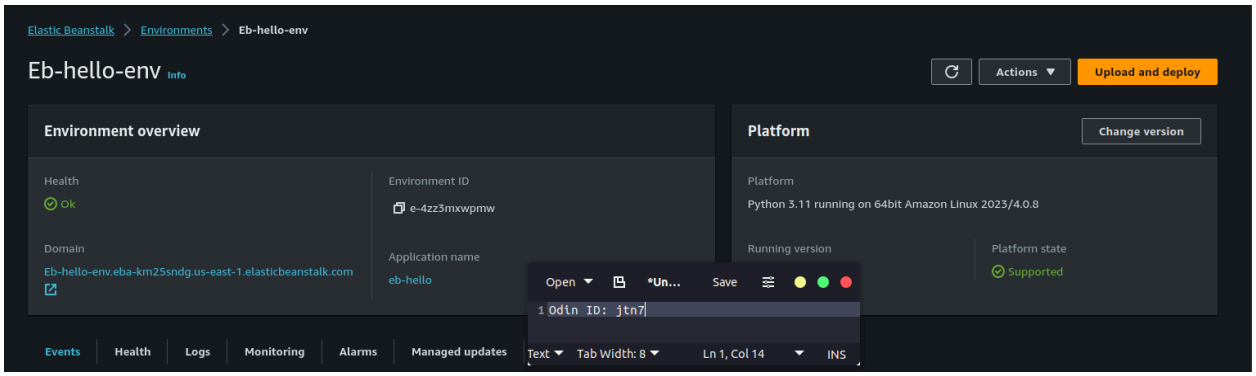
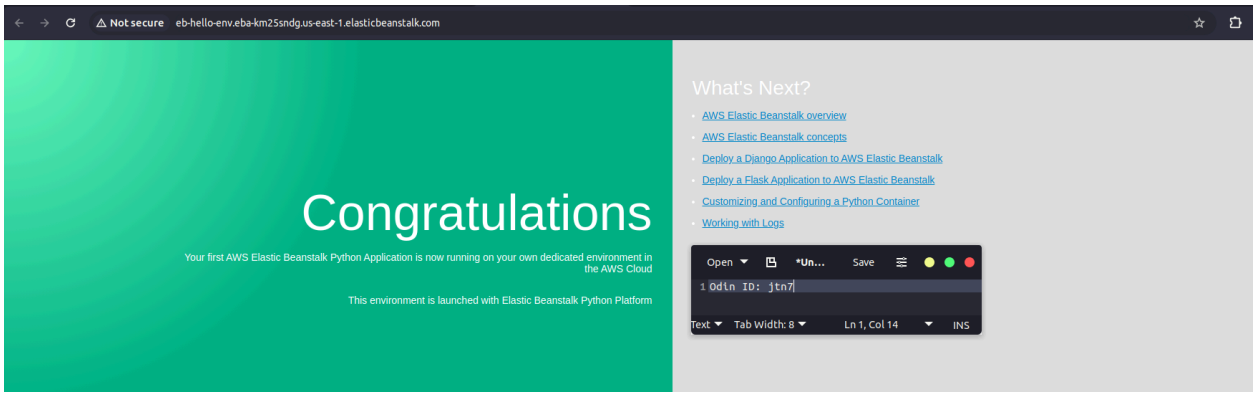
6.1a
When the launch succeeds, click on the URL for the site to view the application that has been deployed.

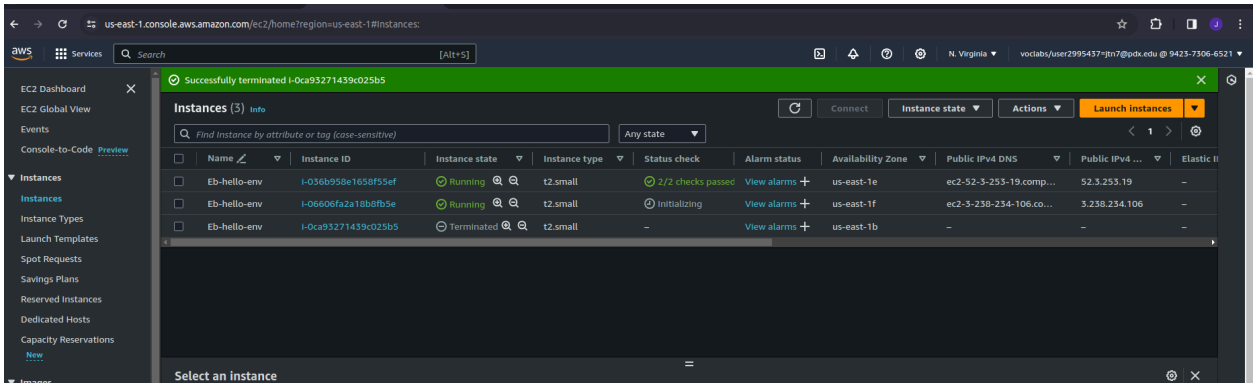
Screenshot of Elastic Beanstalk Eb-hello on console and clicked link.





