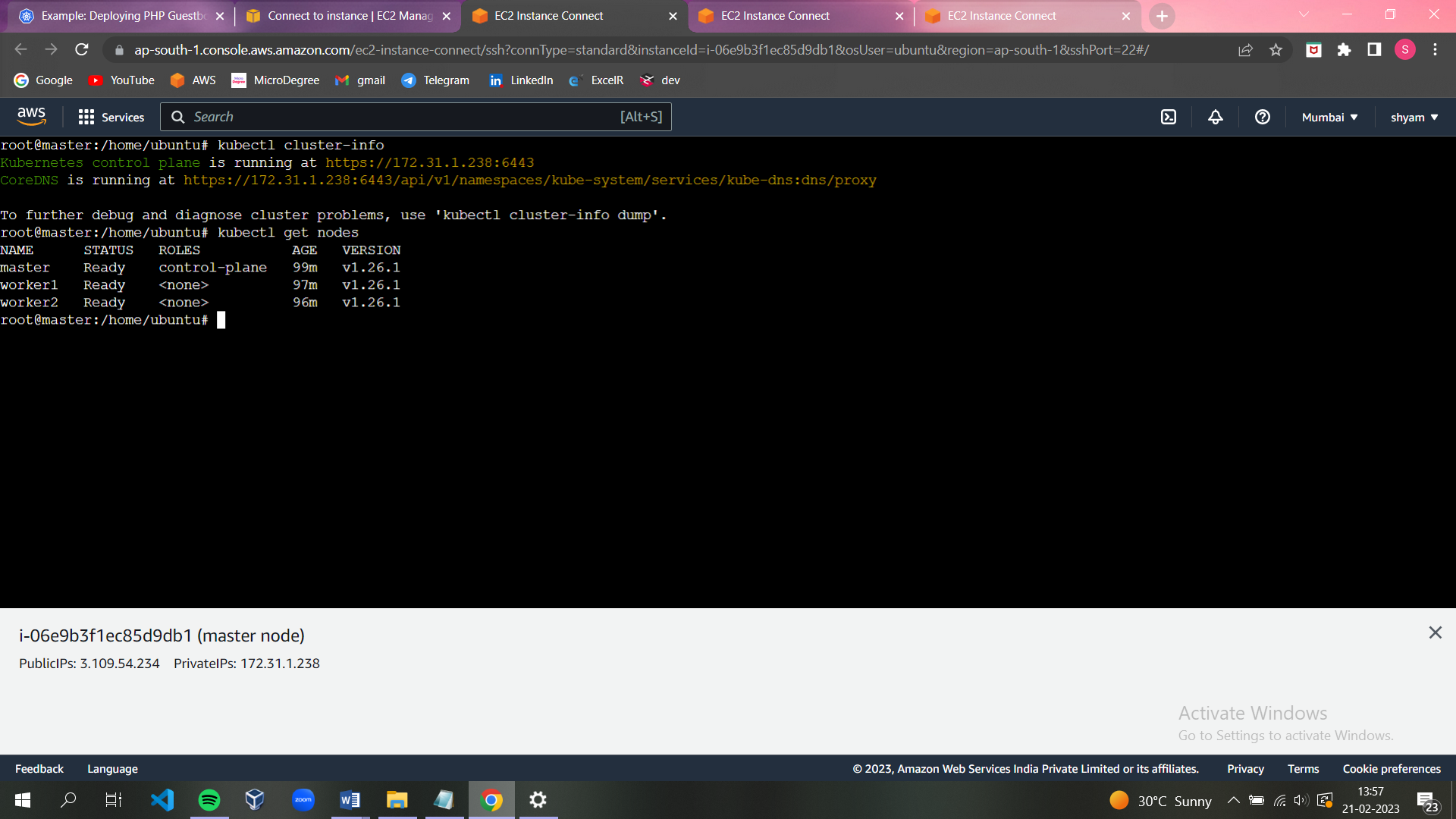
**PROJECT – 2**

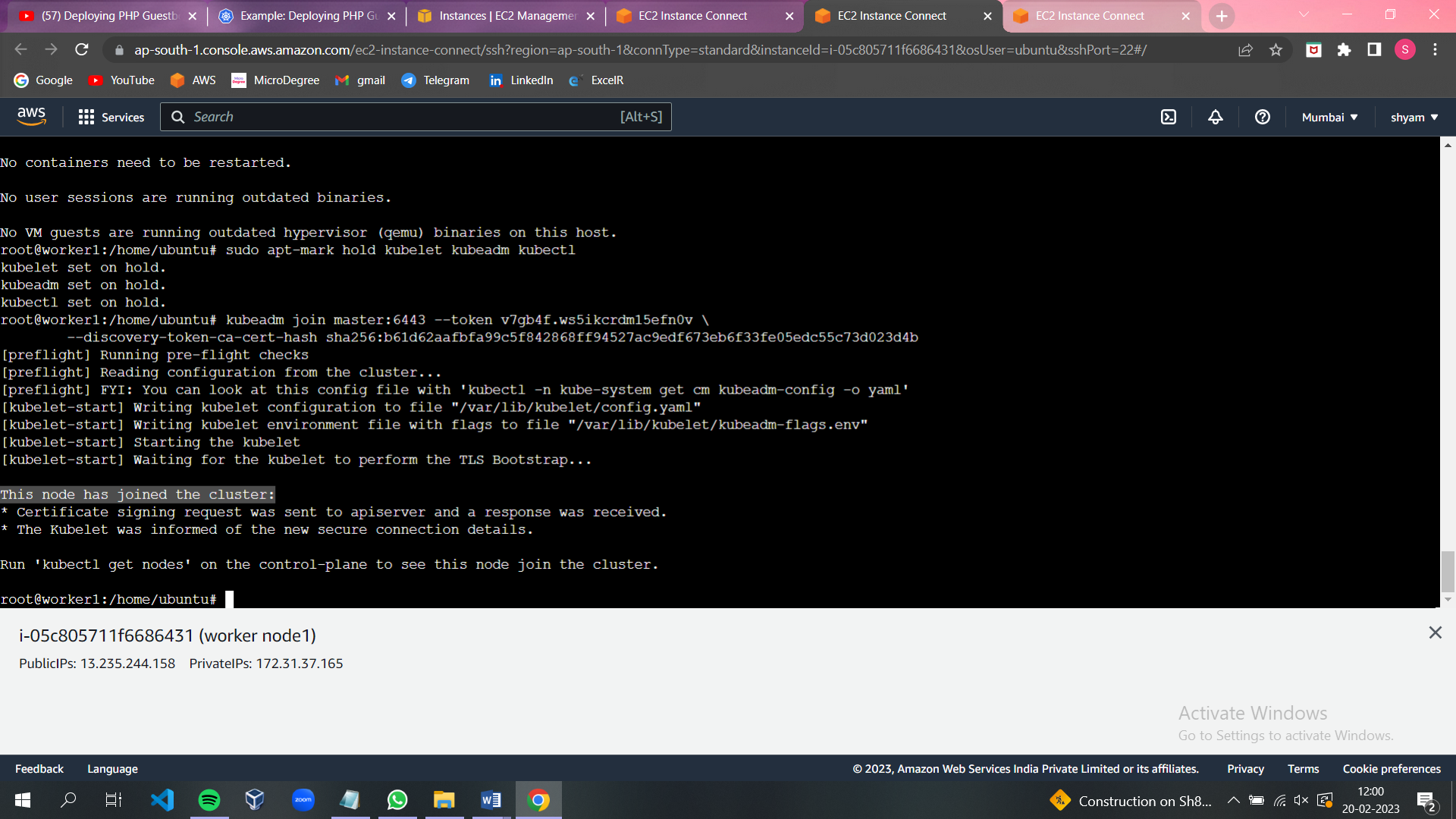
**DEPLOYING PHP GUESTBOOK APPLICATION WITH REDIS**

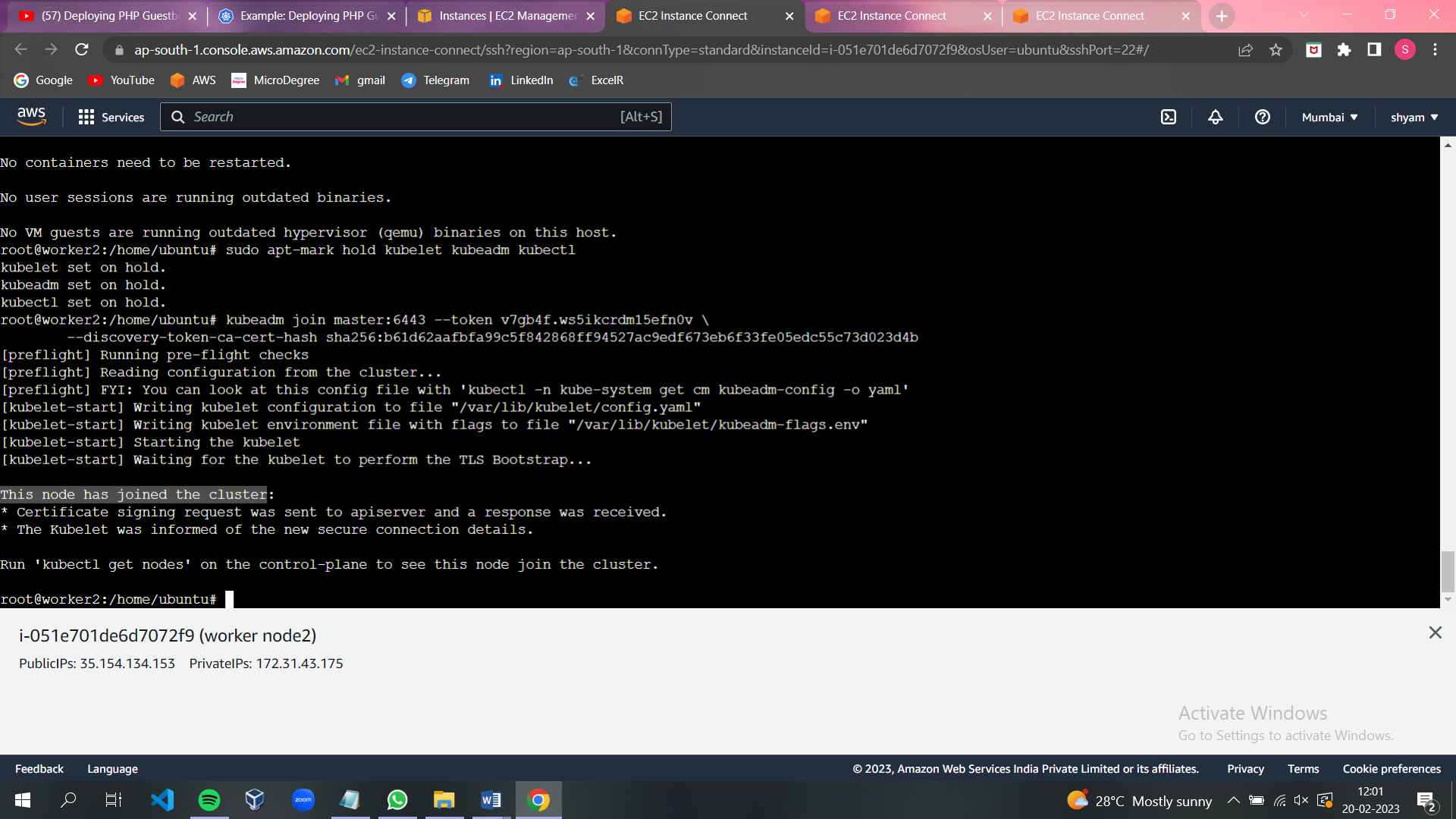
**STEP1- Creatng a kubernetes cluster.**

We need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster.

It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts.







**STEP2- Creating the Redis Deployment.**

The manifest file, included below, specifies a Deployment controller that runs a single replica Redis Pod.

