**Deploying of Gaming application in ACR and using it in AKS with the help of Azure pipeline**

Here we first, we convert the gaming application into a container image and upload this image to **Azure Container Registry (ACR)**. Next, we integrate **Azure Kubernetes Service (AKS)** with the ACR by **configuration YAML file** so that it can pull this image and use it to create a pod within its cluster.

Then we will get a public Load balancer IP, by using the IP we access the gamming application.

This entire process — from building the image to deploying it on AKS — is automated using **Azure Pipelines**.

**Note:** To deploy the image in AKS, which is present in the ACR, we have to configure the manifest file (YAML file). Then only we can able to create pod using image of ACR.

Let’s perform above task in practically

**Step1:** Create the Azure container registry (GamingACR).

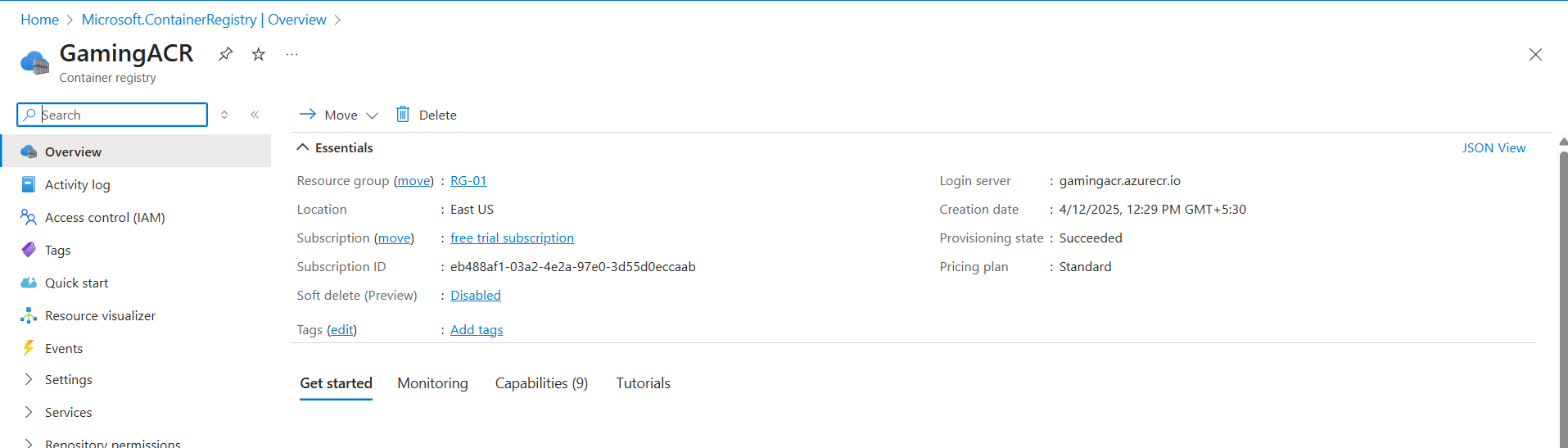
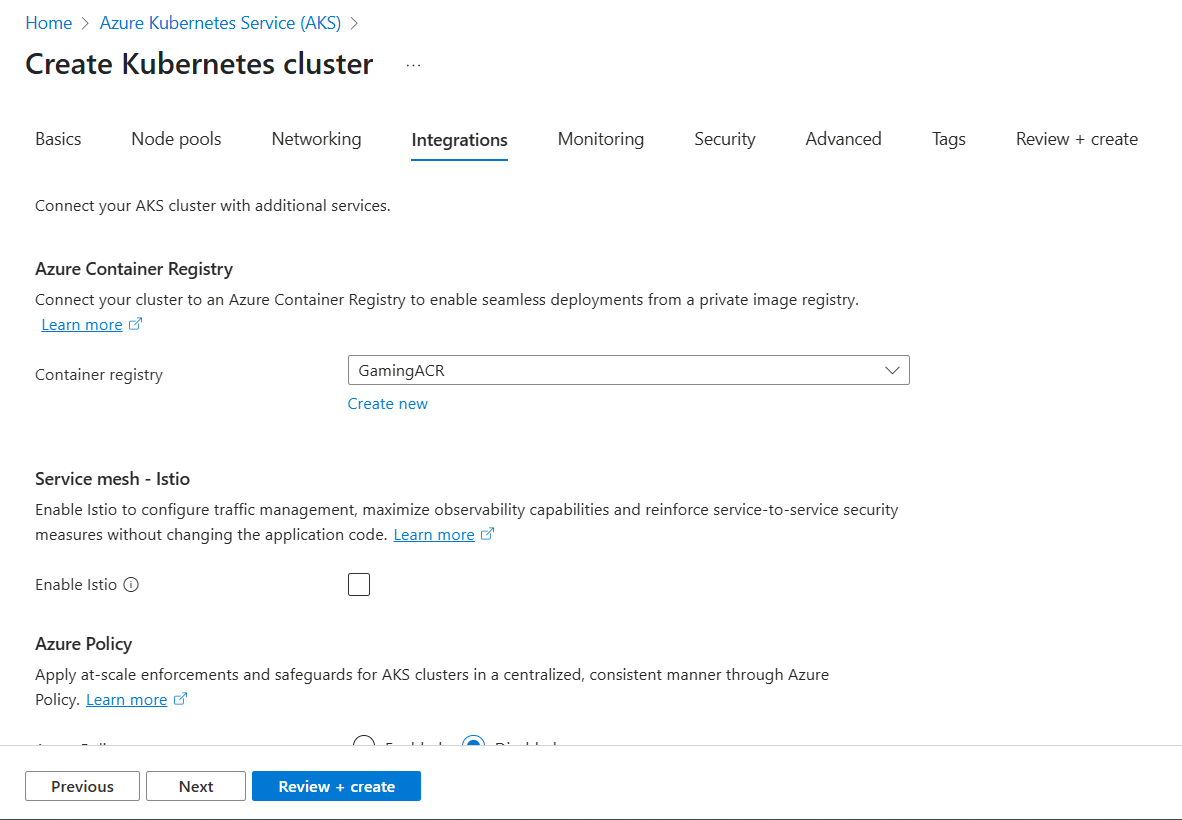


Fig: Azure container registry with name GamingACR.