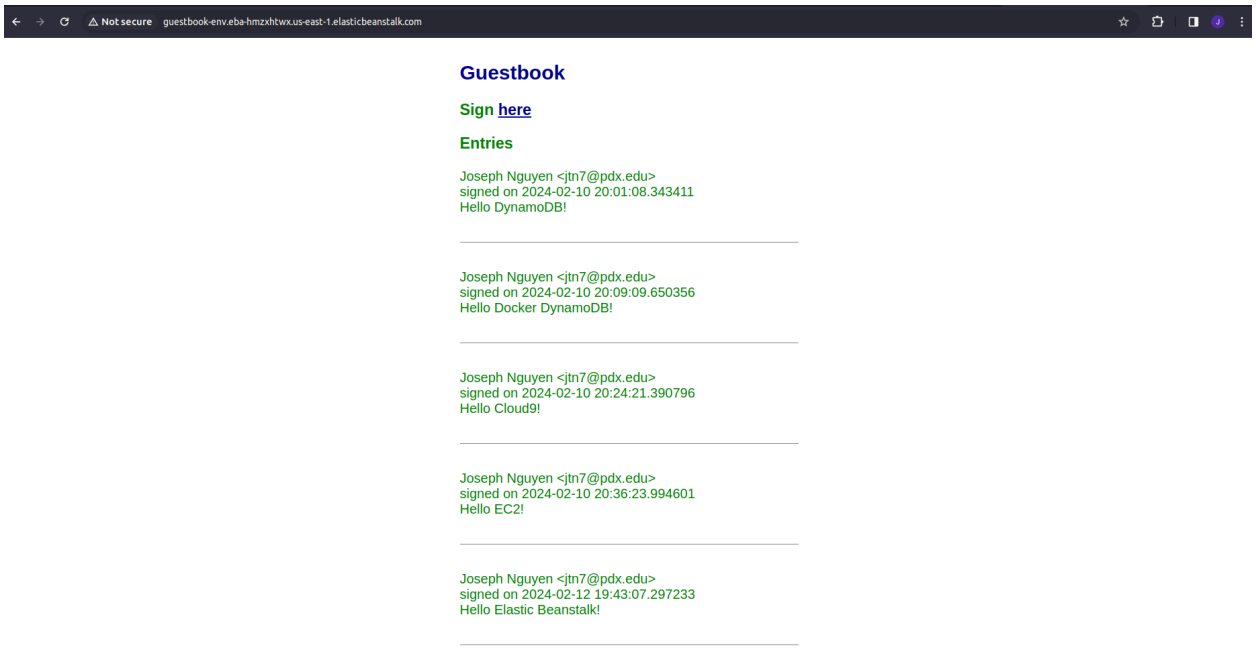
De-select the "Instance state: running" filter shown above in order to see the terminated instance after it shuts down. Terminating the instance will cause the number of instances to fall below our minimum of 2. Wait several minutes and then, using the Refresh icon in the EC2 UI, update the status. Elastic Beanstalk will notice the missing instance and launch a replacement.

Take a screenshot of the replacement VM being started.



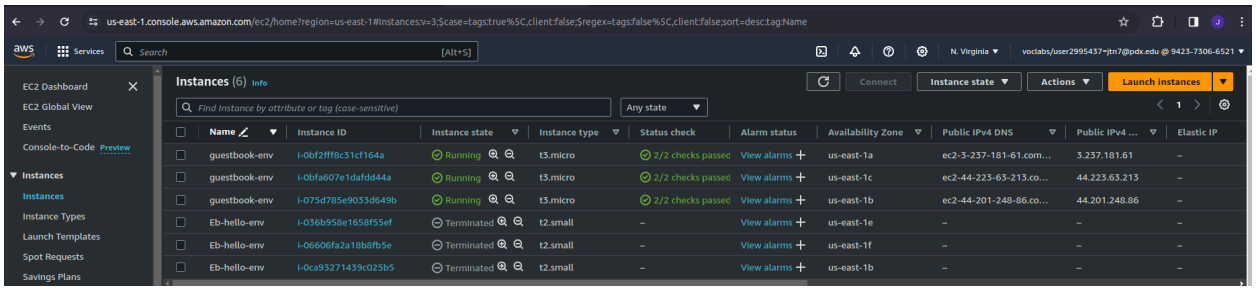
Visit the URL exported by the environment in the creation step and enter a message using your name, PSU e-mail address, and the message "Hello Elastic Beanstalk!".

Take a screenshot of the Guestbook including the URL with the entry in it.



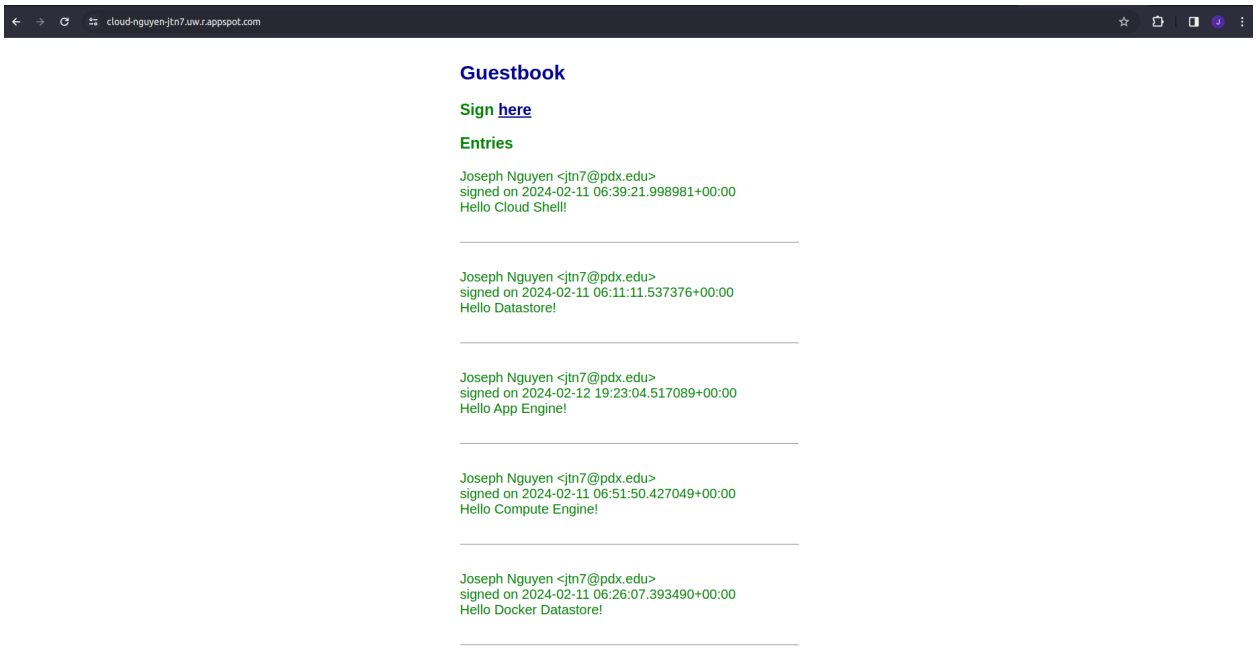
Then, visit the EC2 console to see that the specified minimum number of instances has been created