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: redis-leader

**labels**:

**app**: redis

**role**: leader

**tier**: backend

**spec**:

**replicas**: 1

**selector**:

**matchLabels**:

**app**: redis

**template**:

**metadata**:

**labels**:

**app**: redis

**role**: leader

**tier**: backend

**spec**:

**containers**:

- **name**: leader

**image**: "docker.io/redis:6.0.5"

**resources**:

**requests**:

**cpu**: 100m

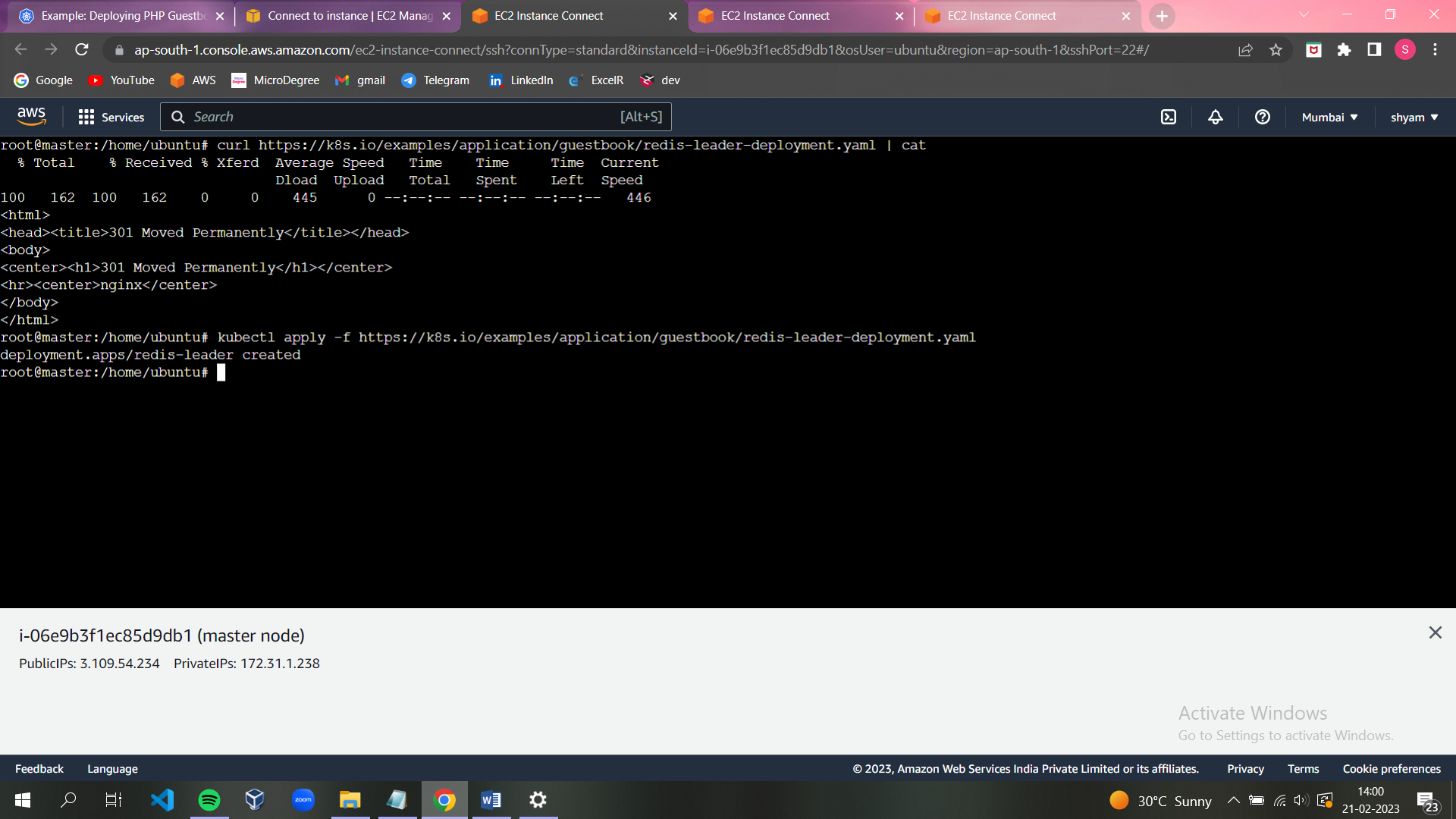
**memory**: 100Mi

**ports**:

- **containerPort**: 6379

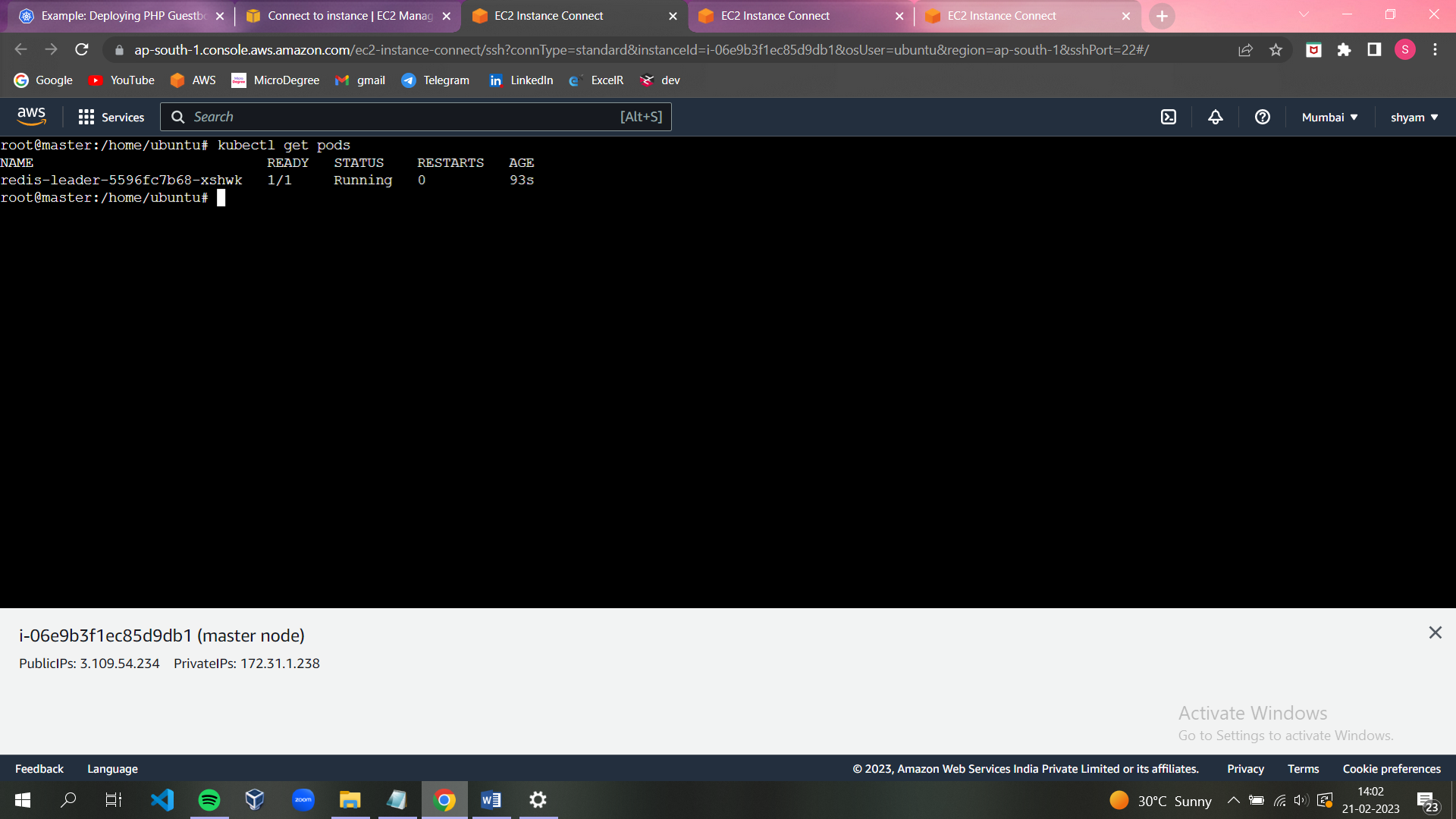
