig
```

Next, create `oci8.pc` file. This file is required later by go oracle driver to be able to communicate with the oracle database server. If you plan only to connect to the oracle database server by using `sqlplus` only, then the file is not necessarily required.

```
sudo nano /usr/lib/pkgconfig/oci8.pc
```

Fill it with this content:

```
instantclient=/home/novalagung/oracle/instantclient_12_2
libdir=${instantclient}
includedir=${instantclient}/sdk/include/

Name: oci8
Description: oci8 library
Version: 12.1
Libs: -L${libdir} -lclntsh
Cflags: -I${includedir}
```

Last, try to connect to the oracle db server using `sqlplus`.

```
sqlplus SYSTEM/MANAGER@//localhost:1521/XE

# SQL*Plus: Release 11.2.0.4.0 Production on Wed Oct 3 06:48:54 2018
# Copyright (c) 1982, 2013, Oracle.  All rights reserved.
#
# Connected to:
# Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
```

The result is: **connected**. Try to perform a simple query like getting the database version.

```
SQL> SELECT * FROM V$VERSION;

BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
PL/SQL Release 11.2.0.2.0 - Production
CORE    11.2.0.2.0    Production
TNS for Linux: Version 11.2.0.2.0 - Production
NLSRTL Version 11.2.0.2.0 - Production
```

B. Setup Oracle Instant Client on Windows 10

First of all, download these three files from https://www.oracle.com/technetwork/topics/winx64soft-089540.html.

- instantclient-basic-windows.x64-12.2.0.1.0.zip
- instantclient-sdk-windows.x64-12.2.0.1.0.zip
- instantclient-sqlplus-windows.x64-12.2.0.1.0.zip

Create `Oracle` folder at `C:\Oracle`, put all downloaded archives into this folder.

```
cd \
mkdir Oracle
```

Then extract zip files, all of them. By default it'll be extracted into `instantclient_12_2` folder under `C:\Oracle`, and let it be.

Append the `C:\Oracle\instantclient_12_2` path into `%PATH%` variable.

Next, set these CGO_ variables.

```
setx CGO_CFLAGS "-IC:\OtherPrograms\Oracle\instantclient_12_2\sdk\include"
setx CGO_LDFLAGS "-LC:\OtherPrograms\Oracle\instantclient_12_2 -loci"
```

Now we need to install **GCC**, and in this tutorial we'll use MSYS2 64bit. Download the installer `msys2-x86_64-*.exe` from https://www.msys2.org. After download process finished, run the installer. Pick any directory path you want, but make sure to remember it. In my place, I install it here.

```
C:\msys64
```

Next, run the **MSYS2 MinGW 64-bit** application. Then execute these commands.

```
# Update pacman
pacman -Su
# Install pkg-config and gcc
pacman -S mingw64/mingw-w64-x86_64-pkg-config mingw64/mingw-w64-x86_64-gcc
```

Now, set the `PKG_CONFIG_PATH` variable to points to the oci8.pc file inside msys64 (mingw64) pkgconfig folder.

```
setx PKG_CONFIG_PATH "C:\msys64\mingw64\lib\pkgconfig\oci8.pc"
```

Then add msys64 (mingw64) binary path into `%PATH%` variable.

```
C:\msys64\mingw64\bin
```

Next create the `oci8.pc` file on inside msys64 (mingw64) pkgconfig folder. This file is required later by go oracle driver to be able to communicate with the oracle database server. If you plan only to connect to the oracle database server by using `sqlplus` only, then the file is not necessarily required.

```
C:\msys64\mingw64\lib\pkgconfig\oci8.pc
```

Below is the content of the file.

```
oralib="C:/Oracle/instantclient_12_2/sdk/lib/msvc"
orainclude="C:/Oracle/instantclient_12_2/sdk/include"
gcclib="C:/msys64/mingw64/lib"
gccinclude="C:/msys64/mingw64/include"

Name: oci8
Version: 12.2
Description: oci8 library
Libs: -L${oralib} -L${gcclib} -loci
Libs.private:
Cflags: -I${orainclude} -I${gccinclude}
```

REMINDER: You need to adjust the `oralib`, `orainclude`, `gcclib`, and `gccinclude` value to match your settings. And also replace the backslash (`\`) into slash (`/`).

OK, the oracle client setup is done. Last step, try to connect to the oracle db server using `sqlplus`.

```
sqlplus SYSTEM/MANAGER@//localhost:1521/XE


# SQL*Plus: Release 11.2.0.4.0 Production on Wed Oct 3 06:48:54 2018
# Copyright (c) 1982, 2013, Oracle.  All rights reserved.
#
# Connected to:
# Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
```

The result is: **connected**. Try to perform a simple query like getting the database version.

```
SQL> SELECT * FROM V$VERSION;


BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
PL/SQL Release 11.2.0.2.0 - Production
CORE    11.2.0.2.0    Production
TNS for Linux: Version 11.2.0.2.0 - Production
NLSRTL Version 11.2.0.2.0 - Production
```

Setup Oracle XE Database Server

In this post, we are going to learn how to setup Oracle XE Database Server on CentOS, Oracle Linux, and using Docker Container.

Table of Contents

A. Setup Oracle XE Database Server on CentOS 6 (Oracle Linux)

A.1. Convert CentOS 6 into Oracle Linux

The easiest way to install Oracle Database Server is through **Oracle Linux** distribution.

Oracle Linux is a Linux distribution packaged and freely distributed by Oracle, available partially under the GNU General Public License since late 2006. It's free, we can easily get it from Oracle Linux Download Page.

There is also an alternative way to get the Oracle Linux, by converting CentOS into Oracle Linux. In this post we'll learn to do that.

OK let's start. First of all, update os package repository.

```
sudo yum update
```

Oracle provides us capability to convert CentOS into Oracle Linux, and they make it to be so easy to use. For detailed information just take a look at https://linux.oracle.com/switch/centos.

Ok, let's download the `centos2ol.sh` file then execute it.