Take a screenshot of them.

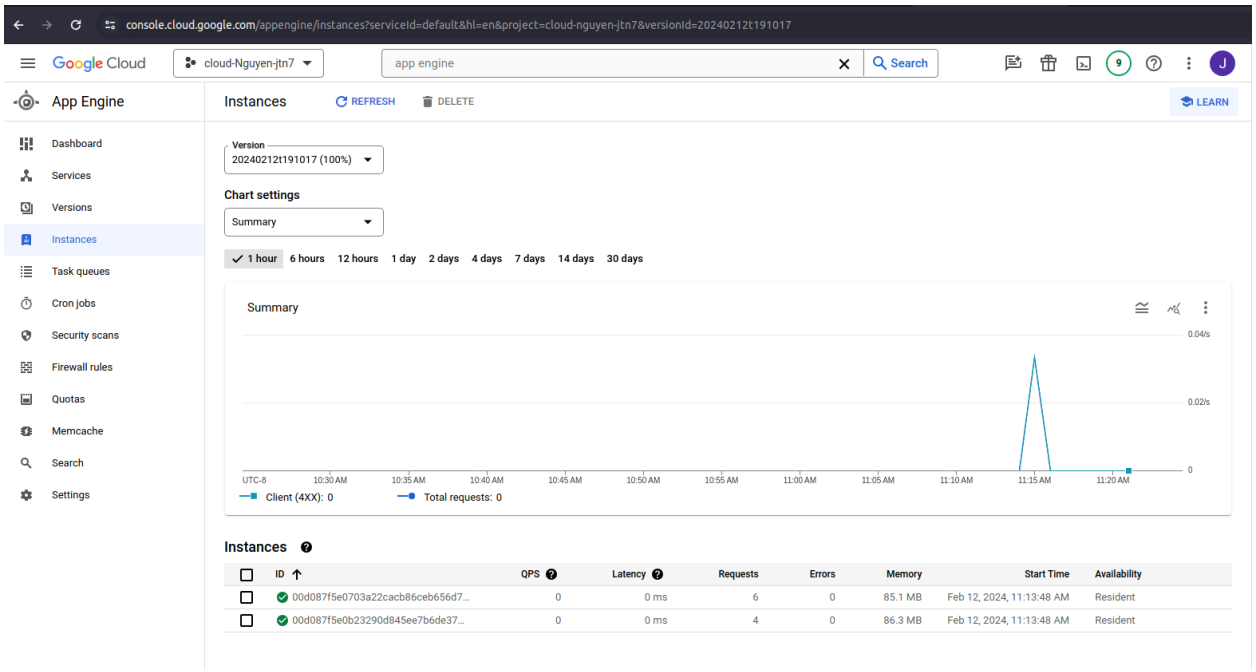


6.1g
When the deployment is complete, visit the URL in a web browser. Sign the guestbook with your name and PSU e-mail address with the message "Hello App Engine!".

Take a screenshot of the output that includes the URL in the address bar for your lab notebook.