**Apply the Redis Deployment from the redis-leader-deployment.yaml file:**

kubectl apply -f <https://k8s.io/examples/application/guestbook/redis-leader-deployment.yaml>



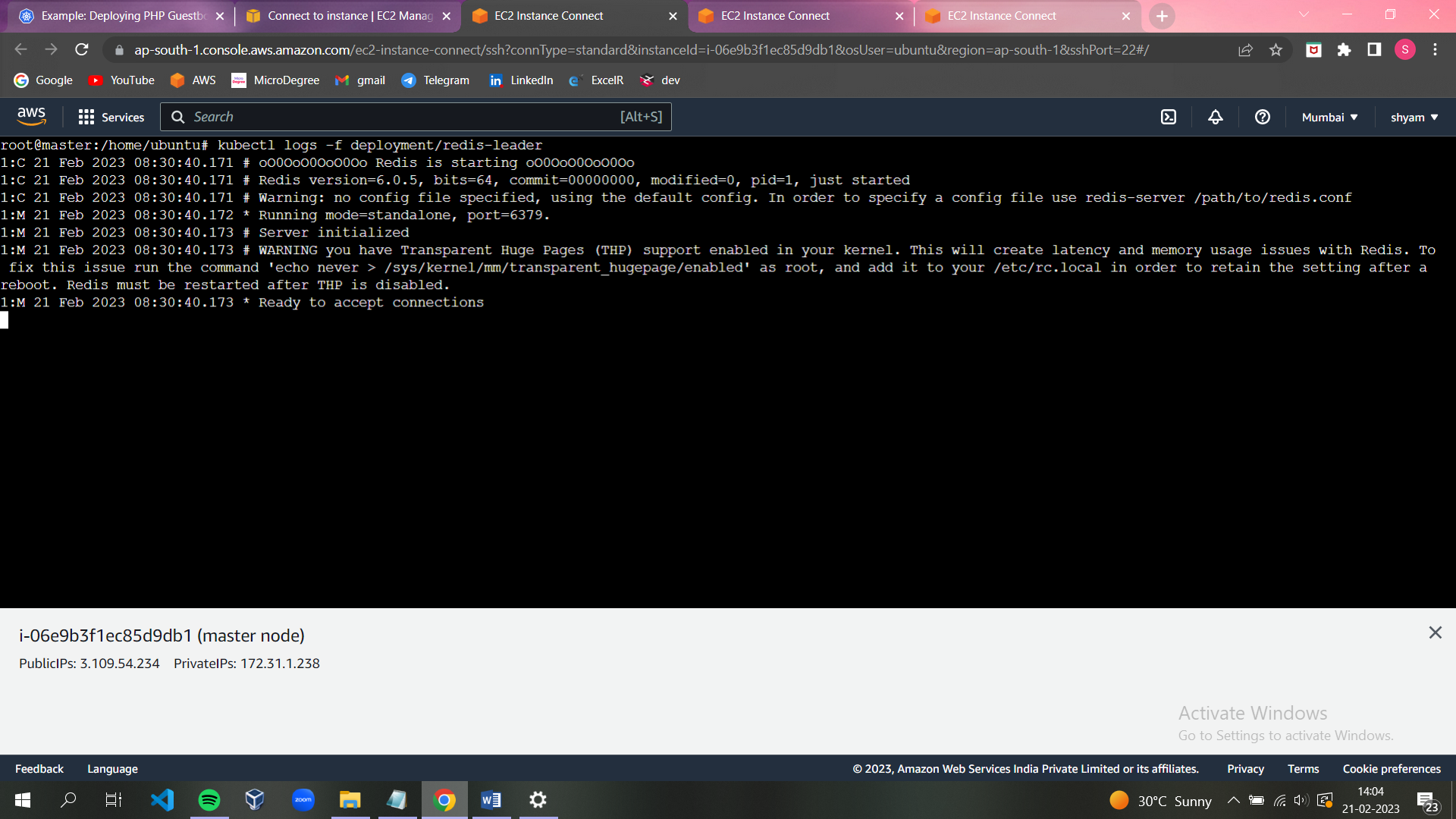
**Query the list of Pods to verify that the Redis Pod is running:**

kubectl get pods



**Run the following command to view the logs from the Redis leader Pod:**

kubectl logs -f deployment/redis-leader



### STEP3- Creating the Redis leader Service.

The guestbook application needs to communicate to the Redis to write its data.

We need to apply a [Service](https://kubernetes.io/docs/concepts/services-networking/service/) to proxy the traffic to the Redis Pod. A Service defines a policy to access the Pods.