```
curl -O https://linux.oracle.com/switch/centos2ol.sh
sudo sh centos2ol.sh

# Checking for required packages...
# Checking your distribution...
# Checking for yum lock...
# Looking for yumdownloader...
# Finding your repository directory...
# Downloading Oracle Linux yum repository file...
# Backing up and removing old repository files...
# Downloading Oracle Linux release package...
#
# ... will take sometime
#
# Dependency Updated:
#   plymouth-core-libs.x86_64 0:0.8.3-29.0.1.el6
#
# Replaced:
#   redhat-logos.noarch 0:60.0.14-12.el6.centos
#
# Finished Transaction
# > Leaving Shell
# Updating initrd...
# Installation successful!
# Run 'yum distro-sync' to synchronize your installed packages
# with the Oracle Linux repository.
```

Next, synchronize the installed packages to the Oracle Linux repository by using command below.

```
sudo yum distro-sync

# Loaded plugins: fastestmirror, security
# Setting up Distribution Synchronization Process
# Loading mirror speeds from cached hostfile
# Only Upgrade available on package: sysstat-9.0.4-33.el6_9.1.x86_64
# Resolving Dependencies
# --> Running transaction check
# ---> Package acpid.x86_64 0:1.0.10-3.el6 will be updated
# ---> Package acpid.x86_64 0:2.0.19-6.0.1.el6 will be an update
#
# ...
#
# Updated:
#   sos.noarch 0:3.2-63.0.1.el6_10.2
#   system-config-network-tui.noarch 0:1.6.0.el6.3-4.0.1.el6
#   systemtap-runtime.x86_64 0:2.9-9.0.1.el6
#   yum-plugin-security.noarch 0:1.1.30-42.0.1.el6_10
#   yum-utils.noarch 0:1.1.30-42.0.1.el6_10
#
# Complete!
```

Just that, your Oracle Linux is ready.

A.2. Setup Oracle XE Database Server on Oracle Linux

You can get Oracle linux from Oracle Linux download page, or by converting CentOS into Oracle Linux.

Download the **Oracle Database Express Edition 11g Release 2 for Linux x64** from https://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/xe-prior-releases-5172097.html. You might need to download it from the web browser since the download process require us to log in using oracle account (create one on the website if you haven't).

Unzip the downloaded oracle xe installer.

```
mkdir -p /home/novalagung/oracle-xe
cd /home/novalagung/oracle-xe
cp /path/to/file/oracle-xe-11.2.0-1.0.x86_64.rpm.zip .
unzip oracle-xe-11.2.0-1.0.x86_64.rpm.zip

sudo rpm -ivh Disk1/oracle-xe-11.2.0-1.0.x86_64.rpm
```

Now the Oracle XE 11g is installed. Next we need to run the Oracle XE Configuration. In this step few prompts will appear asking certain information like port for Oracle Application Express and for database listener.

One default user will be created, it's `SYSTEM` user. We'll need to put some password for this user (cannot left it empty). In this example we use `MANAGER` as the password.

```
sudo /etc/init.d/oracle-xe configure

# Oracle Database 11g Express Edition Configuration
# -------------------------------------------------
# This will configure on-boot properties of Oracle Database 11g Express
# Edition.  The following questions will determine whether the database should
# be starting upon system boot, the ports it will use, and the passwords that
# will be used for database accounts.  Press <Enter> to accept the defaults.
# Ctrl-C will abort.
#
# Specify the HTTP port that will be used for Oracle Application Express [8080]
#
# Specify a port that will be used for the database listener [1521]:
#
# Specify a password to be used for database accounts.  Note that the same
# password will be used for SYS and SYSTEM.  Oracle recommends the use of
# different passwords for each database account.  This can be done after
# initial configuration: MANAGER
# Confirm the password: MANAGER
#
# Do you want Oracle Database 11g Express Edition to be started on boot (y/n) [
#
# Starting Oracle Net Listener...Done
# Configuring database...Done
# Starting Oracle Database 11g Express Edition instance...Done
# Installation completed successfully.
```

Ok, now our Oracle XE is 100% ready. Next, we need to perform some tests, to make sure everything is working fine. We'll try to connect to the database server using default user `SYSTEM` and password `MANAGER`.

```
sqlplus SYSTEM/MANAGER@//localhost:1521/xe

# SQL*Plus: Release 11.2.0.4.0 Production on Wed Oct 3 06:48:54 2018
# Copyright (c) 1982, 2013, Oracle.  All rights reserved.
#
# Connected to:
# Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
```

The result is: **connected**. Try to perform a simple query like getting the database version.

```
SQL> SELECT * FROM V$VERSION;


BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
PL/SQL Release 11.2.0.2.0 - Production
CORE    11.2.0.2.0    Production
TNS for Linux: Version 11.2.0.2.0 - Production
NLSRTL Version 11.2.0.2.0 - Production
```

B. Setup Oracle XE Database Server using Docker

> This tutorial can be implemented in both Windows, Linux, or MacOS operating systems.

Download the **Oracle Database Express Edition 11g Release 2 for Linux x64** from https://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/xe-prior-releases-5172097.html. You might need to download it from the web browser since the download process require us to log in using oracle account (create one on the website if you haven't).

> REMINDER: Even you perform this installation on Windows or MacOS, you must download the Linux x64 installer! not the windows version or the macos version.

Then clone the official oracle docker images from their github.

```
git clone https://github.com/oracle/docker-images.git
```

Move the downloaded oracle xe installer into this path.

```
cd docker-images/OracleDatabase/SingleInstance/dockerfiles
cp oracle-xe-11.2.0-1.0.x86_64.rpm.zip /11.2.0.2/
```

Next, execute the `./buildDockerImage.sh` command with several arguments:

- Flag `-v 11.2.0.2` to specify the oracle version (in this case it's 11.2.0.2). The choosen version must match with the installer version.
- Flag `-x` to pick the **Express Edition** image.
- Flag `-i` to skip the md5sum verification.