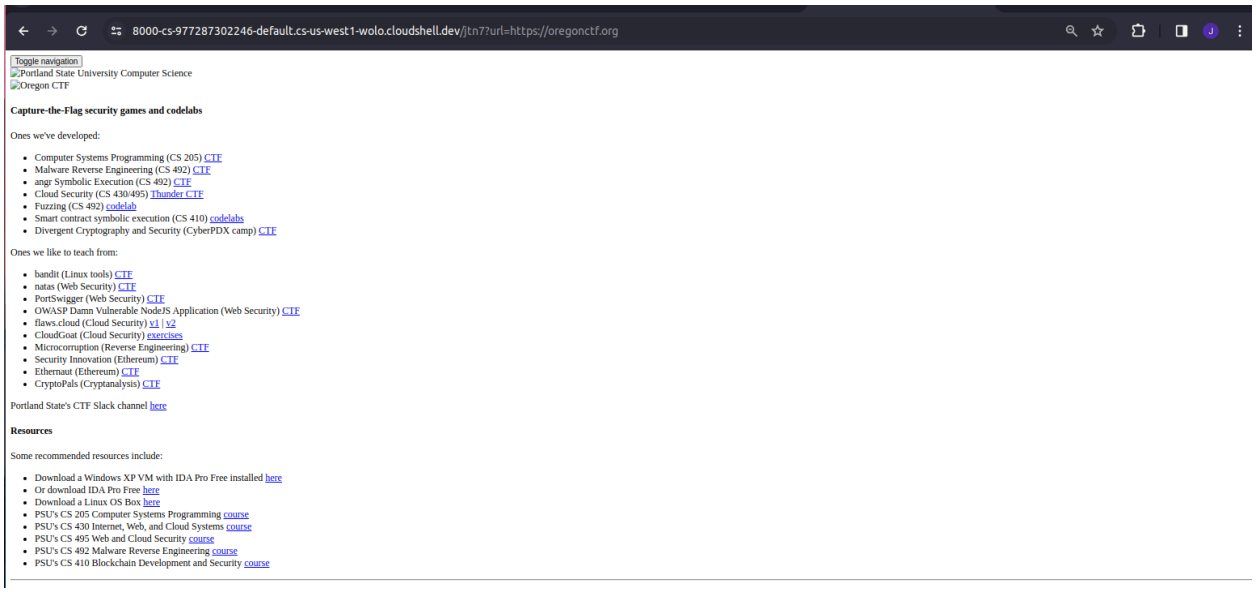
From the web console, visit the App Engine home page and click on "Instances" to view the machines that have been brought up to serve your application.
Take a screenshot of them.



6.2g

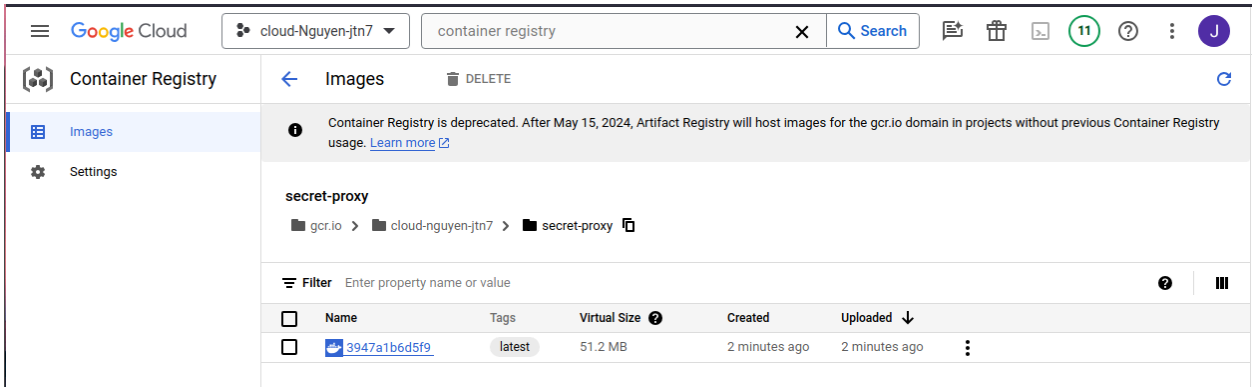
In order for the secret proxy to be created in our application, we must set the SECRET_PROXY_ROUTE environment variable. To do so, modify the docker run command to set this environment variable to /OdinID. (e.g. /wuchang). Bring up the Web Preview again and visit the secret proxy route. Enter the URL https://oregonctf.org .

Take a screenshot of the proxy and its results including the URL containing your OdinID



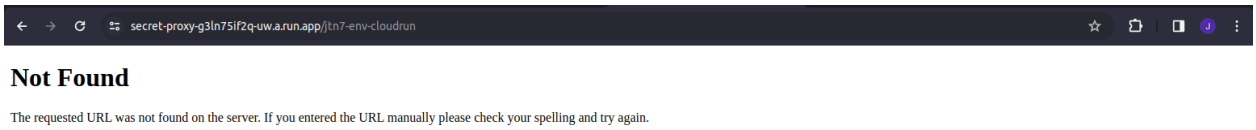
What is the security advantage of passing in the secret proxy route as an environment variable?
Only those with the SECRET_PROXY_ROUTE key can access the secret webpage. So even if the url somehow gets leaked or guessed, unauthorized users cannot access the website without the environment variable.

Then, visit Container Registry in the web console and click on the image you have created. Take a screenshot of the image in the registry that shows the size of the container for your lab notebook.

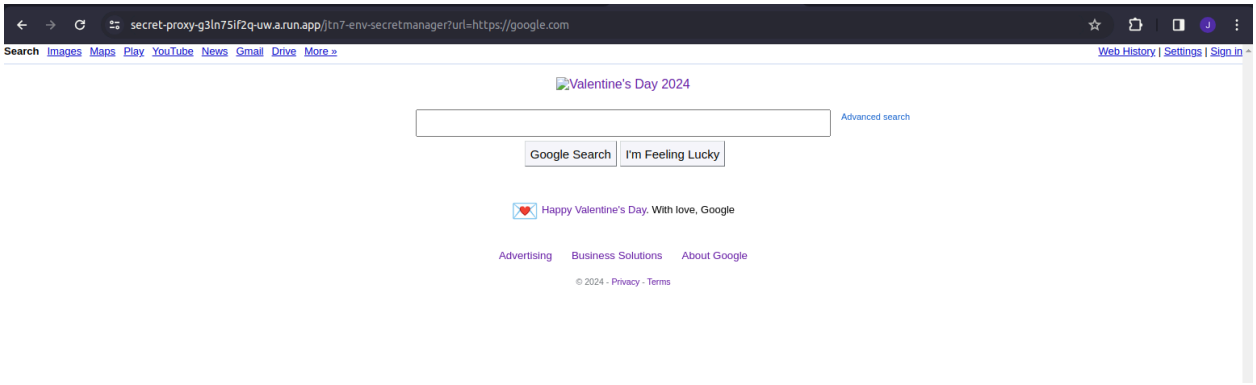


Visit the secret proxy route on the container. Note that because the container must be started up on-demand for the first time, there will be a slight delay on first access to it.
Take a screenshot of it that includes the proxy URL for your lab notebook.