*# SOURCE: https://cloud.google.com/kubernetes-engine/docs/tutorials/guestbook*

**apiVersion**: v1

**kind**: Service

**metadata**:

**name**: redis-leader

**labels**:

**app**: redis

**role**: leader

**tier**: backend

**spec**:

**ports**:

- **port**: 6379

**targetPort**: 6379

**selector**:

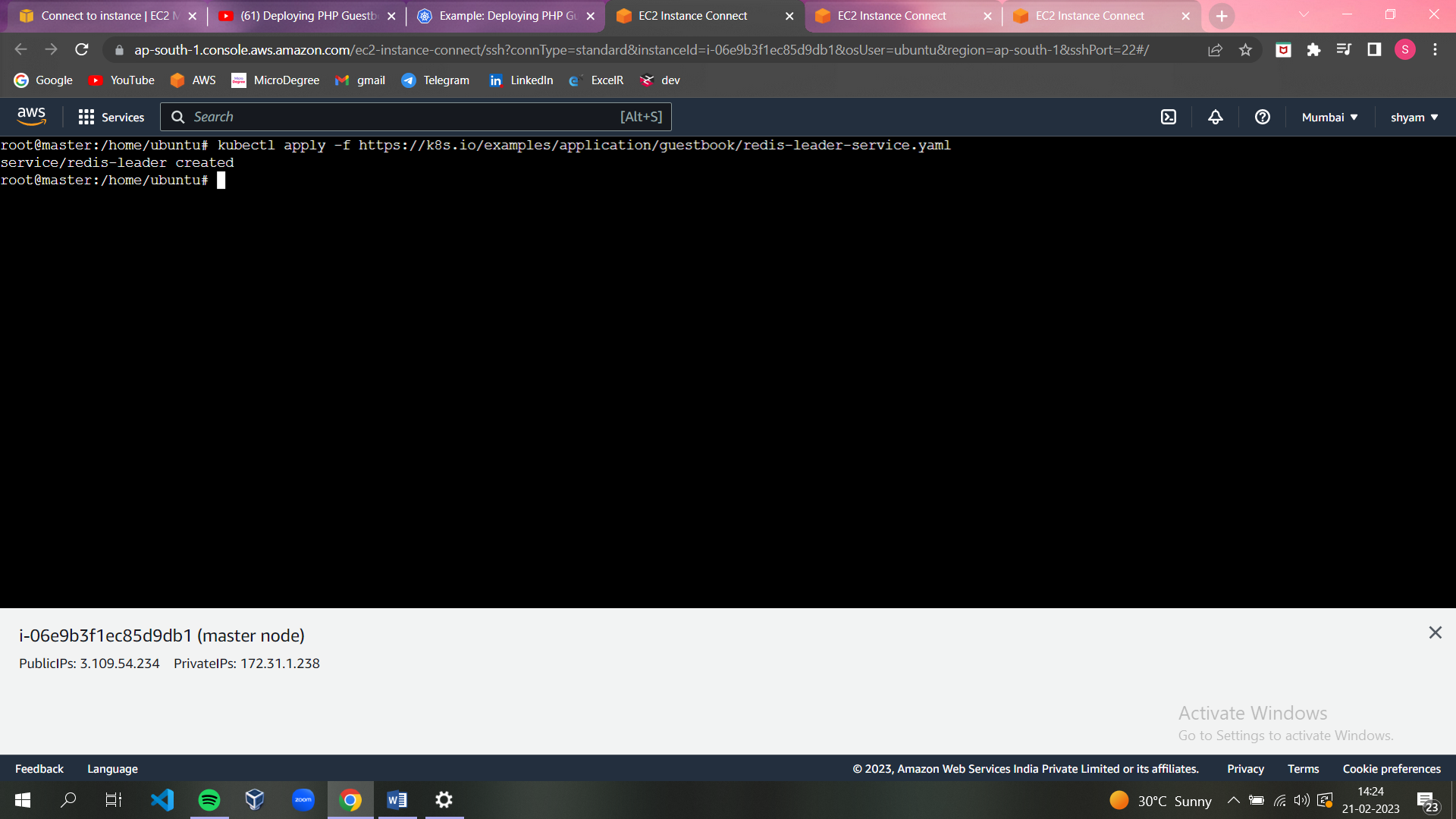
**app**: redis

**role**: leader

**tier**: backend

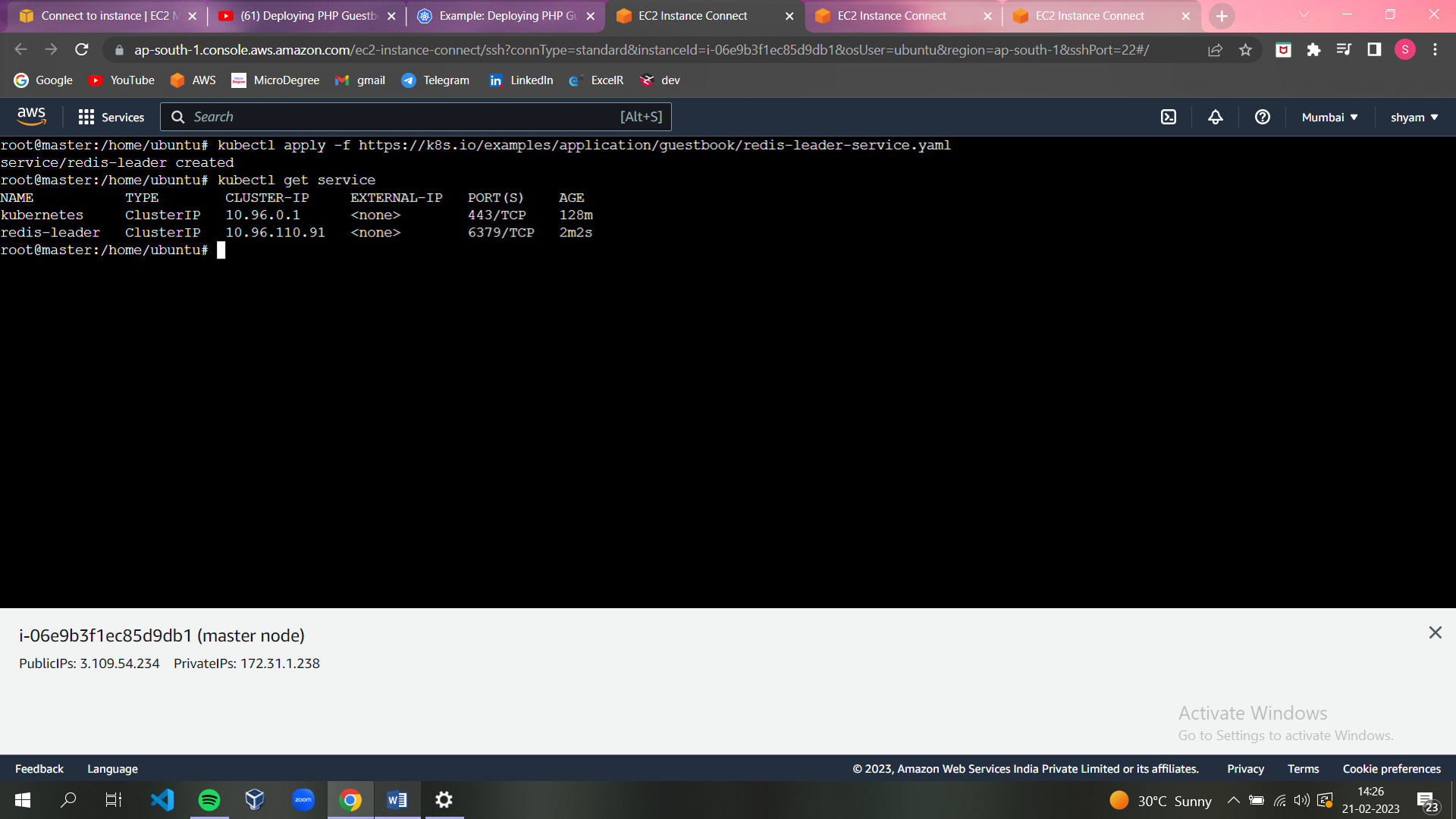
**Apply the Redis Service from the following redis-leader-service.yaml file:**

kubectl apply -f <https://k8s.io/examples/application/guestbook/redis-leader-service.yaml>



**Query the list of Services to verify that the Redis Service is running:**

kubectl get service



### STEP4- Set up Redis followers.

Although the Redis leader is a single Pod, you can make it highly available and meet traffic demands by adding a few Redis followers, or replicas.