```
./buildDockerImage.sh -v 11.2.0.2 -x -i

# Ignored MD5 checksum.
# ==========================
# DOCKER info:
# Containers: 3
#  Running: 0
#  Paused: 0
#  Stopped: 3
# Images: 10
# Server Version: 18.09.0
# ...
#
# ==========================
# Building image 'oracle/database:11.2.0.2-xe' ...
# Sending build context to Docker daemon  631.8MB
# Step 1/10 : FROM oraclelinux:7-slim
# 7-slim: Pulling from library/oraclelinux
# a8d84c1f755a: Pulling fs layer
# a8d84c1f755a: Verifying Checksum
# a8d84c1f755a: Download complete
# ...
#
# Removing intermediate container 51a3bdde4e7e
#  ---> bf56ef57fe4c
# Step 9/10 : HEALTHCHECK --interval=1m --start-period=5m    CMD "$ORACLE_BASE/
#  ---> Running in dcee11bca78e
# Removing intermediate container dcee11bca78e
#  ---> 4fbcb8aec67f
# Step 10/10 : CMD exec $ORACLE_BASE/$RUN_FILE
#  ---> Running in 253bd5706098
# Removing intermediate container 253bd5706098
#  ---> 97fb5f2328d0
# [Warning] One or more build-args [DB_EDITION] were not consumed
# Successfully built 97fb5f2328d0
# Successfully tagged oracle/database:11.2.0.2-xe
# SECURITY WARNING: You are building a Docker image from Windows against a non-
#
#   Oracle Database Docker Image for 'xe' version 11.2.0.2 is ready to be exte
#
#     --> oracle/database:11.2.0.2-xe
#
#   Build completed in 303 seconds.
```

The process will take some time. In the end a new docker image called `oracle/database` will be created.

Next, start a new container using the `oracle/database` image.

```
docker run --name my-oracle-db-server \
    -p 1521:1521 \
    -p 5500:5500 \
    -e ORACLE_SID=xe \
    -e ORACLE_PWD=MANAGER \
    -v oradata:/opt/oracle/oradata \
    --shm-size=2g \
    oracle/database:11.2.0.2-xe

# ORACLE PASSWORD FOR SYS AND SYSTEM: MANAGER
#
# Oracle Database 11g Express Edition Configuration
# -------------------------------------------------
# This will configure on-boot properties of Oracle Database 11g Express
# Edition.  The following questions will determine whether the database should
# be starting upon system boot, the ports it will use, and the passwords that
# will be used for database accounts.  Press <Enter> to accept the defaults.
# Ctrl-C will abort.
#
# Specify the HTTP port that will be used for Oracle Application Express [8080]
# Specify a port that will be used for the database listener [1521]:
# Specify a password to be used for database accounts.  Note that the same
# password will be used for SYS and SYSTEM.  Oracle recommends the use of
# different passwords for each database account.  This can be done after
# initial configuration:
# Confirm the password:
#
# Do you want Oracle Database 11g Express Edition to be started on boot (y/n) [
# Starting Oracle Net Listener...Done
# Configuring database...
#
# ...
#
# ##########################
# DATABASE IS READY TO USE!
# ##########################
# The following output is now a tail of the alert.log:
# QMNC started with pid=24, OS id=685
# Completed: ALTER DATABASE OPEN
# Fri Feb 22 08:17:28 2019
# db_recovery_file_dest_size of 10240 MB is 0.98% used. This is a
# user-specified limit on the amount of space that will be used by this
# database for recovery-related files, and does not reflect the amount of
# space available in the underlying filesystem or ASM diskgroup.
# Starting background process CJQ0
# Fri Feb 22 08:17:28 2019
# CJQ0 started with pid=25, OS id=699
```

Few explanations about above command arguments:

- Flag `-p 1521:1521`, export the oracle listener port.
- Flag `-p 5500:5500`, export the oem express port.
- Flag `-e ORACLE_SID=xe`, specify the oracle SID.
- Flag `-e ORACLE_PWD=MANAGER`, set the default password of `SYS`, `SYSTEM` and `PDB_ADMIN` users.
- Flag `-v oradata:/opt/oracle/oradata`, mirror the volume.
- Flag `--shm-size=2g`, allocate memory size for particular container.

Ok, now lets try to connect to the database server using default user `SYSTEM`.

```
sqlplus SYSTEM/MANAGER@//localhost:1521/XE

# SQL*Plus: Release 11.2.0.4.0 Production on Wed Oct 3 06:48:54 2018
# Copyright (c) 1982, 2013, Oracle.  All rights reserved.
#
# Connected to:
# Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
```

The result is: **connected**. Try to perform a simple query like getting the database version.

```
SQL> SELECT * FROM V$VERSION;

BANNER
--------------------------------------------------------------------------------

Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
PL/SQL Release 11.2.0.2.0 - Production
CORE    11.2.0.2.0    Production
TNS for Linux: Version 11.2.0.2.0 - Production
NLSRTL Version 11.2.0.2.0 - Production
```

If you the container stopped, then you just need to start it. No need to create new container using same specification.

Terraform - Automate setup of AWS EC2 with Internet Gateway and SSH Access enabled

In this post, we are going to learn about the usage of Terraform to automate the setup of AWS EC2 instance with internet gateway and ssh access enabled.

---

1. Prerequisites

1.1. Terraform CLI

Ensure terraform CLI is available. If not, then do install it first.

1.2. Individual AWS IAM user

Prepare a new individual IAM user with programmatic access key enabled and have access to EC2 management. We will use the `access_key` and `secret_key` on this tutorial. If you haven't created the IAM user, then follow the guide on Create Individual IAM User.

1.3. `ssh-keygen` and `ssh` commands

Ensure both `ssh-keygen` and `ssh` command are available.

---

2. Preparation

Create a new folder contains a file named `infrastructure.tf`. We will use the file as the infrastructure code. Every resource setup will be written in HCL language inside the file, including:

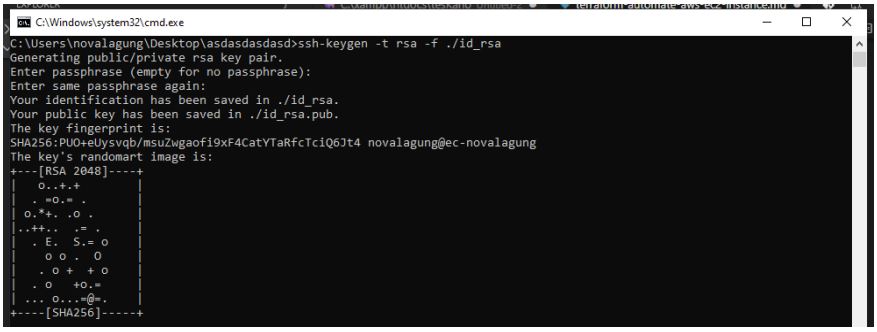
- Uploading key pair (for ssh access to the instance).
- Creating EC2 instance.
- Adding security group to VPC (where the instance will be created).
- Creating a public subnet.
- Creating an internet gateway and associate it to the subnet.

Ok, let's back to the tutorial. Now create the infrastructure file.