Finally, remove the environment variable on Cloud Run by re-deploying with the following flag.
Reload the proxy route again to show its removal.
Take a screenshot of the error page that includes the proxy URL for your lab notebook.



Visit the new secret proxy route URL.
Enter https://google.com
Take a screenshot of it that includes the proxy URL for your lab notebook.

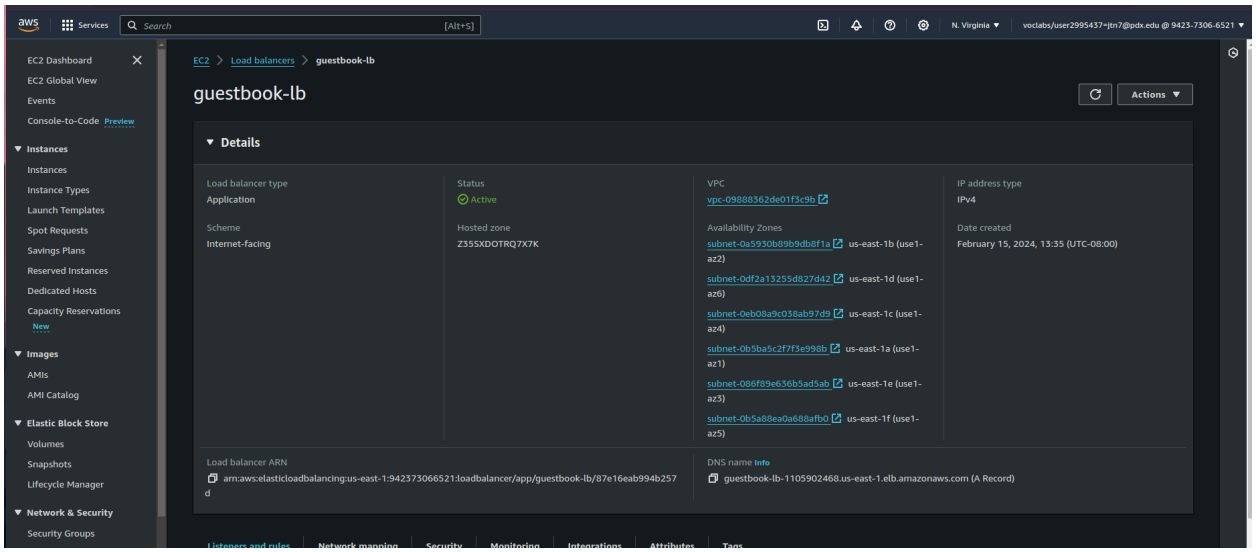


Attempt to access the Metadata service associated with the VM that runs your container by entering the following URLs into the proxy http://169.254.169.254/computeMetadata and http://169.254.169.254/computeMetadata/v1.
Identify the vulnerability in your lab notebook that Google has prevented.
Google is protecting a SSRF vulnerability (server-side request forgery) by preventing you from accessing the site's metadata without the appropriate headers. Hackers in the articles have been able to access the meta-data from Amazon and Shopify to steal information by bypassing seemingly protected data.

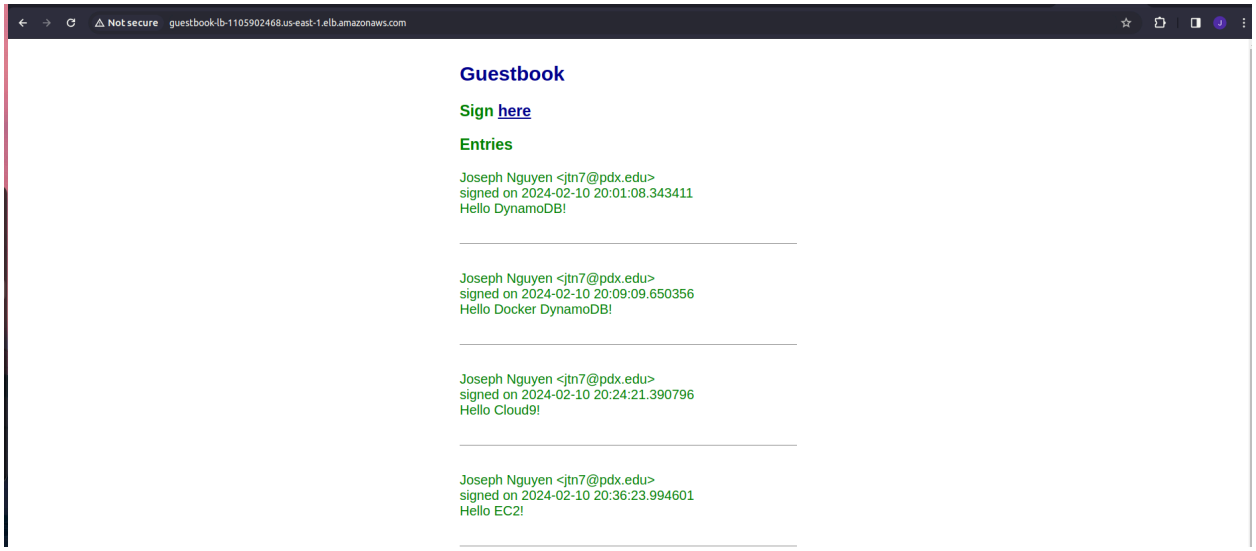
6.3a

The load balancer has a DNS name associated with it that is the frontend endpoint for the service.