*# SOURCE: https://cloud.google.com/kubernetes-engine/docs/tutorials/guestbook*

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: redis-follower

**labels**:

**app**: redis

**role**: follower

**tier**: backend

**spec**:

**replicas**: 2

**selector**:

**matchLabels**:

**app**: redis

**template**:

**metadata**:

**labels**:

**app**: redis

**role**: follower

**tier**: backend

**spec**:

**containers**:

- **name**: follower

**image**: gcr.io/google\_samples/gb-redis-follower:v2

**resources**:

**requests**:

**cpu**: 100m

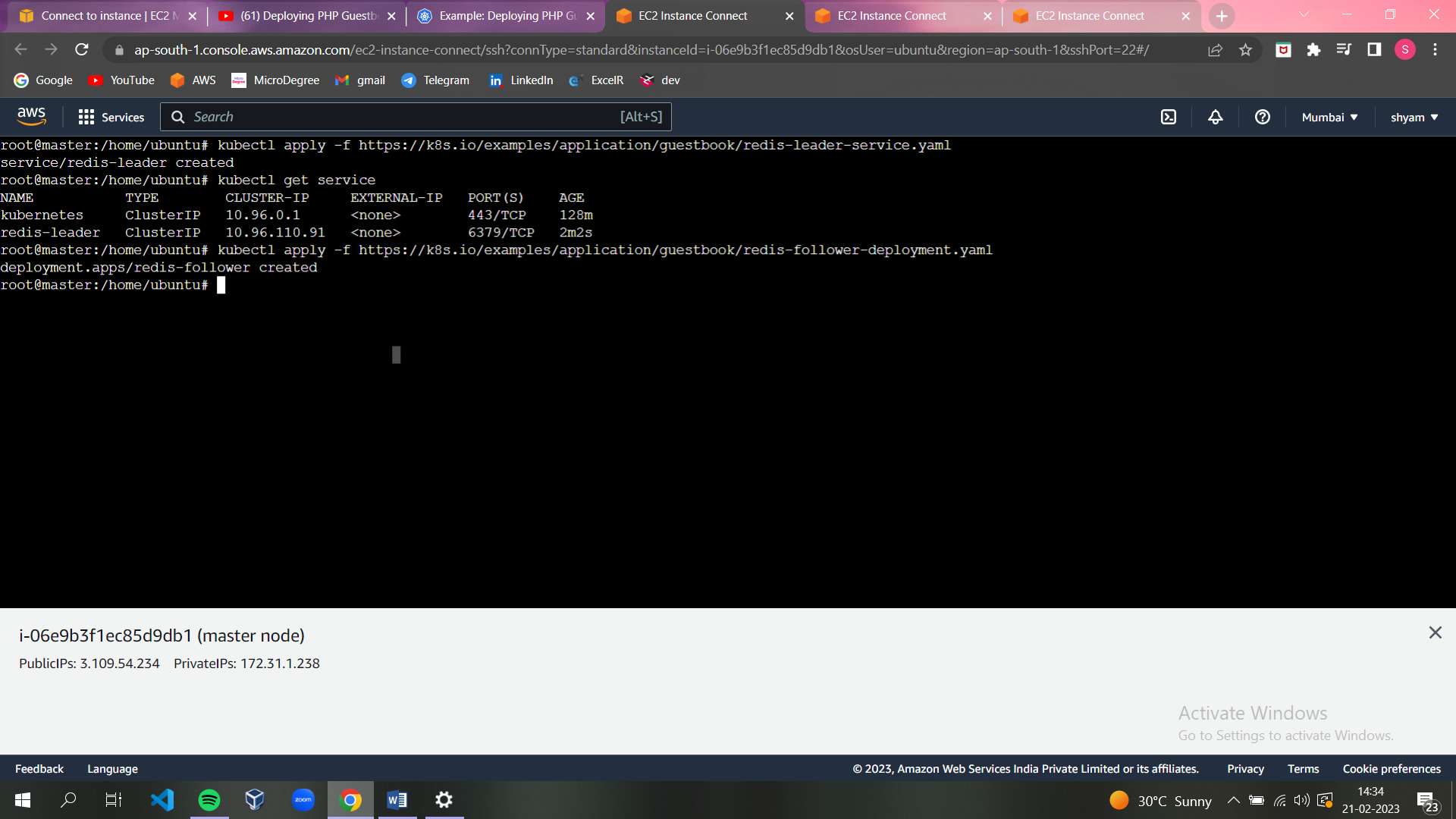
**memory**: 100Mi

**ports**:

- **containerPort**: 6379

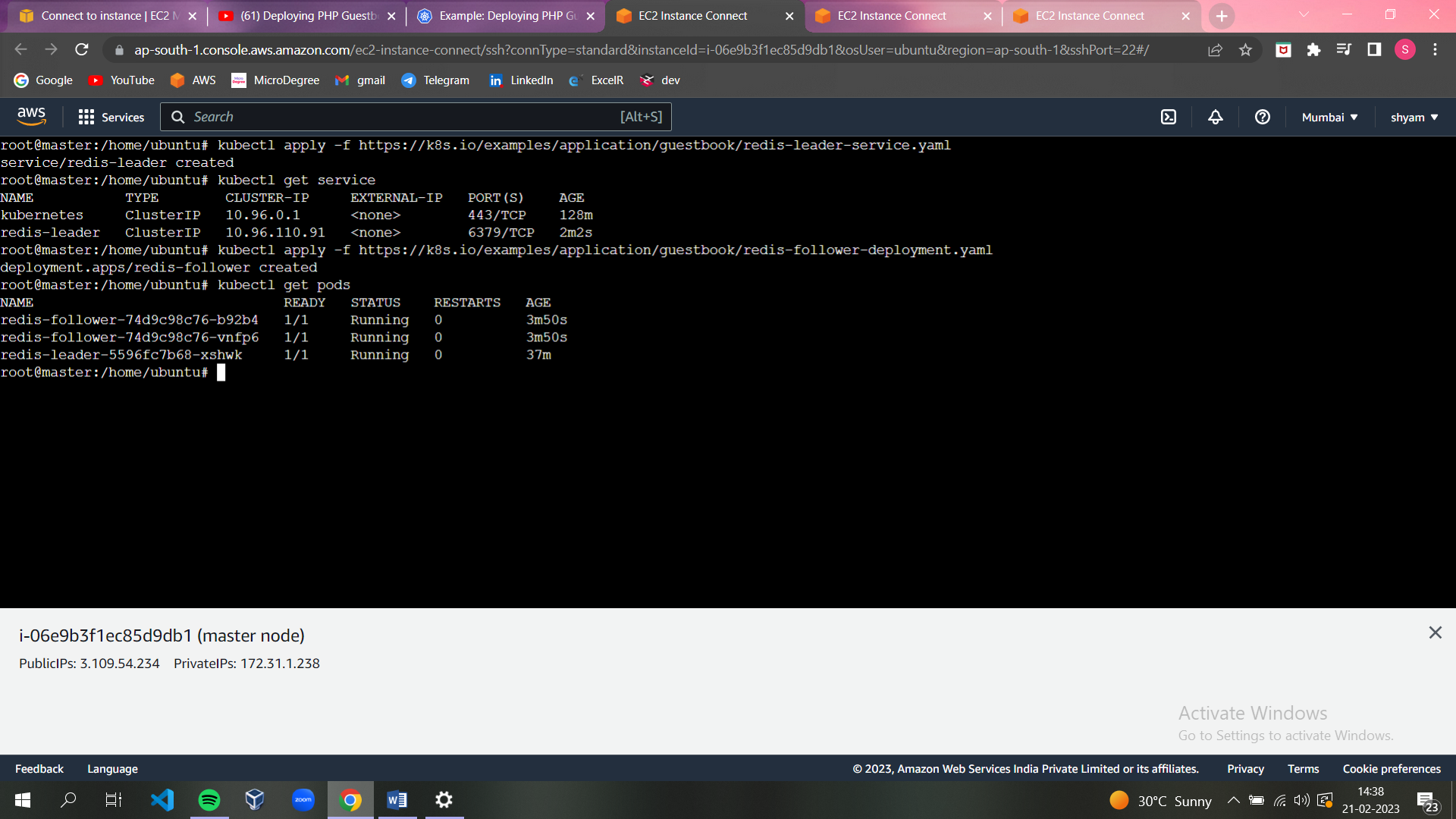
**Apply the Redis Deployment from the following redis-follower-deployment.yaml file:**

kubectl apply -f <https://k8s.io/examples/application/guestbook/redis-follower-deployment.yaml>



**Verify that the two Redis follower replicas are running by querying the list of Pods:**

kubectl get pods



### STEP5- Creating the Redis follower service.

### The guestbook application needs to communicate with the Redis followers to read data. To make the Redis followers discoverable, you must set up another [Service](https://kubernetes.io/docs/concepts/services-networking/service/).