**Step2:** Create a Kubernetes service (my-cluster).



**Fig:** Select the ACR while creating AKS in the integration option.

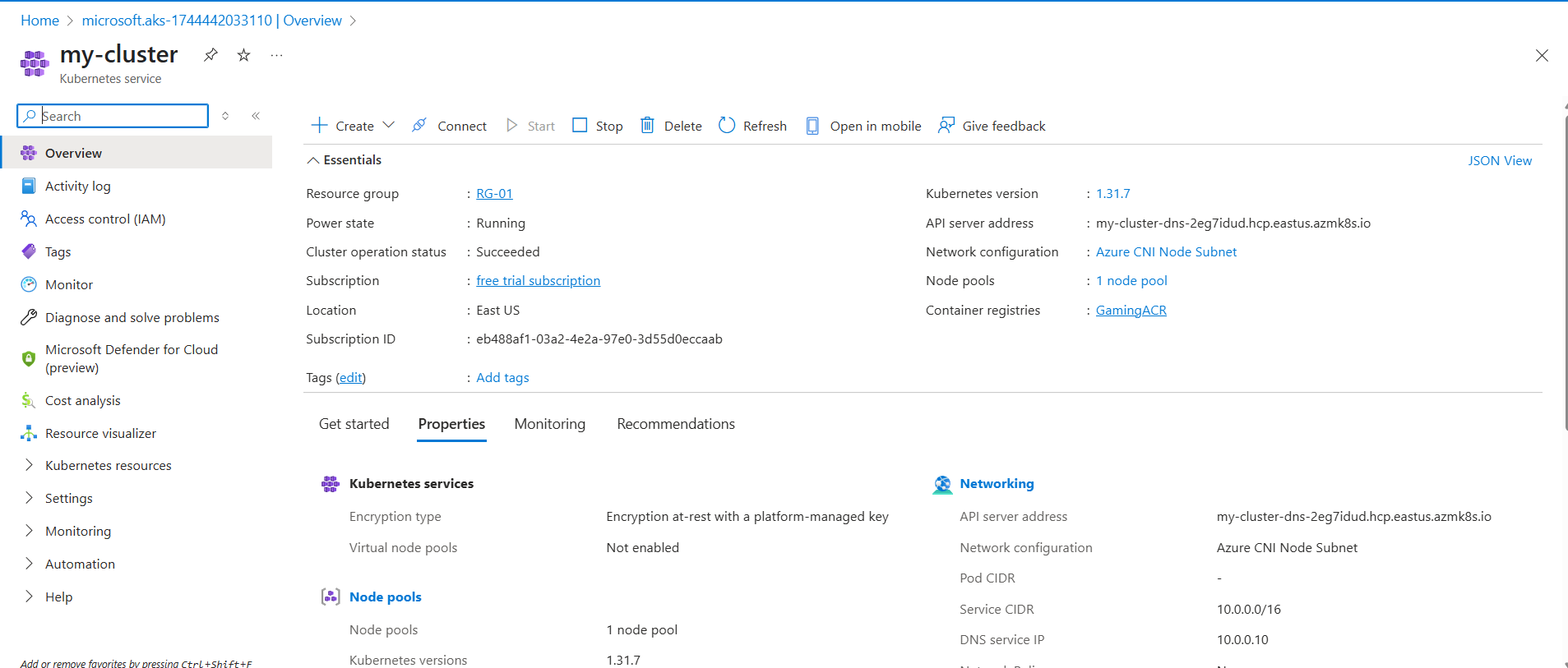


Fig: Kubernetes cluster (my-cluster) is created successfully.

**Step2:** Now push the Gaming application code to the azure repository (gaming-repo) in on organization of my-first-organization1 of project (project-02) repository (gaming-repo).

Code Source: <https://github.com/sheriharish/GammingAPP-AKS>

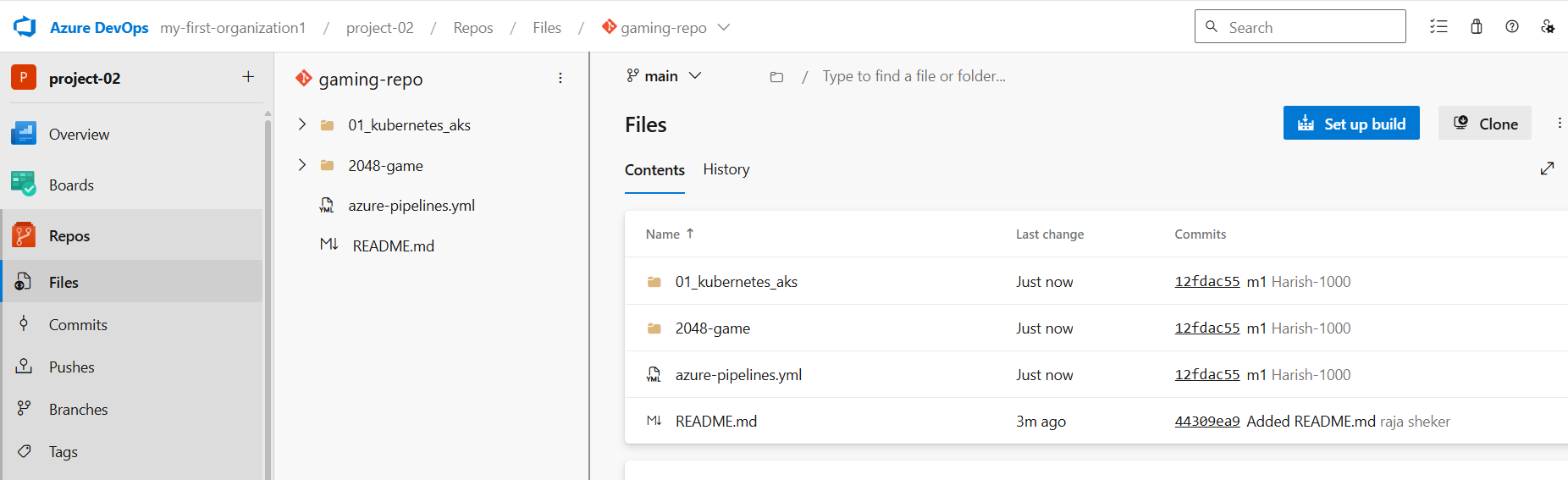


Fig: gaming application code is successfully pushed into Azure repository (gaming-repo).

