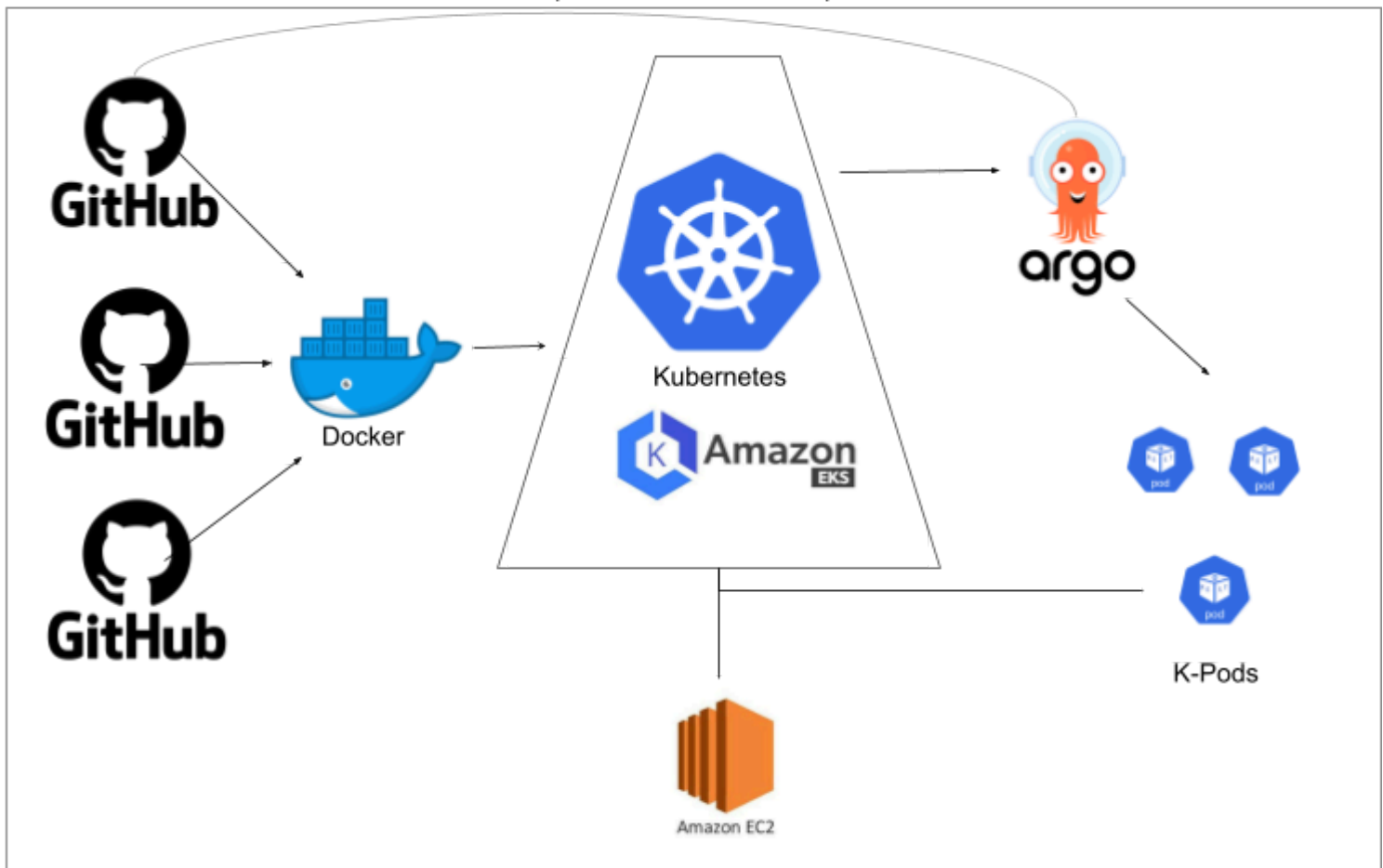


NAME -: RISHIT SUMAN

EMAIL-: [rishitsuman1@gmail.com](mailto:rishitsuman1@gmail.com)

MOB.NO. -: +917856023393

**GITOPS CI/CD PIPELINE WITH ARGO-CD ROLLOUT**  
(GAME DEPLOYMENT)



## Task 1: Setup and Configuration:

### 1. Create a GitRepository

Created a new GitHub repository named "tetriswa" to store the source code of our web application. This repository will be the central location for managing our application code and configurations.