Take a screenshot of the DNS name of the guestbook-lb load balancer for your lab notebook

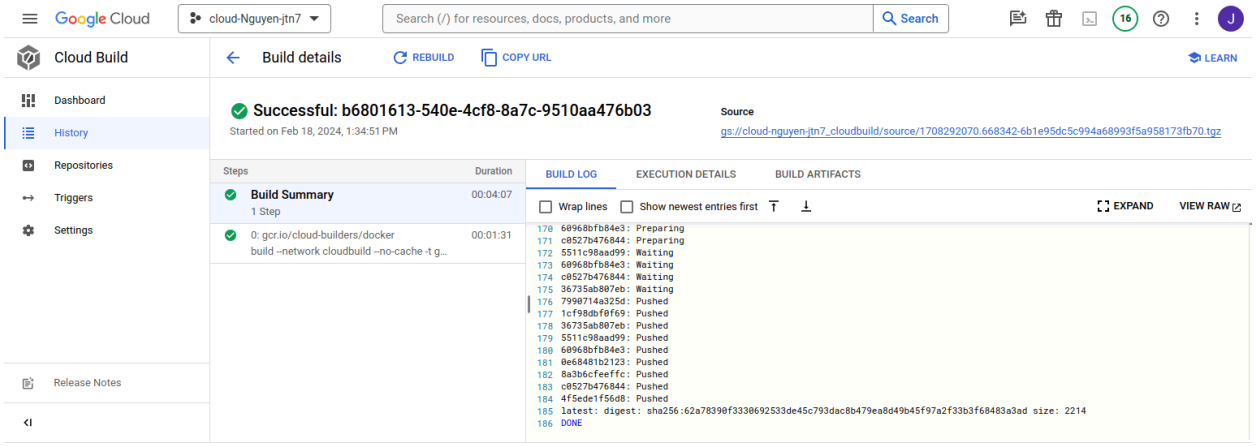


Take a screenshot of the Guestbook app running in a browser that includes the DNS name of the site.

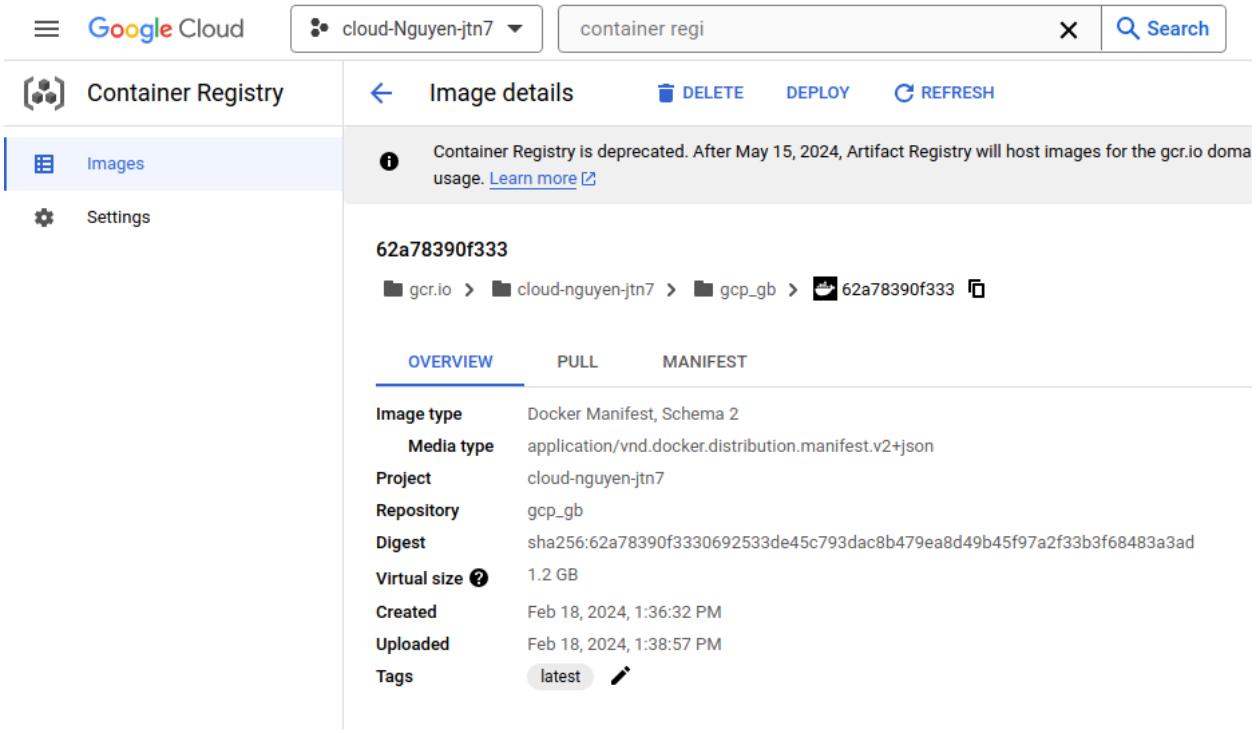


6.3g
After executing the command, from the web console, visit the Cloud Build home page, click on History, then on the build you just executed. Scroll down to see that the docker command that Cloud Build has executed on your behalf resulted in the image being created.