```
mkdir terraform-automate-aws-ec2-instance
cd terraform-automate-aws-ec2-instance
touch infrastructure.tf
```

Next, create a new public-key cryptography using `ssh-keygen` command below. This will generate the `id_rsa.pub` public key and `id_rsa` private key. Later we will upload the public key into AWS and use the private key to perform `ssh` access into the newly created EC2 instance.

```
cd terraform-automate-aws-ec2-instance

ssh-keygen -t rsa -f ./id_rsa
```



3. Infrastructure Code

Now we shall start writing the infrastructure config. Open `infrastructure.tf` in any editor.

3.1. Define AWS provider

Define the provider block with AWS as chosen cloud provider. Also define these properties: `region`, `access_key`, and `secret_key`; with values derived from the created IAM user.

Write a block of code below into `infrastructure.tf`

```
provider "aws" {
    region = "ap-southeast-1"
    access_key = "AKIAWLTS5CSXP7E3YLWG"
    secret_key = "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"
}
```

3.2. Generate new key pair then upload to AWS

Define new `aws_key_pair` resource block with local name: `my_instance_key_pair`. Put the previously generated `id_rsa.pub` public key inside the block to upload it to AWS.

```
resource "aws_key_pair" "my_instance_key_pair" {
    key_name = "terraform_learning_key_1"
    public_key = file("id_rsa.pub")
}
```

3.3. Create a new EC2 instance

Define another resource block, but this one will be the `aws_instance` resource. Name the EC2 instance as `my_instance`, then specify the values of VPC, instance type, key pair, security group, subnet, and public IP within the block.

Each part of the code below is self-explanatory.

```
# create a new AWS ec2 instance.
resource "aws_instance" "my_instance" {

    # ami => Amazon Linux 2 AMI (HVM), SSD Volume Type (ami-0f02b24005e4aec36)
    ami = "ami-0f02b24005e4aec36"

    # instance type => t2.micro.
    instance_type = "t2.micro"

    # key pair: terraform_learning_key_1.
    key_name = aws_key_pair.my_instance_key_pair.key_name

    # vpc security groups: my_vpc_security_group.
    vpc_security_group_ids = [aws_security_group.my_vpc_security_group.id]

    # public subnet: my_public_subnet.
    # this subnet is used as the gateway of the internet.
    subnet_id = aws_subnet.my_public_subnet.id

    # associate one public IP address to this particular instance.
    associate_public_ip_address = true
}
```

The `key_name` property filled with a value coming from the `my_instance_key_pair` that we defined previously. Statement `aws_key_pair.my_instance_key_pair.key_name` return the `key_name` of the particular key pair, in this example it is `terraform_learning_key_1`.

For both `vpc_security_group_ids` and `subnet_id`, the values are taken from another resource block, similar to the `key_name`. However, for these two properties, we haven't defined the resource block yet.

Btw, property `vpc_security_group_ids` accept an array of string as the value, so that's why it's wrapped inside `[]`. Even it is only one security group, the value needs to be in an array format.

3.4. Allocate a VPC resource with a security group attached to it

Allocate a VPC resource block, and then define a security group resource within the VPC.

```
# allocate a VPC named my_vpc.
resource "aws_vpc" "my_vpc" {
    cidr_block = "10.0.0.0/16"
    enable_dns_hostnames = true
}

# create a security group for my_vpc.
resource "aws_security_group" "my_vpc_security_group" {

    # tag this security group to my_vpc.
    vpc_id = aws_vpc.my_vpc.id

    # define the inbound rule, allow TCP/SSH access from anywhere.
    ingress {
        from_port = 22
        to_port = 22
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }

    # define the inbound rule, allow TCP/HTTP access on port 80 from anywhere.
    ingress {
        from_port = 80
        to_port = 80
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }

    # define the outbound rule, allow all kinds of accesses from anywhere.
    egress {
        from_port = 0
        to_port = 0
        protocol = "-1"
        cidr_blocks = ["0.0.0.0/0"]
    }
}
```

Above security group is created for `my_vpc` (see `vpc_id = aws_vpc.my_vpc.id`). This particular VPC has three inbound/outbound rules:

- Allow ssh access from anywhere. Later we need to remotely connect to the instance to see whether it's properly set up or not.

- Allow incoming access through port `80`. This might be required, so we can perform any tools/dependency installations, etc.
- Allow all kinds of outgoing accesses from anywhere. By doing this we will be able to perform remote access, download, etc to anywhere from the instance.

> ingress is equivalent to inbound, and egress for outbound

3.5. Allocate new public subnet to VPC

We have defined a VPC `my_vpc` with CIDR block `10.0.0.0/16` allocated. Now we shall create a subnet (for public access) with CIDR block slightly smaller, `10.0.0.0/24`.

```
# create a subnet for my_vpc.
resource "aws_subnet" "my_public_subnet" {
    vpc_id = aws_vpc.my_vpc.id
    cidr_block = "10.0.0.0/24"
}
```

If we go back to the definition of `my_instance` block above, this particular subnet is attached there.

3.6. Create an internet gateway and route table association

Now create an internet gateway for `my_vpc`. Then attach it to a new route table for public access.