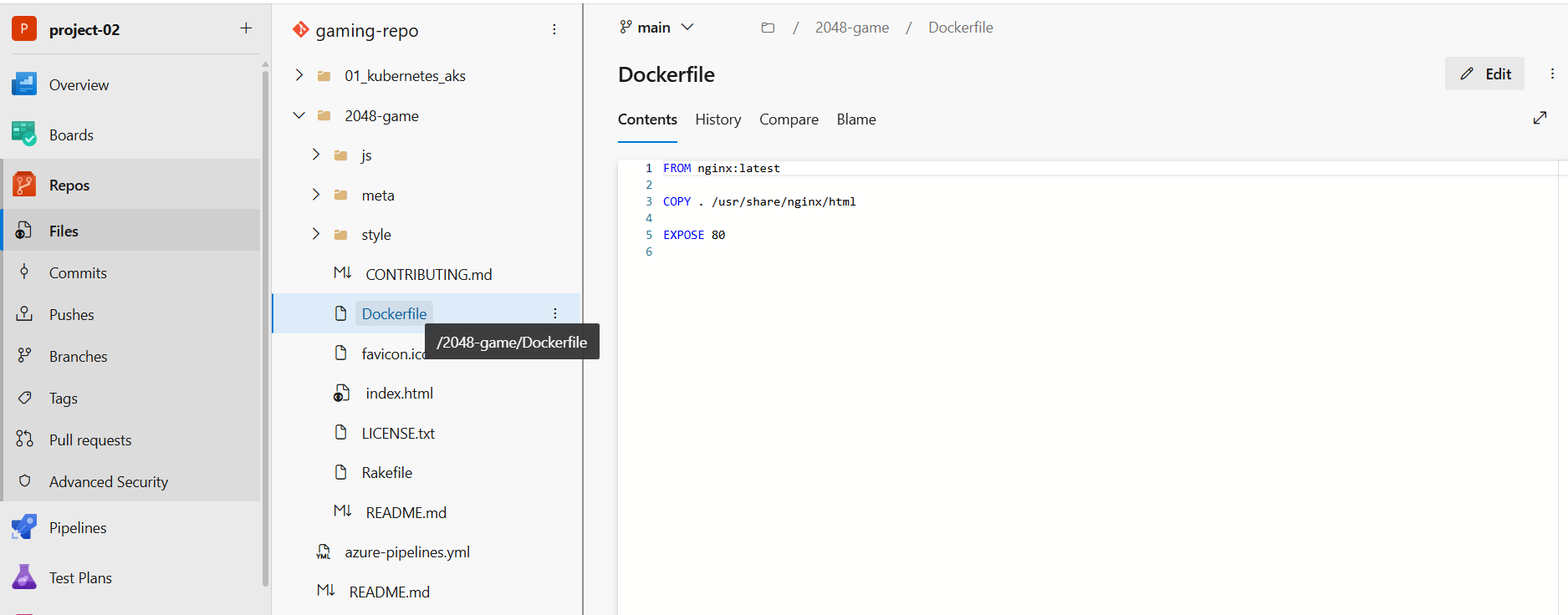


Fig: Docker file.

This **Docker file** to convert the gaming application code into image.

And the **azure-pipeline.yaml** file is used to configure and push the created image into the ACR (GamingACR).

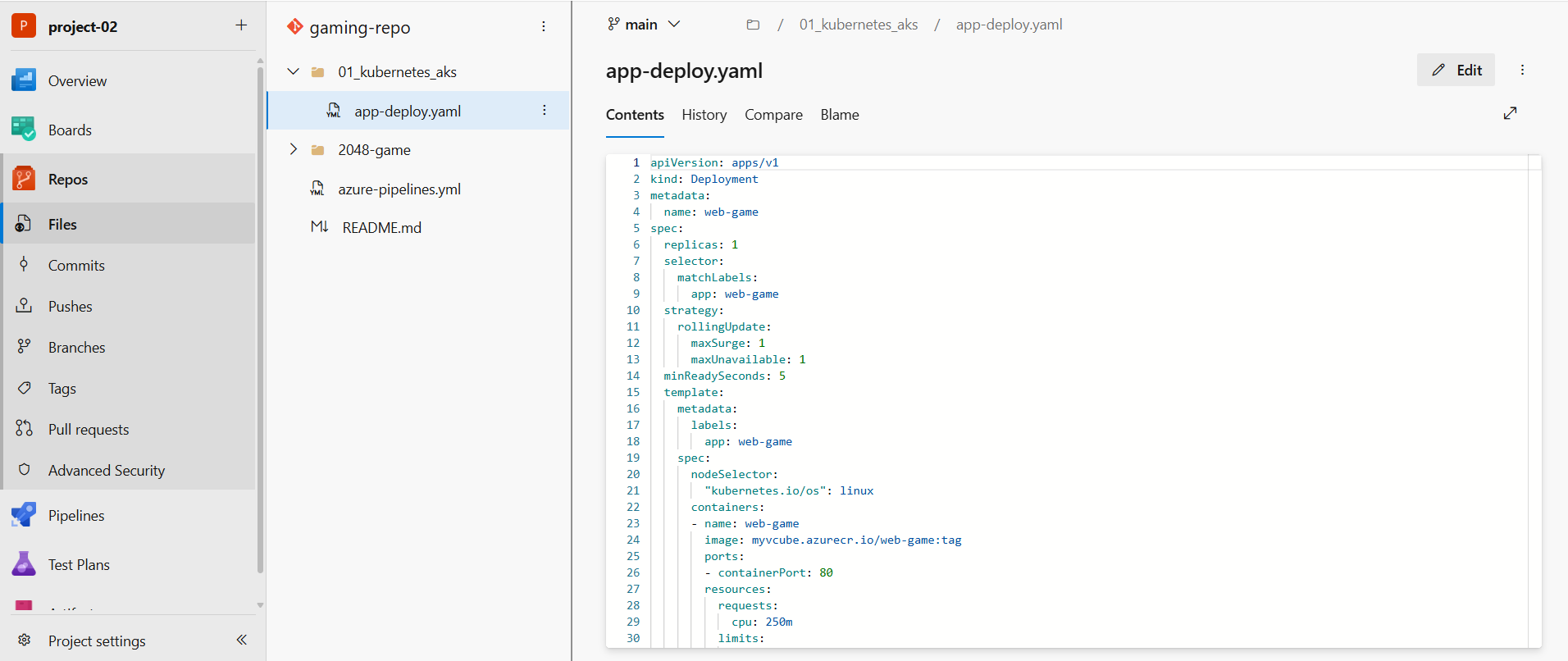


Fig: app-deployment.yaml file.