Repository link: <https://github.com/RISHIT2070/tetriswa>

### 2. Install Argo CD on Your Kubernetes Cluster

Installed Argo CD on our AWS EKS cluster by following the official documentation. Argo CD provides a declarative way to manage Kubernetes resources through GitOps practices. It continuously monitors the Git repository for changes and automatically applies them to the cluster.

### 3. Install Argo Rollouts

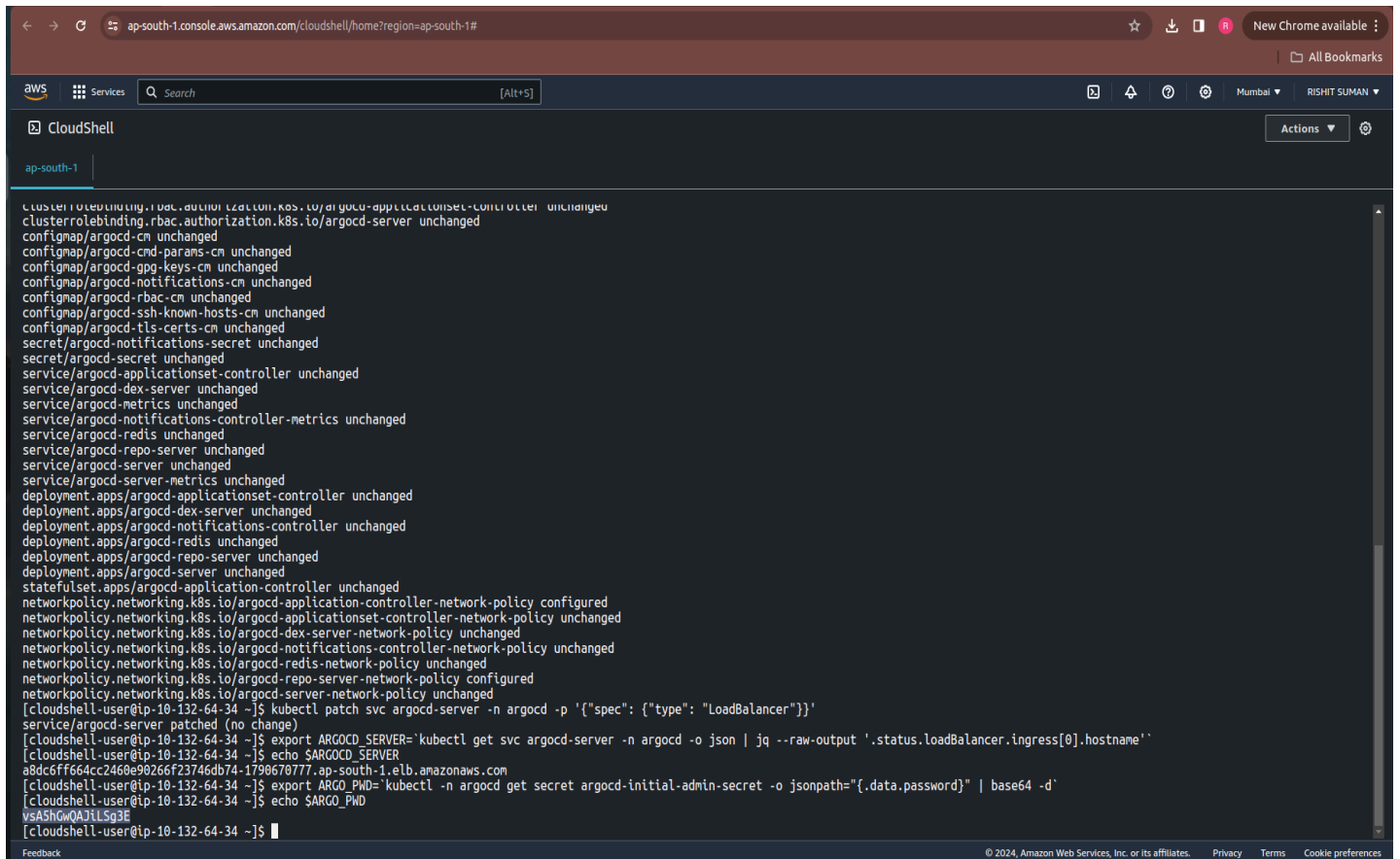
Installed the Argo Rollouts controller in our AWS EKS cluster using the provided installation guide. Argo Rollouts extends Argo CD by adding support for advanced deployment strategies such as canary releases and blue-green deployments. It allows us to manage the rollout process of our application in a controlled and automated manner.