```
# create an internet gateway, tag it to my_vpc.
resource "aws_internet_gateway" "my_internet_gateway" {
    vpc_id = aws_vpc.my_vpc.id
}

# create a new route table for attaching my_internet_gateway into my_vpc.
resource "aws_route_table" "my_public_route_table" {
    vpc_id = aws_vpc.my_vpc.id
    route {
        cidr_block = "0.0.0.0/0"
        gateway_id = aws_internet_gateway.my_internet_gateway.id
    }
}
```

Associate the public route table above into `my_public_subnet`, so then we will get an internet access on `my_instance` instance.

```
# create a route table association to connect my_public_subnet with my_public_
resource "aws_route_table_association" "my_public_route_table_association" {
  subnet_id = aws_subnet.my_public_subnet.id
  route_table_id = aws_route_table.my_public_route_table.id
}
```

Pretty much everything is done, except we need to show the DNS or public IP of newly created instance, so then we can test it using ssh access. use the `output` block to print both public DNS and IP of the instance.

```
output "public-dns" {
    value = aws_instance.my_instance.*.public_dns[0]
}
output "public-ip" {
    value = aws_instance.my_instance.public_ip
}
```

The infra file is ready. Now we shall perform the terraforming process.

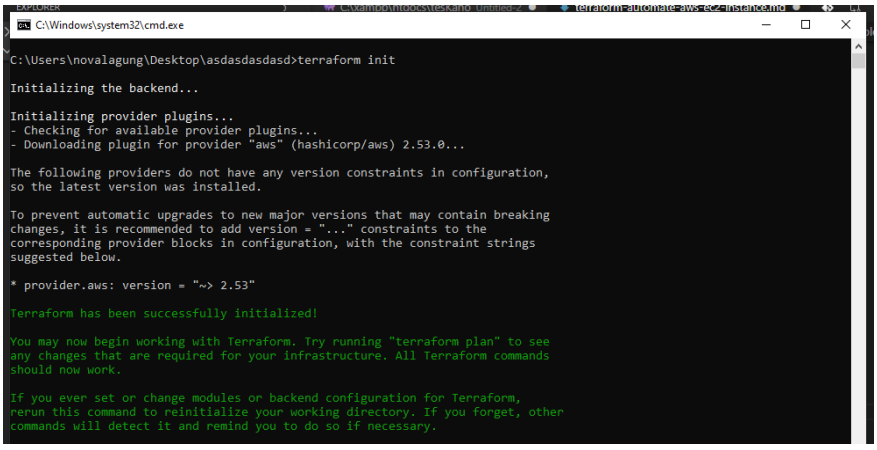
---

4. Run Terraform

4.1. Terraform initialization

First, run the `terraform init` command. This command will do some setup/initialization, certain dependencies (like AWS provider that we used) will be downloaded.

```
cd terraform-automate-aws-ec2-instance
terraform init
```



4.2. Terraform plan

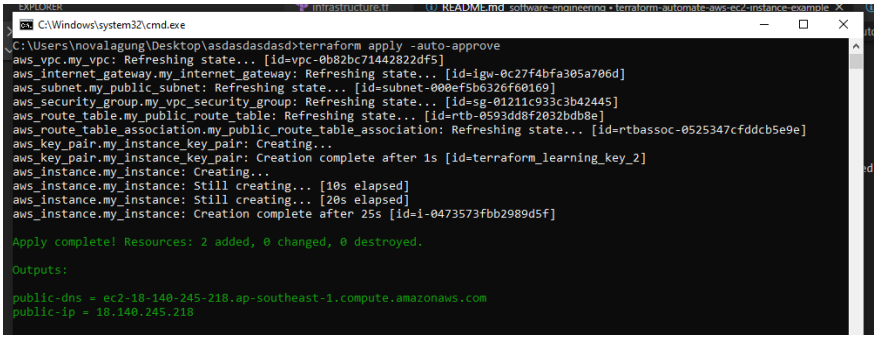
Next, run `terraform plan`, to see the plan of our infrastructure. This step is optional, however, might be useful for us to see the outcome from the infra file.

4.3. Terraform apply

Last, run the `terraform apply` command to execute the infrastructure plan.

```
cd terraform-automate-aws-ec2-instance

terraform apply -auto-approve
```

The `-auto-approve` flag is optional, it will skip the confirmation prompt during execution.

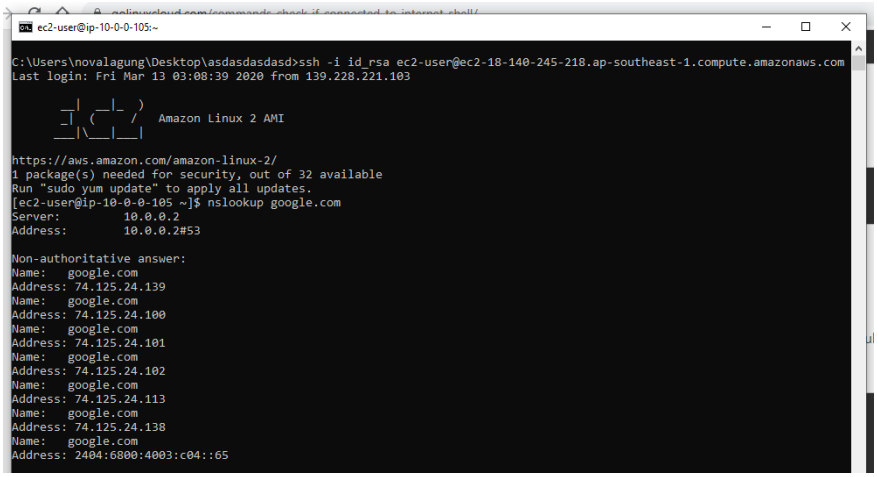


In the infra file, we defined two outputs, DNS and public IP, it shows up after the terraforming process is done.

5. Test Instance

Now we shall test the instance. Use the `ssh` command to remotely connect to a particular instance. Either DNS or public IP can be used, just pick one.