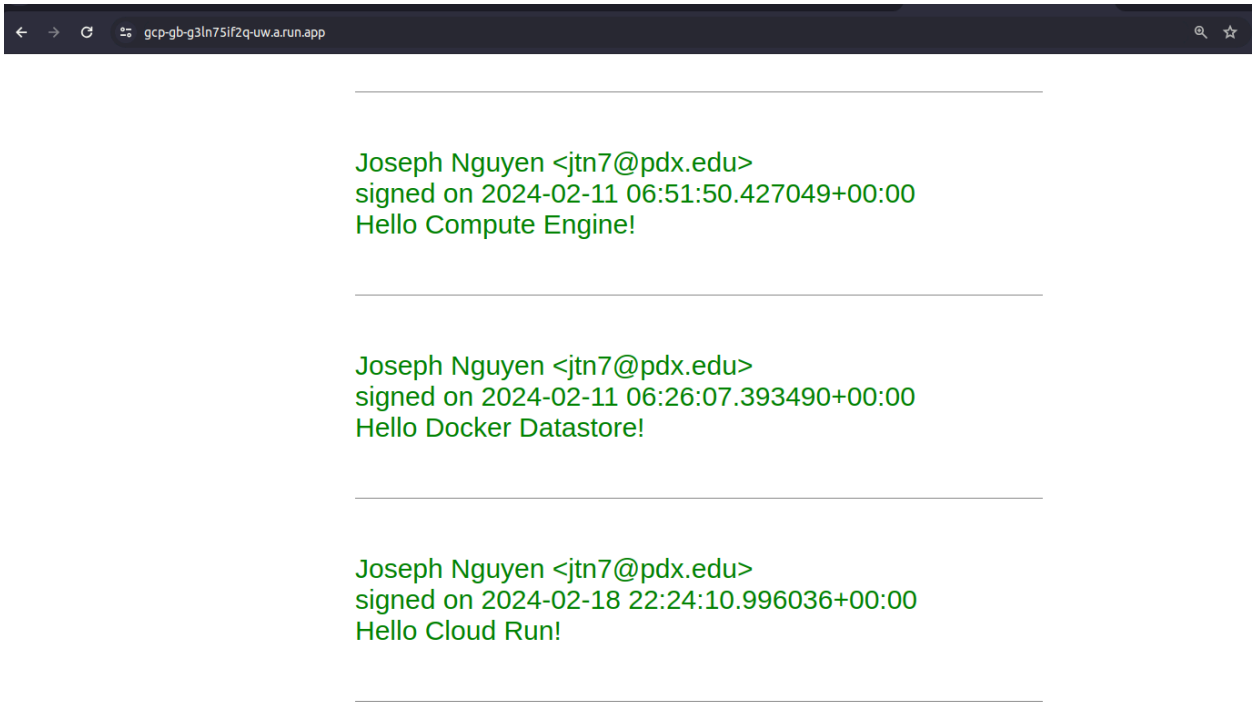
Take a screenshot that includes the output of the command and the time it took to execute.



To inspect the image that has been created, visit Container Registry.
Take a screenshot showing the container image and its virtual size



Add an entry with the message "Hello Cloud Run!". Show your Guestbook app running in a browser.
Take a screenshot that includes the URL Cloud Run has created for your site.



View the "Details" section to the right and answer the following questions:
What port do container instances listen on? **Port 8080**
What are the maximum number of instances Cloud Run will autoscale up to for your service? **100**

6.4g
Visit the file and examine the code for __blur_image.
Answer the following questions for your lab notebook:

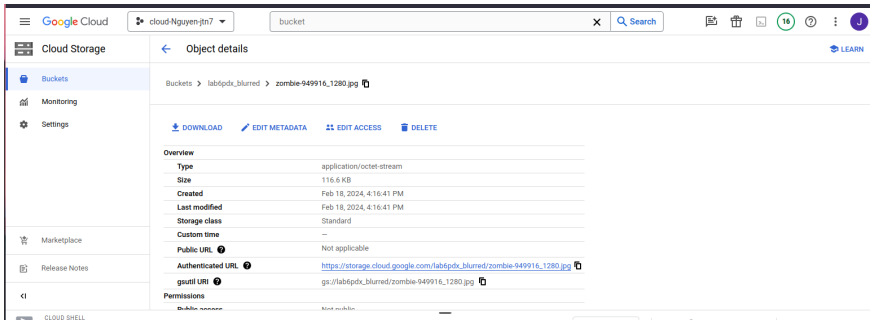
After downloading the file from the bucket, where is it stored?
It is stored in temp_local_filename, a temporary file.

What class in the ImageMagick package is used to do the blurring of the file?
The Image class is used to do the blurring of the file.