*# SOURCE: https://cloud.google.com/kubernetes-engine/docs/tutorials/guestbook*

**apiVersion**: v1

**kind**: Service

**metadata**:

**name**: redis-follower

**labels**:

**app**: redis

**role**: follower

**tier**: backend

**spec**:

**ports**:

*# the port that this service should serve on*

- **port**: 6379

**selector**:

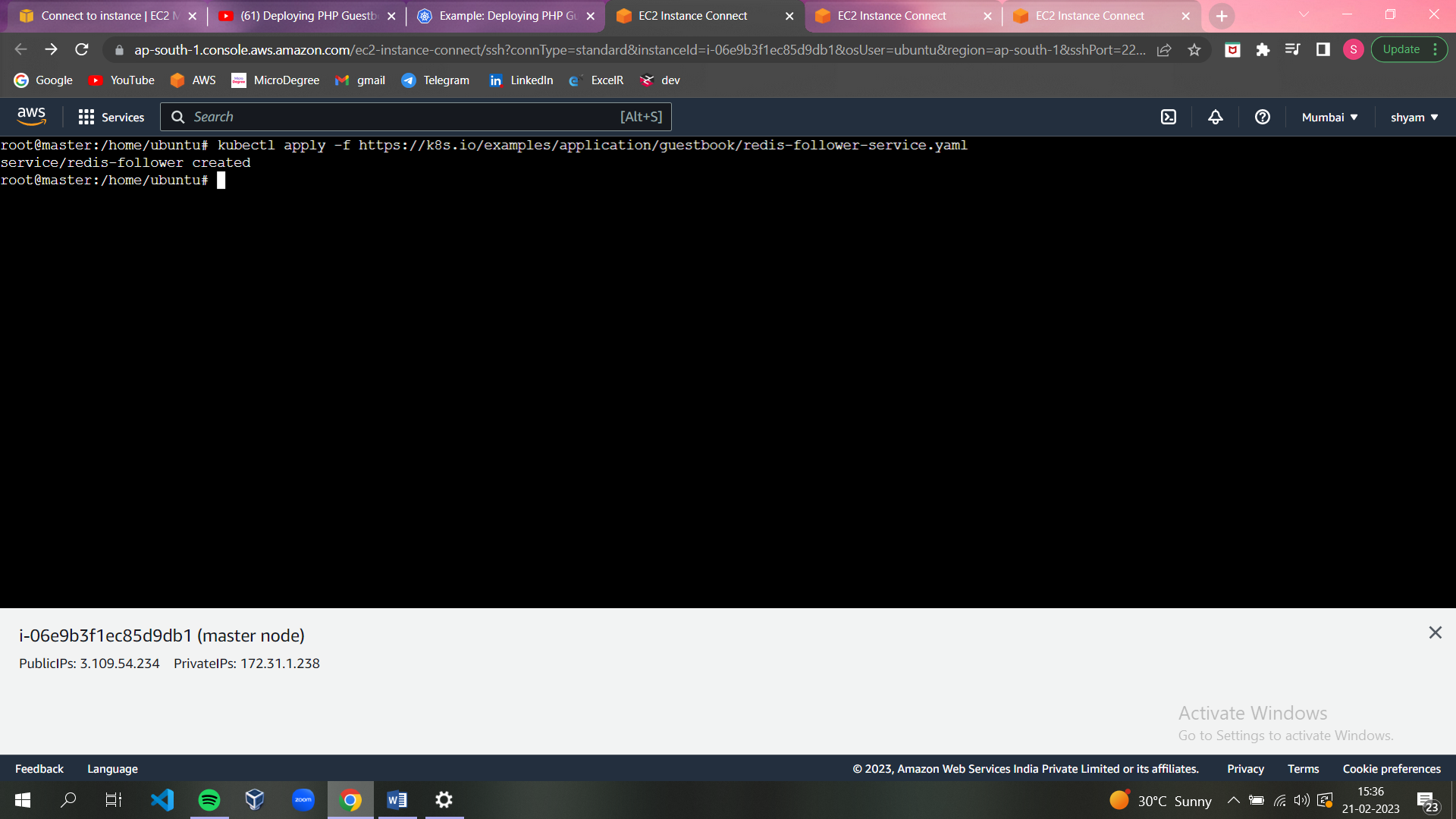
**app**: redis

**role**: follower

**tier**: backend

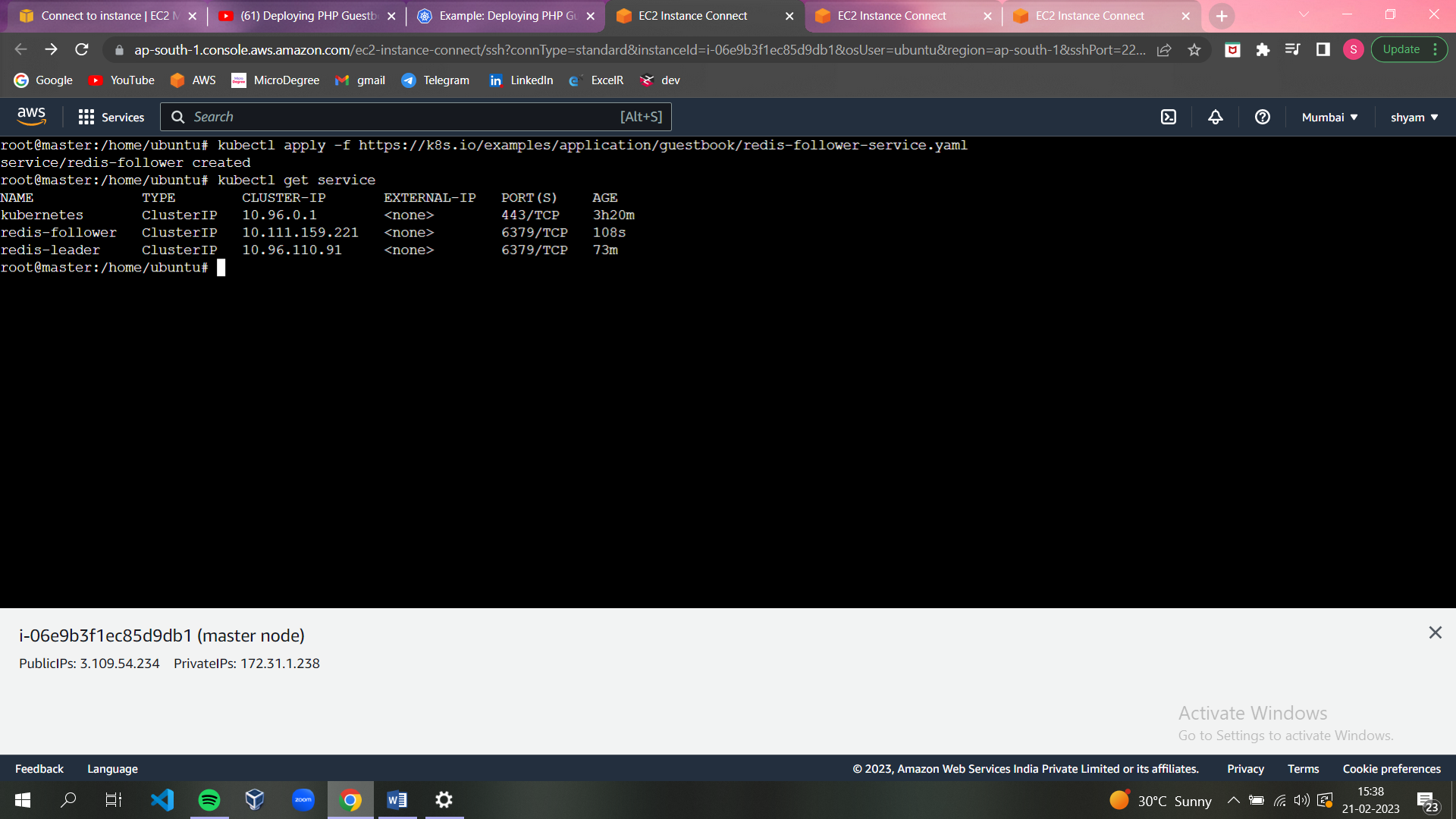
**Apply the Redis Service from the following redis-follower-service.yaml file:**

kubectl apply -f https://k8s.io/examples/application/guestbook/redis-fol



**Query the list of Services to verify that the Redis Service is running:**

kubectl get service



## **Set up and Expose the Guestbook Frontend**.

Now that we have the Redis storage of your guestbook up and running, start the guestbook web servers. Like the Redis followers, the frontend is deployed using a Kubernetes Deployment.

The guestbook app uses a PHP frontend. It is configured to communicate with either the Redis follower or leader Services, depending on whether the request is a read or a write. The frontend exposes a JSON interface, and serves a jQuery-Ajax-based UX.

### STEP6- Creating the Guestbook Frontend Deployment .

*# SOURCE: https://cloud.google.com/kubernetes-engine/docs/tutorials/guestbook*

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: frontend

**spec**:

**replicas**: 3

**selector**:

**matchLabels**:

**app**: guestbook

**tier**: frontend

**template**:

**metadata**:

**labels**:

**app**: guestbook

**tier**: frontend

**spec**:

**containers**:

- **name**: php-redis

**image**: gcr.io/google\_samples/gb-frontend:v5

**env**:

- **name**: GET\_HOSTS\_FROM

**value**: "dns"

**resources**:

**requests**:

**cpu**: 100m

**memory**: 100Mi

**ports**:

- **containerPort**: 80

**Apply the frontend Deployment from the frontend-deployment.yaml file:**

kubectl apply -f https://k8s.io/examples/application/guestbook/frontend-deployment.yaml

### 

**Query the list of Pods to verify that the three frontend replicas are running:**

kubectl get pods -l app=guestbook -l tier=frontend

### 

### STEP7- Creating the Frontend Service.

The Redis Services we applied is only accessible within the Kubernetes cluster because the default type for a Service is [ClusterIP](https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types). ClusterIP provides a single IP address for the set of Pods the Service is pointing to. This IP address is accessible only within the cluster.