```
ssh -i id_rsa ec2-user@ec2-18-140-245-218.ap-southeast-1.compute.amazonaws.com
```



We can see from the image above that we can connect to ec2 instance via SSH, and the instance is connected to the internet.

Terraform - Automate setup of AWS EC2 with Application Load Balancer and Auto Scaling enabled

In this post, we are going to learn about the usage of Terraform to automate the setup of AWS EC2 instance in an auto-scaling environment with an Application Load Balancer applied.

Since we will be using the auto-scaling feature, then the app within the instance needs to be deployed in an automated manner.

The application is a simple go app, currently hosted on Github in a private repo. We will clone the app using Github token, we will talk about it in details in some part of this tutorial.

---

1. Prerequisites

1.1. Terraform CLI

Ensure terraform CLI is available. If not, then do install it first.

1.2. Individual AWS IAM user

Prepare a new individual IAM user with programmatic access key enabled and have access to EC2 management. We will use the `access_key` and `secret_key` on this tutorial. If you haven't created the IAM user, then follow a guide on Create Individual IAM User.

1.3. `ssh-keygen` and `ssh` commands

Ensure both `ssh-keygen` and `ssh` command are available.

---

2. Preparation

Create a new folder contains a file named `infrastructure.tf`. We will use the file as the infrastructure code. Every resource setup will be written in HCL language inside the file, including:

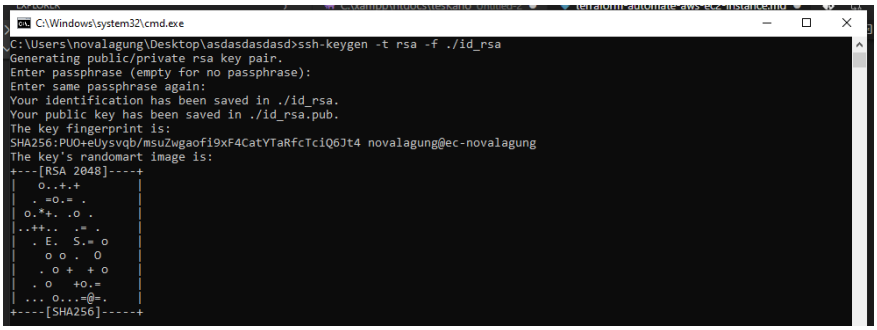
- Uploading key pair (for ssh access to the instance).
- Subnetting on two different availability zones (within the same region).
- Defining Application Load Balancer, it's listener, security group, and target group.
- Defining Auto-scaling and it's launch config.

Ok, let's back to the tutorial. Now create the infrastructure file.

```
mkdir terraform-automate-aws-ec2-instance
cd terraform-automate-aws-ec2-instance
touch infrastructure.tf
```

Next, create a public-key cryptography using `ssh-keygen` command below. This will generate the `id_rsa.pub` public key and `id_rsa` private key. Later we will upload the public key into AWS and use the private key to perform `ssh` access into the newly created EC2 instance.

```
cd terraform-automate-aws-ec2-instance
ssh-keygen -t rsa -f ./id_rsa
```



3. Infrastructure Code

Now we shall start writing the infrastructure config. Open `infrastructure.tf` in any editor.

3.1. Define AWS provider

Define the provider block with AWS as chosen cloud provider. Also define these properties: `region` , `access_key` , and `secret_key` ; with values derived from the created IAM user.

Write a block of code below into `infrastructure.tf`

```
provider "aws" {
    region = "ap-southeast-1"
    access_key = "AKIAWLTS5CSXP7E3YLWG"
    secret_key = "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"
}
```

3.2. Generate new key pair then upload to AWS

Define new `aws_key_pair` resource block with local name: `my_instance_key_pair` . Put the previously generated `id_rsa.pub` public key inside the block to upload it to AWS.

```
resource "aws_key_pair" "my_instance_key_pair" {
    key_name = "terraform_learning_key_1"
    public_key = file("id_rsa.pub")
}
```

3.3. Book a VPC, and enable internet gateway on it

Book a VPC, name it `my_vpc`. Then enable internet gateway on it. Each part of the code below is self-explanatory.

```
# allocate a vpc named my_vpc.
resource "aws_vpc" "my_vpc" {
    cidr_block = "10.0.0.0/16"
    enable_dns_hostnames = true
}

# setup internet gateway for my_vpc.
resource "aws_internet_gateway" "my_vpc_igw" {
    vpc_id = aws_vpc.my_vpc.id
}

# attach the internet gateway my_vpc_igw into my_vpc.
resource "aws_route_table" "my_public_route_table" {
    vpc_id = aws_vpc.my_vpc.id
    route {
        cidr_block = "0.0.0.0/0"
        gateway_id = aws_internet_gateway.my_vpc_igw.id
    }
}
```

3.4. Allocate two different subnets on two different availability zones (within the same region)

Application Load Balancer or ALB requires two subnets setup on two availability zones (within the same region).

In this example, the region we used is `ap-southeast-1`, as defined in the provider block above (see 3.1). There are two zones available within this region, `ap-southeast-1a` and `ap-southeast-1b`. The ALB (not classic network load balancer) requires at least to be enabled on two different zones, so we will use those two.

```
# prepare a subnet for availability zone ap-southeast-1a.
resource "aws_subnet" "my_subnet_public_southeast_1a" {
    vpc_id = aws_vpc.my_vpc.id
    cidr_block = "10.0.0.0/24"
    availability_zone = "ap-southeast-1a"
}
# associate the internet gateway into newly created subnet for ap-southeast-1a
resource "aws_route_table_association" "my_public_route_association_for_southea
    subnet_id = aws_subnet.my_subnet_public_southeast_1a.id
    route_table_id = aws_route_table.my_public_route_table.id
}

# prepare a subnet for availability zone ap-southeast-1b
resource "aws_subnet" "my_subnet_public_southeast_1b" {
    vpc_id = aws_vpc.my_vpc.id
    cidr_block = "10.0.1.0/24"
    availability_zone = "ap-southeast-1b"
}
# associate the internet gateway into newly created subnet for ap-southeast-1b
resource "aws_route_table_association" "my_public_route_association_for_southea
    subnet_id = aws_subnet.my_subnet_public_southeast_1b.id
    route_table_id = aws_route_table.my_public_route_table.id
}
```

The internet gateway associated with two zones that we just created. In this example, it is required for the application hosted within instances on these zones to be able to connect to the internet.

3.5. Define ALB resource block, listener, security group, and target group

The ALB will be created with two subnets attached (subnets from `ap-southeast-1a` and `ap-southeast-1b`).