This **app-deployment.yaml** file is used to integrate the ACR with the AKS in order to configure and create the pod within the cluster.

**Note:** Configure the YAML files (azure-pipeline.yaml & app-deploy.yaml) accounting to your subscription and namings as showing in the below one example.

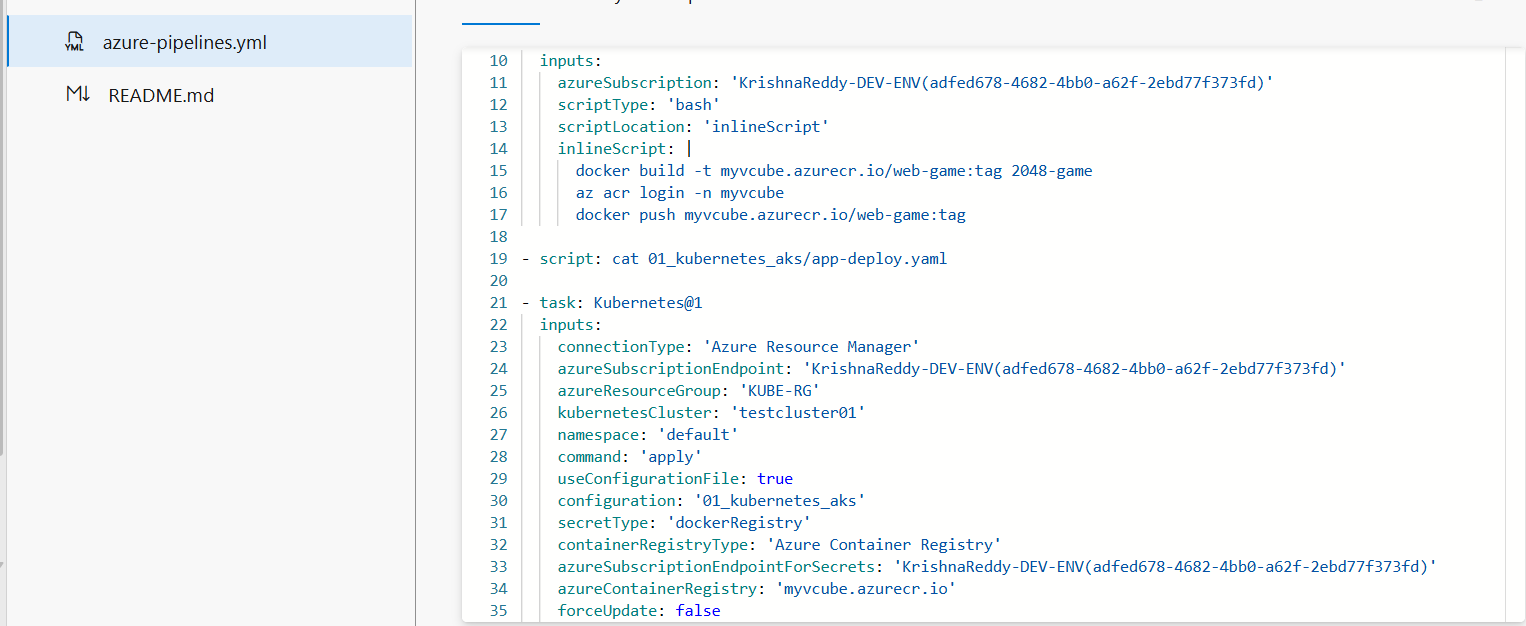


Fig: Change the above content or values according to your subscription and names of ACR and AKS.

**InlineScript:**

docker build -t gamingacr.azurecr.io/web-game:tag 2048-game

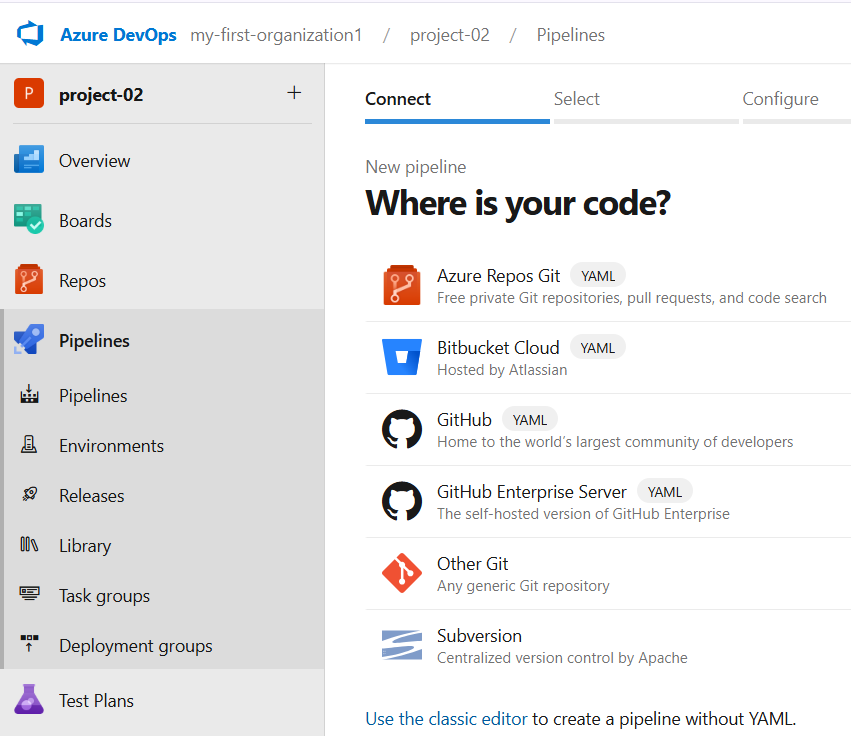
az acr login -n gamingacr.azurecr.io

docker push gamingacr.azurecr.io/web-game:tag

**Step3:** Now perform the CI and CD using the single YAML file (azue-pipeline.yaml).

**Note:** Here instead of using classic editor we are using the “**Azure Repos Git (yaml)**”. That main benefit of using this feature is by configuring a single YAML file we can perform both CI (pipeline) and CD (Release).