If we want guests to be able to access our guestbook, we must configure the frontend Service to be externally visible, so a client can request the Service from outside the Kubernetes cluster. However a Kubernetes user can use kubectl port-forward to access the service even though it uses a ClusterIP.

*# SOURCE: https://cloud.google.com/kubernetes-engine/docs/tutorials/guestbook*

**apiVersion**: v1

**kind**: Service

**metadata**:

**name**: frontend

**labels**:

**app**: guestbook

**tier**: frontend

**spec**:

*# if your cluster supports it, uncomment the following to automatically create*

*# an external load-balanced IP for the frontend service.*

*# type: LoadBalancer*

*#type: LoadBalancer*

**ports**:

*# the port that this service should serve on*

- **port**: 80

**selector**:

**app**: guestbook

**tier**: frontend

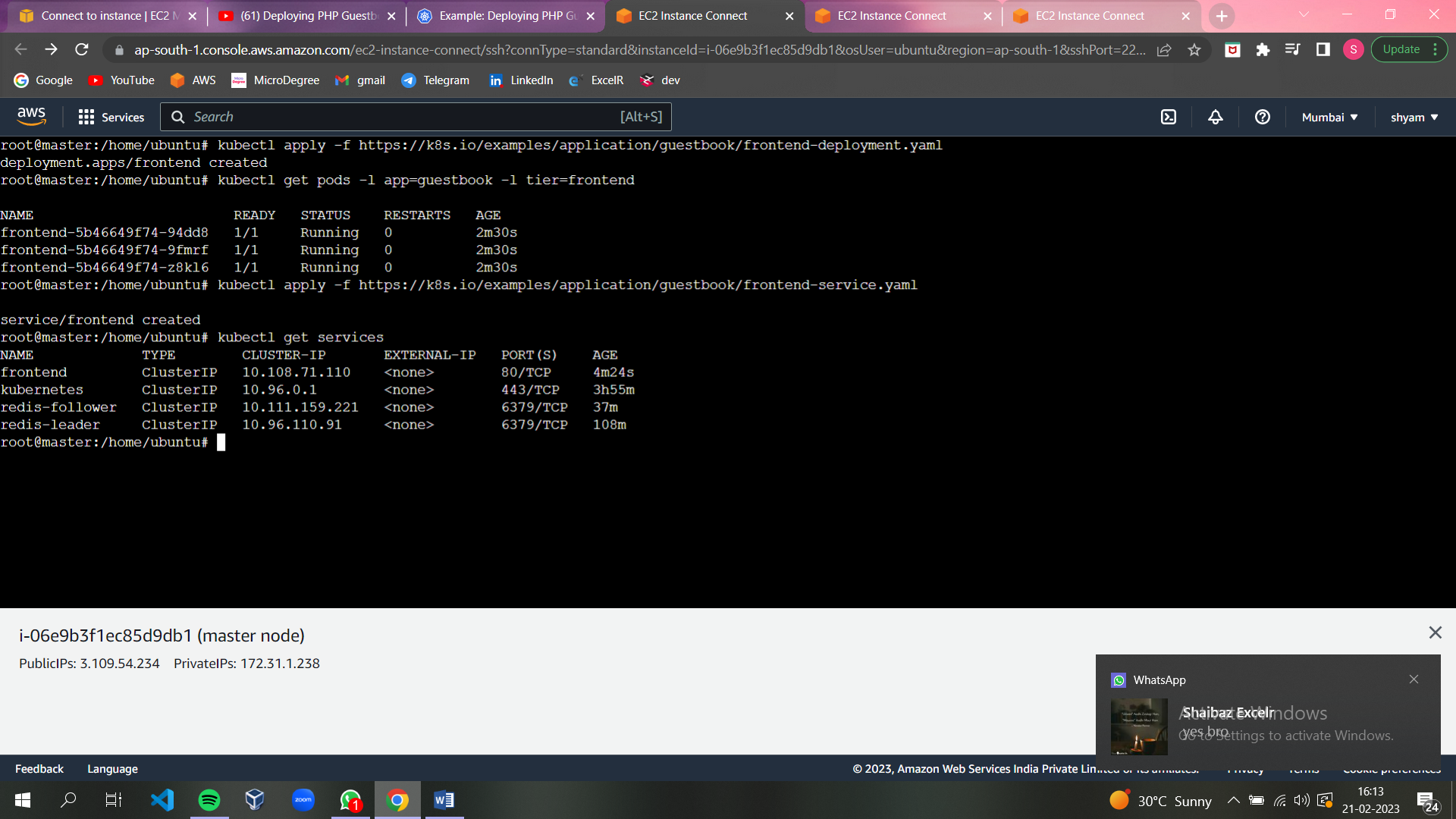
**Apply the frontend Service from the frontend-service.yaml file**:

kubectl apply -f https://k8s.io/examples/application/guestbook/frontend-service.yaml

### 

**Query the list of Services to verify that the frontend Service is running:**

kubectl get services



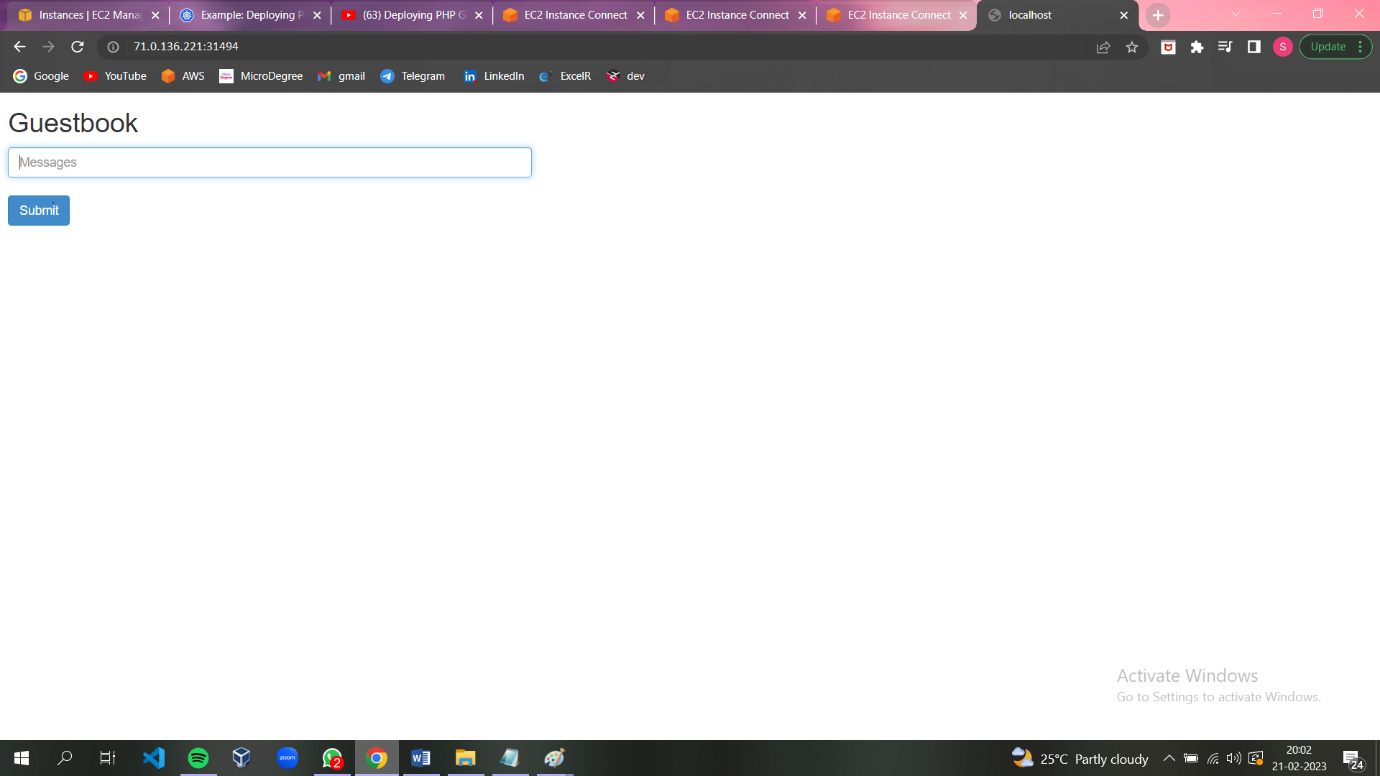
### STEP8- Viewing the Frontend Service via LoadBalancer

If you deployed the frontend-service.yaml manifest with type: LoadBalancer you need to find the IP address to view your Guestbook.