```
# create an Application Load Balancer.
# attach the previous availability zones' subnets into this load balancer.
resource "aws_lb" "my_alb" {
    name = "my-alb"
    internal = false # set lb for public access
    load_balancer_type = "application" # use Application Load Balancer
    security_groups = [aws_security_group.my_alb_security_group.id]
    subnets = [ # attach the availability zones' subnets.
        aws_subnet.my_subnet_public_southeast_1a.id,
        aws_subnet.my_subnet_public_southeast_1b.id
    ]
}
```

The security group for our load balancer has only two rules.

- Allow only incoming TCP/HTTP request on port `80`.
- Allow every kind of outgoing request.

```
# prepare a security group for our load balancer my_alb.
resource "aws_security_group" "my_alb_security_group" {
    vpc_id = aws_vpc.my_vpc.id
    ingress {
        from_port = 80
        to_port = 80
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }
    egress {
        from_port = 0
        to_port = 0
        protocol = "-1"
        cidr_blocks = ["0.0.0.0/0"]
    }
}
```

Next, we shall prepare the ALB listener. The load balancer will listen for every incoming request to port `80`, and then the particular request will be directed towards port `8080` on the instance.

Port `8080` is chosen here because the application (that will be deployed later) will listen to this port.

```
# create an alb listener for my_alb.
# forward rule: only accept incoming HTTP request on port 80,
# then it'll be forwarded to port target:8080.
resource "aws_lb_listener" "my_alb_listener" {
    load_balancer_arn = aws_lb.my_alb.arn
    port = 80
    protocol = "HTTP"
    default_action {
        target_group_arn = aws_lb_target_group.my_alb_target_group.arn
        type = "forward"
    }
}

# my_alb will forward the request to a particular app,
# that listen on 8080 within instances on my_vpc.
resource "aws_lb_target_group" "my_alb_target_group" {
    port = 8080
    protocol = "HTTP"
    vpc_id = aws_vpc.my_vpc.id
}
```

3.6. Define launch config (and it's required dependencies) for auto-scaling

We are not going to simply create an instance then deploy the application into it. Instead, the instance creation and app deployment will be automated using AWS auto-scaling feature.

In the resource block below, we will set up the launch configuration for the auto-scaling. This launch config is the one that decides how the instance will be created.

- The *Amazon Linux 2 AMI t2.micro* is used here.
- The launched instance will have a public IP attached, this is better to be set to `false`, but in here we might need it for testing purposes.
- The previously allocated key pair will also be used on the instance, to make it accessible through SSH access. This part is also for testing purposes.

Other than that, there is one point left that is very important, the `user_data`. The user data is a block of bash script that will be executed during instance bootstrap. We will use this to automate the deployment of our application. The whole script is stored in a file named `deployment.sh`, we will prepare it later.

```
# setup launch configuration for the auto-scaling.
resource "aws_launch_configuration" "my_launch_configuration" {

    # Amazon Linux 2 AMI (HVM), SSD Volume Type (ami-0f02b24005e4aec36).
    image_id = "ami-0f02b24005e4aec36"

    instance_type = "t2.micro"
    key_name = aws_key_pair.my_instance_key_pair.key_name # terraform_learning_

    security_groups = [aws_security_group.my_launch_config_security_group.id]

    # set to false on prod stage.
    # otherwise true, because ssh access might be needed to the instance.
    associate_public_ip_address = true
    lifecycle {
        # ensure the new instance is only created before the other one is destr
        create_before_destroy = true
    }

    # execute bash scripts inside deployment.sh on instance's bootstrap.
    # what the bash scripts going to do in summary:
    # fetch a hello world app from Github repo, then deploy it in the instance
    user_data = file("deployment.sh")
}
```

Below is the launch config security group. In this block, we define the security group specifically for the instances that will be created by the auto scale launch config. Three rules defined here:

- Allow incoming `TCP/SSH` access on port `22`.
- Allow `TCP/HTTP` access on port `8080`.
- Allow every kind of outgoing requests.

```
# security group for launch config my_launch_configuration.
resource "aws_security_group" "my_launch_config_security_group" {
    vpc_id = aws_vpc.my_vpc.id
    ingress {
        from_port = 22
        to_port = 22
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }
    ingress {
        from_port = 8080
        to_port = 8080
        protocol = "tcp"
        cidr_blocks = ["0.0.0.0/0"]
    }
    egress {
        from_port = 0
        to_port = 0
        protocol = "-1"
        cidr_blocks = ["0.0.0.0/0"]
    }
}
```

Ok, the autoscale launch config is ready, now we shall attach it into our ALB.

```
# create an autoscaling then attach it into my_alb_target_group.
resource "aws_autoscaling_attachment" "my_aws_autoscaling_attachment" {
    alb_target_group_arn = aws_lb_target_group.my_alb_target_group.arn
    autoscaling_group_name = aws_autoscaling_group.my_autoscaling_group.id
}
```

Next, we shall prepare the auto-scaling group config. This resource is used to determine when or on what condition the scaling process run.

- As per the below config, the auto-scaling will have a minimum of 2 instances alive, and 5 max.
- The `ELB` health check is enabled.
- The previous two subnets on `ap-southeast-1a` and `ap-southeast-1b` are applied.

```
# define the autoscaling group.
# attach my_launch_configuration into this newly created autoscaling group belo
resource "aws_autoscaling_group" "my_autoscaling_group" {
    name = "my-autoscaling-group"
    desired_capacity = 2 # ideal number of instance alive
    min_size = 2 # min number of instance alive
    max_size = 5 # max number of instance alive
    health_check_type = "ELB"

    # allows deleting the autoscaling group without waiting
    # for all instances in the pool to terminate
    force_delete = true

    launch_configuration = aws_launch_configuration.my_launch_configuration.id
    vpc_zone_identifier = [
        aws_subnet.my_subnet_public_southeast_1a.id,
        aws_subnet.my_subnet_public_southeast_1b.id
    ]
    timeouts {
        delete = "15m" # timeout duration for instances
    }
    lifecycle {
        # ensure the new instance is only created before the other one is destr
        create_before_destroy = true
    }
}
```

3.7. Print the ALB public DNS

Everything is pretty much done, except we need to print the ALB public DNS, so then we can do the testing.

