Intro

Please implement **one** of the following exercises, share your code (on GitHub in a public or private repo) with your Pulumi recruiter prior to your HW Review, and be prepared to walk through it on a call. If any details of the exercise are unclear, please ask for clarification.

> Please timebox your work on the exercise to **four hours or less**. It is acceptable to return an incomplete/non-working exercise.

Option 1 - Container Application (non-Kubernetes)

Using Pulumi, create and deploy a web application running in a managed container service such as AWS ECS, Azure Container Apps, or GCP Cloud Run. The web application should display a customized web page that returns a configurable value in its web page response.

Use **one** of AWS, Azure, or Google Cloud and your language of choice.

At a minimum your Pulumi application code should consist of:

- A Dockerfile to build your custom docker image
    - Suggestion: use this gist as your starting app and Docker image. Modify the application to read a configurable value via an environment variable.
- A managed container service, including but not limited to:
    - AWS ECS
    - Azure Container Apps
    - Azure Container Instances
    - GCP Cloud Run
- Relevant cloud provider resources to deploy and run the Docker image.
- A configurable value (via pulumi config) that changes the value displayed in the web page response.

> NOTE: you may not utilize any ECS functionality provided by the Pulumi AWSX package

Success criteria:

- Should be able to deploy the provided Web Application into your network.

Option 2 - Kubernetes Application

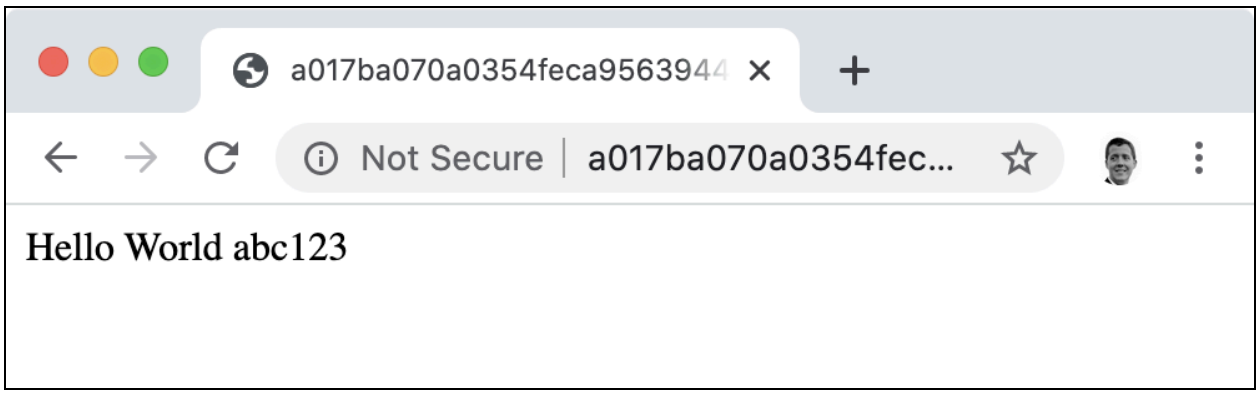
Using Pulumi, create and deploy a web application running in Kubernetes. The web application should display a customized web page that returns a configurable value in its web page response.

Use **one** of AWS, Azure, Google Cloud, or Digital Ocean and your language of choice.

At a minimum your Pulumi application code should consist of:

- A Dockerfile to build your custom docker image
    - Suggestion: use this gist as your starting app and Docker image. Modify the application to read a configurable value via an environment variable.
- A Kubernetes cluster
- Relevant Kubernetes resources to deploy and run the Docker image
- A configurable value (via pulumi config) that changes the value displayed in the web page response

For clarity, deploy a web application that serves a web page like below where abc123 is the configurable value:



Success criteria:

- Should be able to load the web page in a browser
- Should be able to change the configured value, run `pulumi up`, reload the web page and see the value change

Option 3 - Web Server Fleet Abstraction

Using Pulumi, create an abstraction for a “web server fleet” that supports at least two operating systems. Your application should accept the following (pseudocode) as input and provision virtual machines as defined by the inputs.

```
new WebServerFleet({
     subnets: ['subnet-abc123', 'subnet-abc123'],
     machines: [
         {os: 'ubuntu', size: 'small', count: 3},
         {os: 'amazonlinux', size: 'medium', count: 2},
     ],
})
```

Use **one** of AWS, Azure, Google Cloud, or Digital Ocean and your language of choice.

At a minimum your Pulumi application code should consist of:

- A minimal set of resources to build a VPC/network on the cloud you chose (no abstraction needed)
- An interface/class for the arguments for your web server fleet that takes the following:
    - The network and/or subnets to deploy into
    - The number of virtual machines to deploy
    - The “abstract” size of the virtual machines in the form of:
        - “small”
        - “medium”
        - “large”
    - The “abstract” operating system of the virtual machines in the form of:
        - “ubuntu”
        - “amazon"
        - (or other OS of your choice)
- A `WebServerFleet` abstraction via a ComponentResource implementation that does the following:
    - Provisions the defined number of VMs meeting using the provided inputs
    - Picks a different cloud machine type based on the abstract virtual machine size, e.g.:
        - “small” => “t2.medium”
        - “medium” => “t2.large”
        - “large” => “t2.2xlarge”
    - Picks the appropriate OS based on the OS type input, e.g.
        - “ubuntu” => “ubuntu-18.04”
        - “amazon” => “amzn2-ami-hvm-2.0”
- For each operating system implementation install a web server using a startup script via userdata or metadata startup script for the virtual machine.
    - Suggestion: use nginx for simplicity.

Success criteria:

- A minimal set of resources to build a VPC/network
- Given the WebServerFleet inputs, deploys the correct type and number of virtual machines specified
- Note: Ubuntu and Amazon Linux are just examples, use the operating systems of your choice