Go To🡪Organization (my-first-organization)🡪project (project-02)🡪pipelines🡪New pipeline🡪Azure Repos Git (yaml)🡪Repository (gaming-repo)🡪Save and Run.



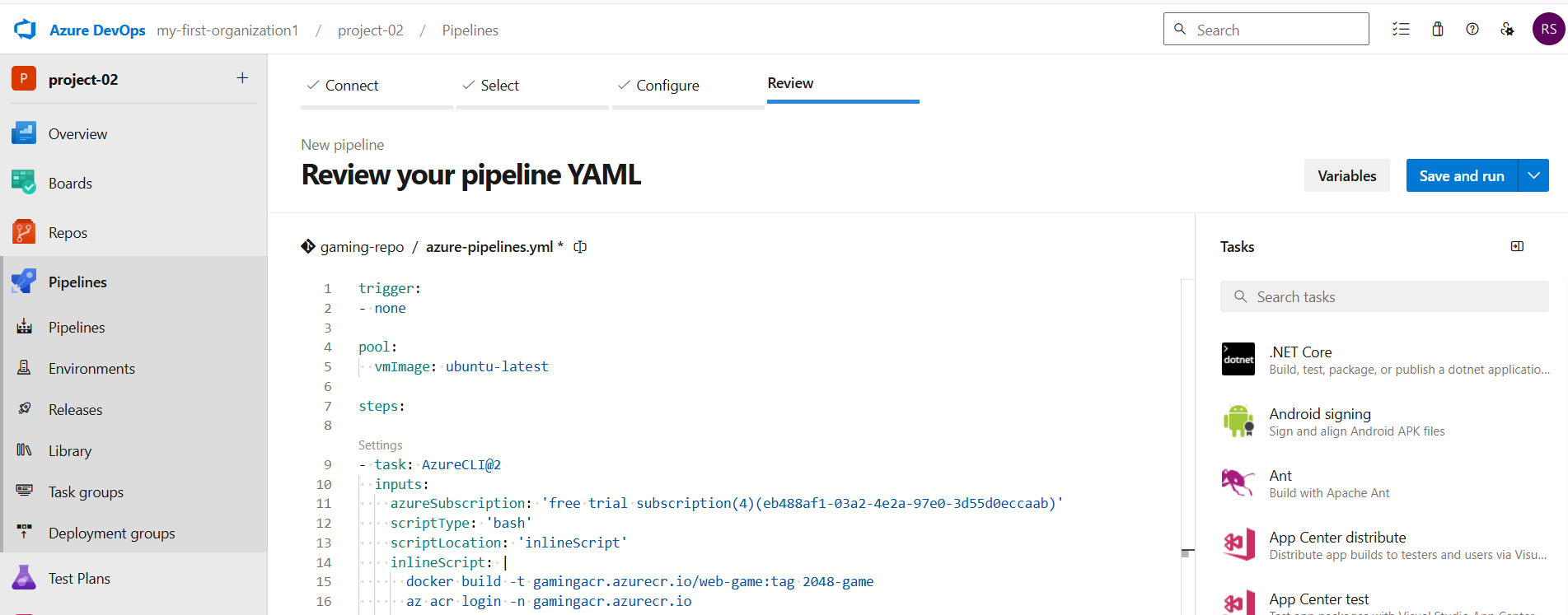
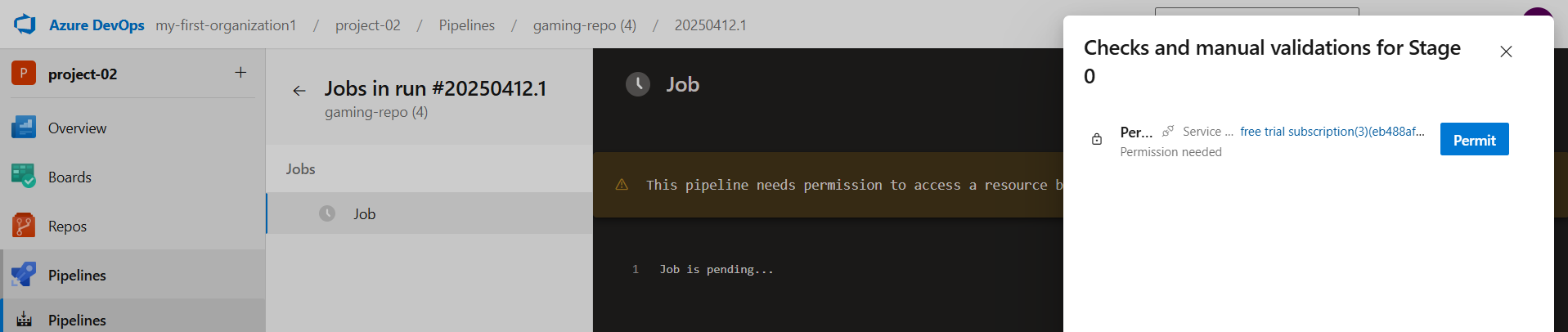


Fig: Run the YAML file.

After pressing **Save and Run**, the jobs will run automatically, but it will sometimes ask for permission to access the resource so for this just press **permit** as shown below figure.