```
# print load balancer's DNS, test it using curl.
#
# curl my-alb-625362998.ap-southeast-1.elb.amazonaws.com
output "alb-url" {
    value = aws_lb.my_alb.dns_name
}
```

4. App Deployment Script

We have done with the infrastructure code, next prepare the deployment script.

Create a file named `deployment.sh` in the same directory where the infra code is placed. It will contain bash scripts for automating app deployment. This file will be used by auto-scaling launcher to automate app setup during instance bootstrap.

The application is written in Go, and the AMI *Amazon Linux 2 AMI t2.micro* that used here does not have any Go tools ready, that's why we need to set it up.

> **Deploying app** means that the app is ready (has been built into binary), so what we need is simply just run the binary.
>
> However to make our learning process better, in this example, we are going to fetch the app source code and perform the build and deploy processes within the instance.

Ok, here we go, the bash script.

```
#!/bin/bash

# install git
sudo yum -y install git

# download go, then install it
wget https://dl.google.com/go/go1.14.linux-amd64.tar.gz
sudo tar -C /usr/local -xzf go1.14.linux-amd64.tar.gz

# clone the hello world app.
# The app is hosted on private repo,
# that's why the github token is used on cloning the repo
github_token=30542dd8874ba3745c55203a091c345340c18b7a
git clone https://$github_token:x-oauth-basic@github.com/novalagung/hello-world
    && echo "cloned" \
    || echo "clone failed"

# export certain variables required by go
export GO111MODULE=on
export GOROOT=/usr/local/go
export GOCACHE=~/gocache
mkdir -p $GOCACHE
export GOPATH=~/goapp
mkdir -p $GOPATH

# create local vars specifically for the app
export PORT=8080
export INSTANCE_ID=`curl -s http://169.254.169.254/latest/meta-data/instance-id

# build the app
cd hello-world
/usr/local/go/bin/go env
/usr/local/go/bin/go mod tidy
/usr/local/go/bin/go build -o binary

# run the app with nohup
nohup ./binary &
```

5. Run Terraform

5.1. Terraform initialization

First, run the `terraform init` command. This command will do some setup/initialization, certain dependencies (like AWS provider that we used) will be downloaded.

```
cd terraform-automate-aws-ec2-instance
terraform init
```

5.2. Terraform plan

Next, run `terraform plan`, to see the plan of our infrastructure. This step is optional, however, might be useful for us to see the outcome from the infra file.

5.3. Terraform apply

Last, run the `terraform apply` command to execute the infrastructure plan.

```
cd terraform-automate-aws-ec2-instance
terraform apply -auto-approve
```

The `-auto-approve` flag is optional, it will skip the confirmation prompt during execution.

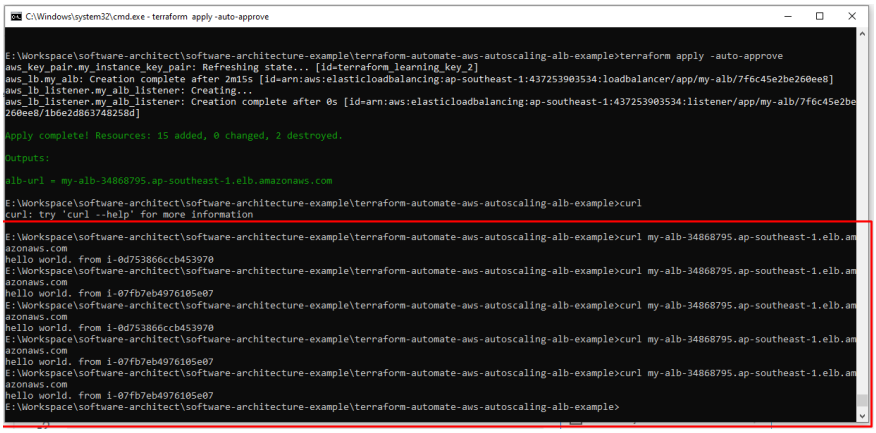
After the process is done, public DNS shall appear. Next, we shall test the instance.

---

6. Test Instance

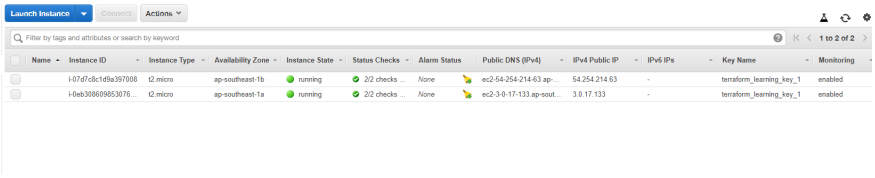
Use the `curl` command to make an HTTP request to the ALB public DNS instance.

```
curl -X GET my-alb-613171058.ap-southeast-1.elb.amazonaws.com
```



We can see from the image above, the HTTP response is different from one another across those multiple `curl` commands. The load balancer manages the traffic, sometimes we will get the instance A, B, etc.

In the AWS console, the instances that up and running are visible.


Launch Instance
Connect
Actions
Filter by tags and attributes or search by keyword
1 to 2 of 2
Name
Instance ID
Instance Type
Availability Zone
Instance State
Status Checks
Alarm Status
Public DNS (IPv4)
IPv4 Public IP
IPv6 IPs
Key Name
Monitoring
i-07d7c8c1d9a397008
t2.micro
ap-southeast-1b
running
2/2 checks ...
None
ec2-54-254-214-63.ap-...
54.254.214.63
-
terraform_learning_key_1
enabled
i-0eb308609853076...
t2.micro
ap-southeast-1a
running
2/2 checks ...
None
ec2-3-0-17-133.ap-sout...
3.0.17.133
-
terraform_learning_key_1
enabled