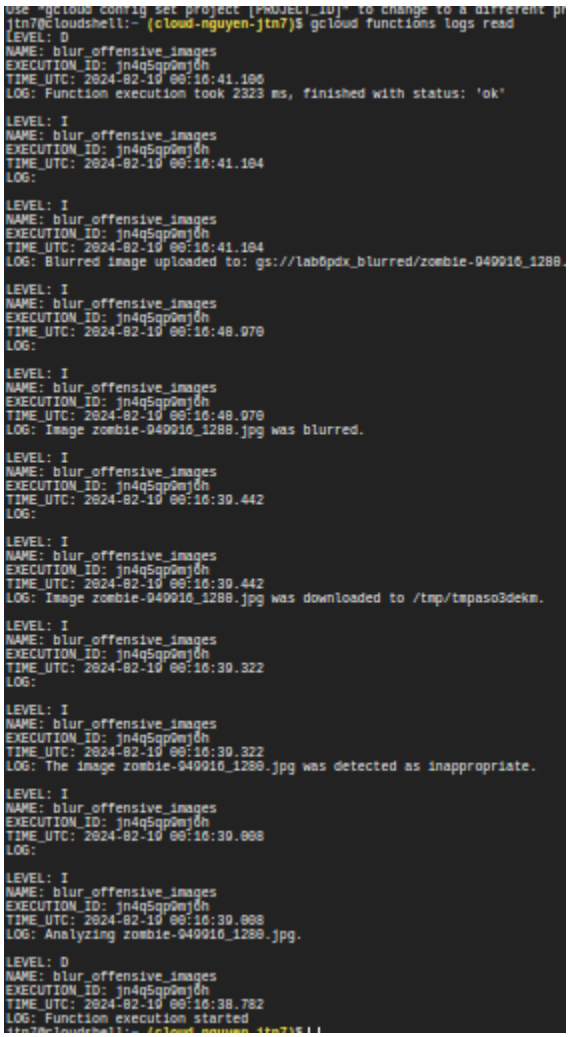
What lines of code perform the blurring of the image and its storage back into the filesystem?

```
    print(f"Image {file_name} was downloaded to {temp_local_filename}.")

    # Blur the image using ImageMagick.
    with Image(filename=temp_local_filename) as image:
        image.resize(*image.size, blur=16, filter="hamming")
        image.save(filename=temp_local_filename)

    print(f"Image {file_name} was blurred.")

    # Upload result to a second bucket, to avoid re-triggering the function.
    # You could instead re-upload it to the same bucket + tell your function
    # to ignore files marked as blurred (e.g. those with a "blurred" prefix)
    blur_bucket_name = os.getenv("BLURRED_BUCKET_NAME")
    blur_bucket = storage_client.bucket(blur_bucket_name)
    new_blob = blur_bucket.blob(file_name)
    new_blob.upload_from_filename(temp_local_filename)
    print(f"Blurred image uploaded to: gs://{blur_bucket_name}/{file_name}")
```

Once uploaded, the function should automatically execute via the bucket trigger and blur the image.
Take a screenshot of the blurred image in the output bucket for your lab notebook



The log entries for the functions can be read via the web console or command-line via gcloud functions logs read
Include a screenshot of the output logs that show that the above image was blurred.



By default, the subscription we create will be pull-based. One can specify push-based subscriptions using a variety of flags when creating subscriptions.
Attempt to pull a message from the subscription.
Why are there no items returned?
The message was created before we subscribed to the message broker, therefore we did not receive message # 1.

Back in Cloud Shell, publish a second message onto the topic.
What is the messageId of the published message?

```
jtn7@cloudshell:~ (cloud-nguyen-jtn7)$ gcloud pubsub topics publish topic-jtn7 --message="Message #2"
messageIds:
- '10523082996635659'
```

Then, back in the VM, attempt to pull a message from the subscription again.
The message from Cloud Shell should be received within the VM, with an acknowledgement sent back to Cloud Pub/Sub, notifying the service that the message has been received by all of its subscribers and can be deleted.
Take a screenshot of the output of the successful pull that includes the message and its messageId.

```
jtn7@pubsub:~$ gcloud pubsub subscriptions pull sub-${USER}
|    DATA     |      MESSAGE_ID     | ORDERING_KEY | ATTRIBUTES | DELIVERY_ATTEMPT |
                                                                      ACK_ID
|
| Message #2 | 10523082996635659 |              |            |                  | RFAGFixdRkhRNxkIaFEOT14
jPzUgKEURAQgUBXx9cVtFdV9bGgdRDRlyfGkhOA9HAwpDUHtVWxENem1cbYyo94dEX0B0al8RAgJNUXdWXR0ObVpZfC_E7vmr3pKLSEAv
OYm8qeBpe_Xq3atuZiA9XxJLLD5-NTBFQV5AEkw-BURJUytDCypYEU4EISE-MD5FUw |
```

Go back to your Cloud Shell session running the publisher. Then, enter several messages.
Take a screenshot showing the messageIds and messages sent

```
(env) jtn7@cloudshell:~ (cloud-nguyen-jtn7)$ python3 publisher.py
Enter a message to send: never gonna give you up
Published 10523186382678512 to topic projects/cloud-nguyen-jtn7/topics/my_topic
Enter a message to send: never gonna let you down
Published 10523357802070141 to topic projects/cloud-nguyen-jtn7/topics/my_topic
Enter a message to send: never gonna run around and desert you
Published 10523074451838460 to topic projects/cloud-nguyen-jtn7/topics/my_topic
Enter a message to send:
```

Then, go back to the VM session running the subscriber. The messages from the publisher should appear along with their messageIds.
Take a screenshot showing the same messageIds and messages received

```
(env) jtn7@pubsub:~$ python3 subcriber.py
Received message 10523186382678512: 2024-02-19 00:53:23 (projects/cloud-nguyen-jtn7/topics/my_topic) : never gonna give you up
Received message 10523357802070141: 2024-02-19 00:53:30 (projects/cloud-nguyen-jtn7/topics/my_topic) : never gonna let you down
Received message 10523074451838460: 2024-02-19 00:54:08 (projects/cloud-nguyen-jtn7/topics/my_topic) : never gonna run around and desert you
```