Run the following command to get the IP address for the frontend Service.

kubectl get service frontend

Copy the external IP address, and load the page in your browser to view your guestbook.

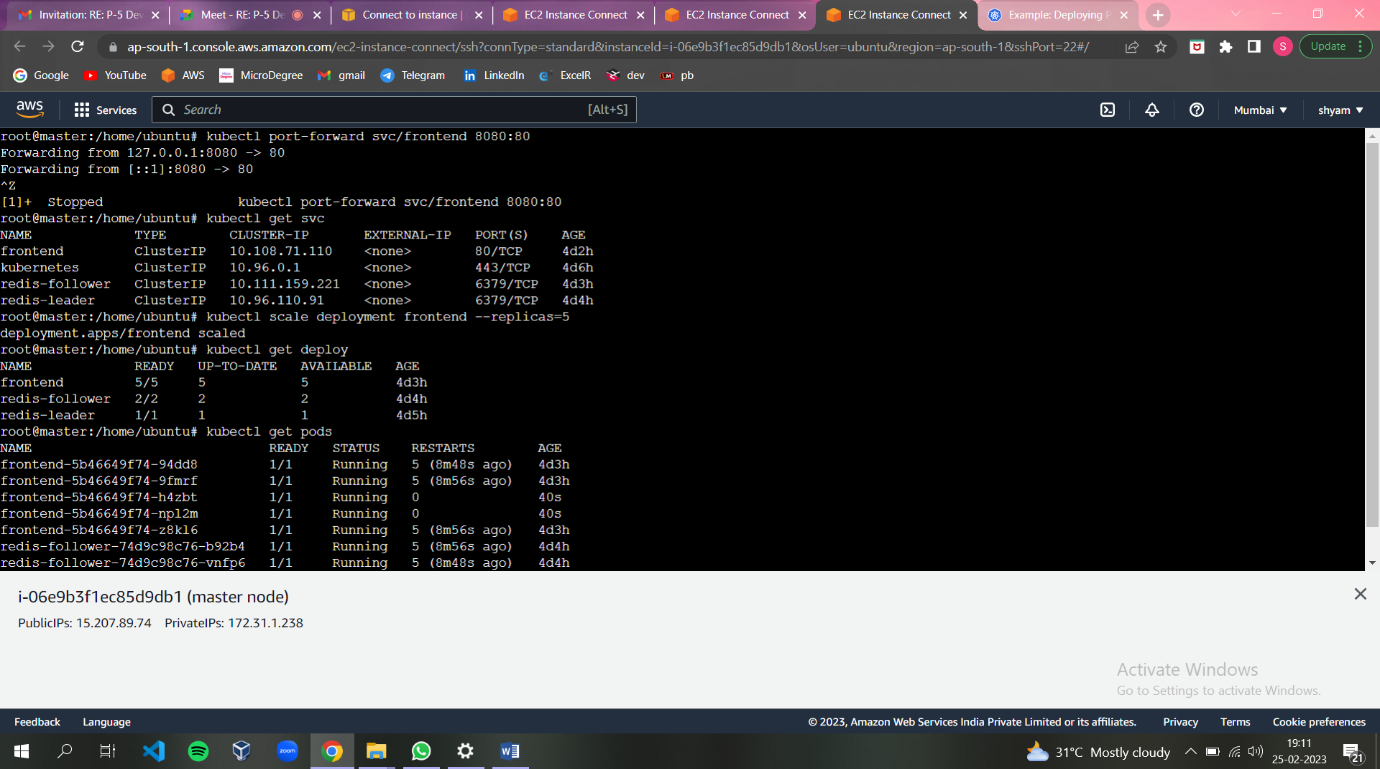


## **STEP9-** **Scale the Web Frontend.**

You can scale up or down as needed because your servers are defined as a Service that uses a Deployment controller.

Run the following command to scale up the number of frontend Pods:

kubectl scale deployment frontend --replicas=5



Query the list of Pods to verify the number of frontend Pods running:

### kubectl get pods

### 

### Run the following command to scale down the number of frontend Pods:

### kubectl scale deployment frontend --replicas=2

### 

Query the list of Pods to verify the number of frontend Pods running:

kubectl get pods

### 

## **STEP10- Cleaning up.**

Deleting the Deployments and Services also deletes any running Pods. Use labels to delete multiple resources with one command.

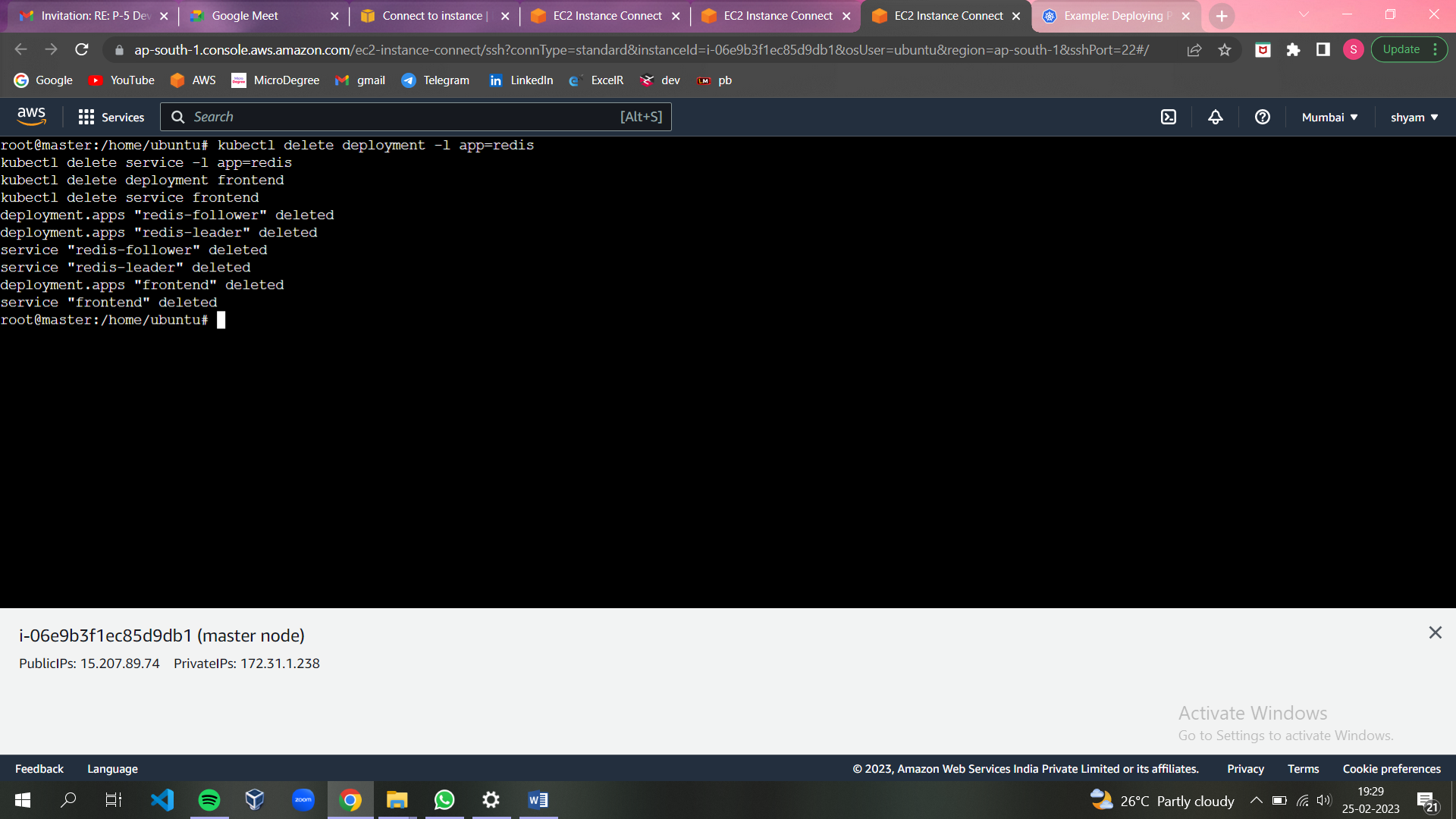
Run the following commands to delete all Pods, Deployments, and Services.

kubectl delete deployment -l app=redis

kubectl delete service -l app=redis

kubectl delete deployment frontend

kubectl delete service frontend



Query the list of Pods to verify that no Pods are running:

kubectl get pods

### 

**---------------------------------------------------------------------------------**