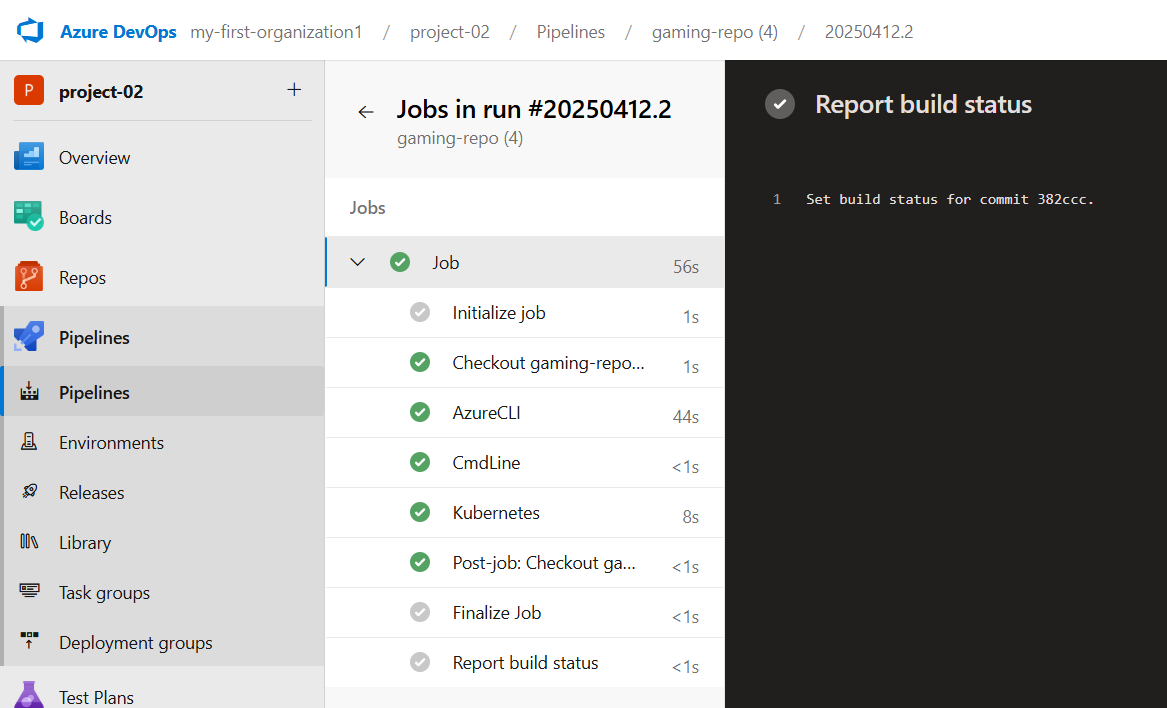


Fig: All Jobs are executed successfully.

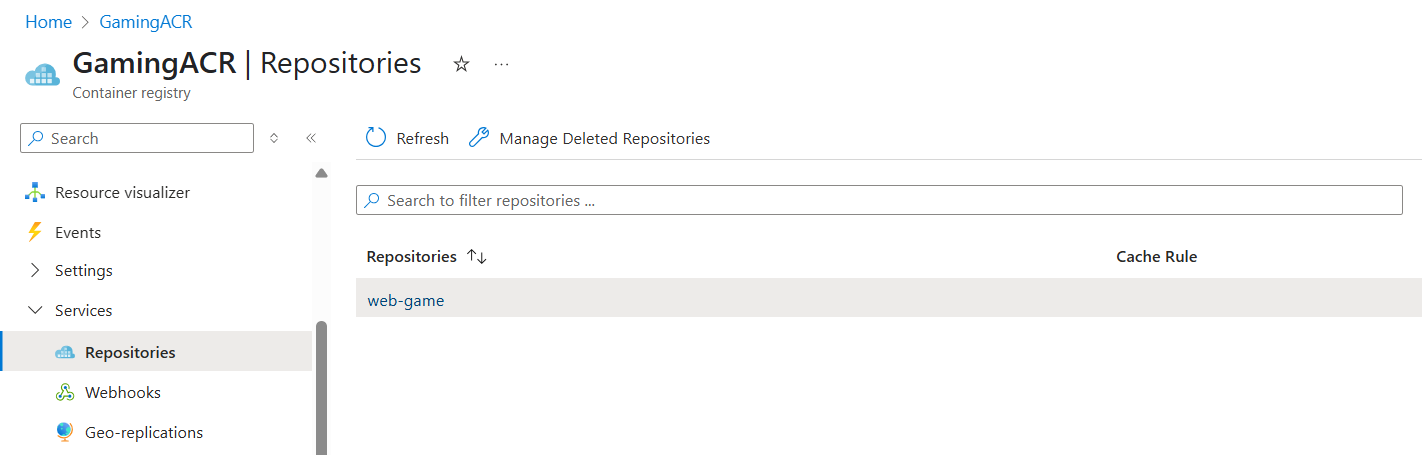


Fig: Image (web-game) is uploaded successfully into the ACR (GamingACR).

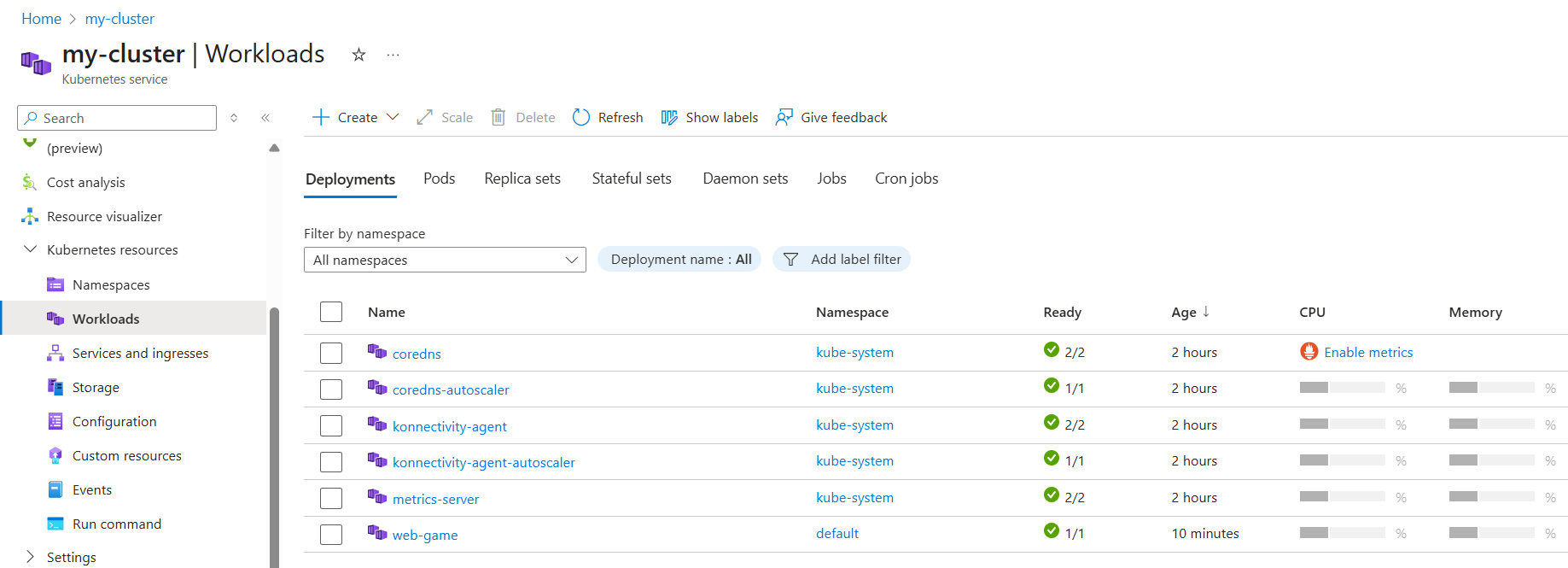


Fig: Worker node (web-game) is created successfully in the cluster (my-cluster).

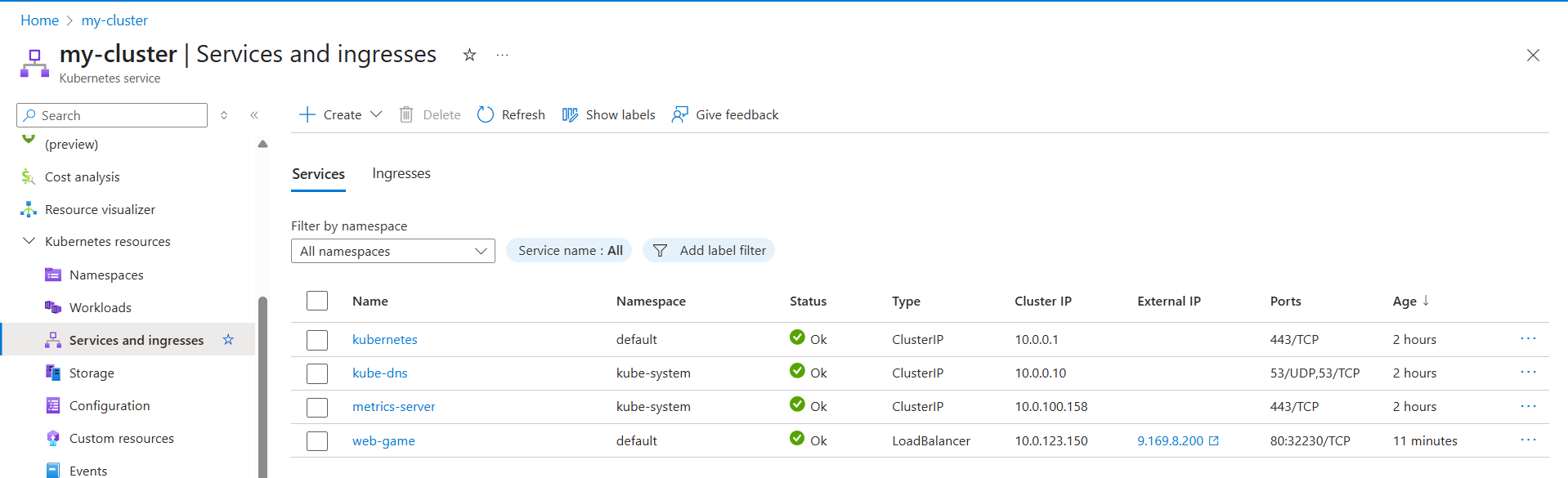


Fig: An external service is created with public IP.

By copying the public IP and brows it in any browser to check whether the gaming application live or not.

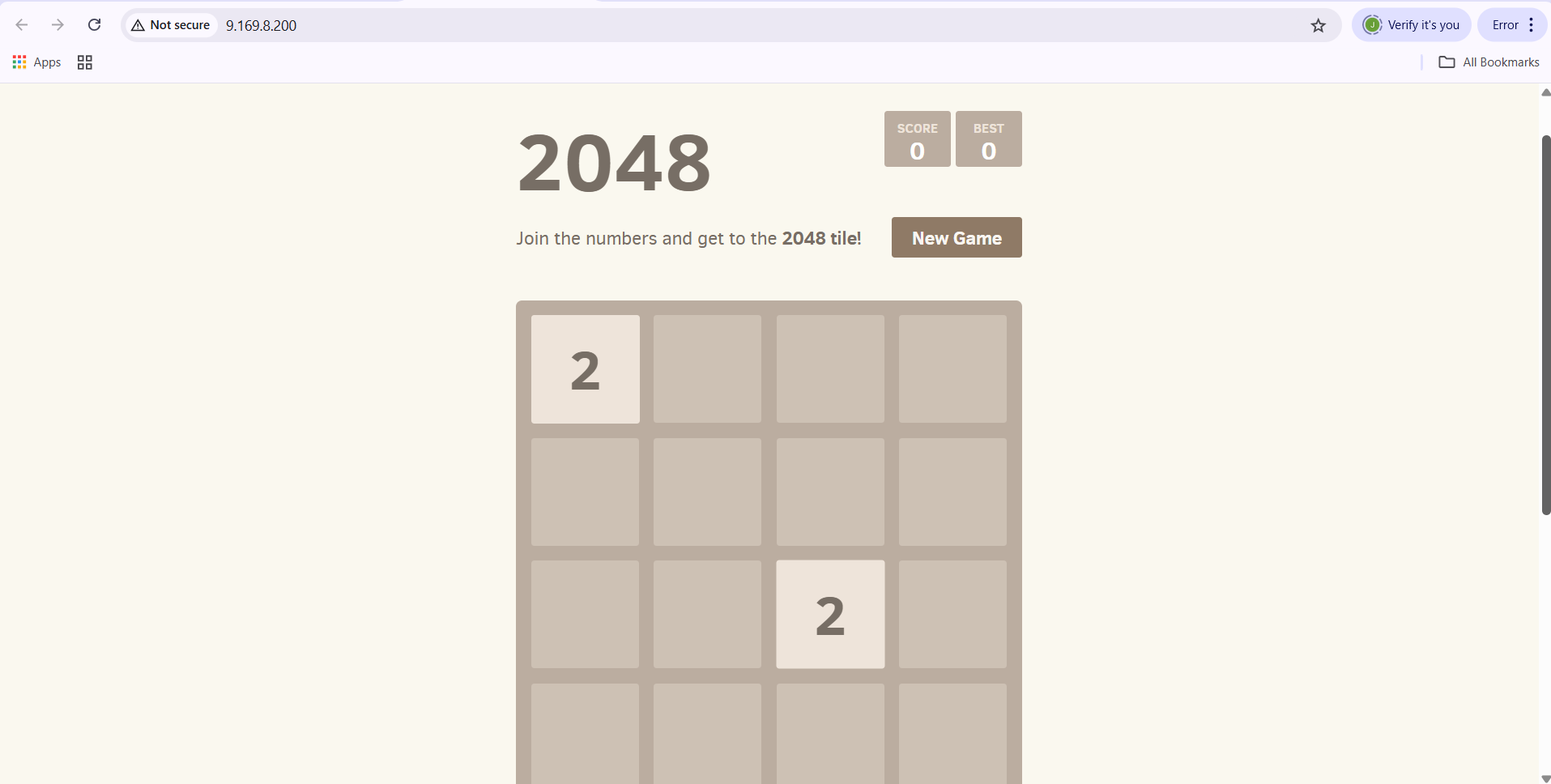


Fig: Gaming application is up and running.

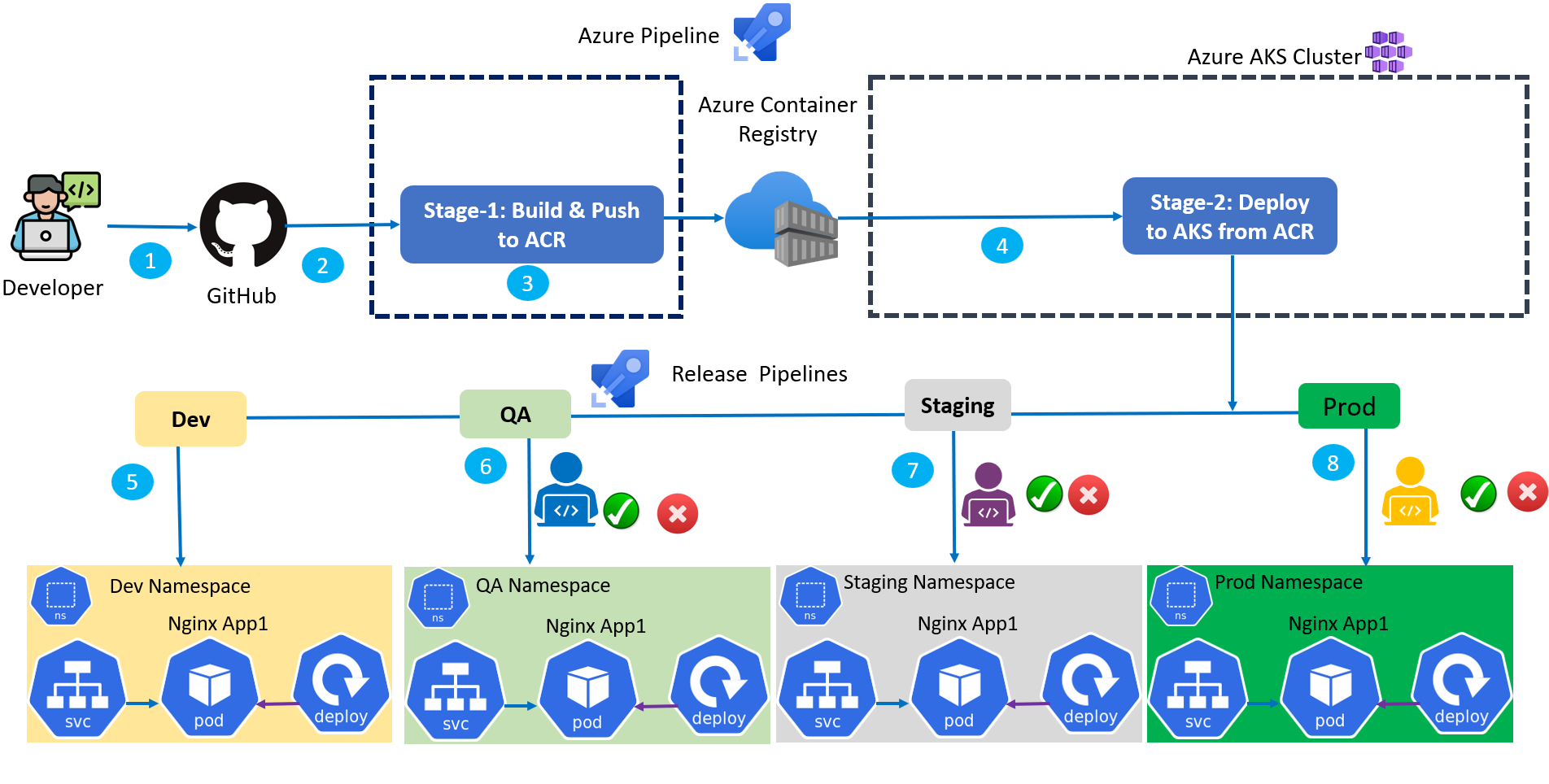


Fig: Automating Deployments on Azure Kubernetes Service (AKS) with Azure Pipelines!