## #Created AWS EKS Cluster Named "Gitops" with 2 Nodes -:

The screenshot displays the AWS Management Console for an Amazon Elastic Kubernetes Service (EKS) cluster named "Gitops". The console is in the "ap-south-1" region. The left sidebar shows the "Amazon Elastic Kubernetes Service" navigation menu with options like "Clusters", "Amazon EKS Anywhere", and "Related services". The main content area shows the "Gitops" cluster details, including its status (Active), Kubernetes version (1.29), support period (Standard support until March 23, 2025), and provider (EKS). Below this, the "Resources" tab is selected, showing a list of nodes. The nodes are listed in a table with columns for Node name, Instance type, Node group, Created, and Status. Two nodes are shown, both with a status of "Ready".

Node name	Instance type	Node group	Created	Status
ip-172-31-0-35.ap-south-1.compute.internal	t3.medium	nodes	Created 26 minutes ago	Ready
ip-172-31-46-76.ap-south-1.compute.internal	t3.medium	nodes	Created 26 minutes ago	Ready

## #Installing ARGO CD & ARGO ROLLOUT On AWS CLI



```
< → ↻ ap-south-1.console.aws.amazon.com/cloudshell/home?region=ap-south-1# ☆ ⬇ 📄 R New Chrome available ⋮
All Bookmarks

AWS Services 🔍 Search [Alt+S] ⓘ ⚙️ 🔄 📄 Mumbai RISHIT SUMAN ▾

CloudShell Actions ⌵ ⚙️

ap-south-1

clusterrolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller unchanged
clusterrolebinding.rbac.authorization.k8s.io/argocd-server unchanged
configmap/argocd-cm unchanged
configmap/argocd-cmd-params-cm unchanged
configmap/argocd-gpg-keys-cm unchanged
configmap/argocd-notifications-cm unchanged
configmap/argocd-rbac-cm unchanged
configmap/argocd-ssh-known-hosts-cm unchanged
configmap/argocd-tls-certs-cm unchanged
secret/argocd-notifications-secret unchanged
secret/argocd-secret unchanged
service/argocd-applicationset-controller unchanged
service/argocd-dex-server unchanged
service/argocd-metrics unchanged
service/argocd-notifications-controller-metrics unchanged
service/argocd-redis unchanged
service/argocd-repo-server unchanged
service/argocd-server unchanged
service/argocd-server-metrics unchanged
deployment.apps/argocd-applicationset-controller unchanged
deployment.apps/argocd-dex-server unchanged
deployment.apps/argocd-notifications-controller unchanged
deployment.apps/argocd-redis unchanged
deployment.apps/argocd-repo-server unchanged
deployment.apps/argocd-server unchanged
statefulset.apps/argocd-application-controller unchanged
networkpolicy.networking.k8s.io/argocd-application-controller-network-policy configured
networkpolicy.networking.k8s.io/argocd-applicationset-controller-network-policy unchanged
networkpolicy.networking.k8s.io/argocd-dex-server-network-policy unchanged
networkpolicy.networking.k8s.io/argocd-notifications-controller-network-policy unchanged
networkpolicy.networking.k8s.io/argocd-redis-network-policy unchanged
networkpolicy.networking.k8s.io/argocd-repo-server-network-policy configured
networkpolicy.networking.k8s.io/argocd-server-network-policy unchanged
[cloudshell-user@ip-10-132-64-34 ~]$ kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
service/argocd-server patched (no change)
[cloudshell-user@ip-10-132-64-34 ~]$ export ARGOCD_SERVER='kubectl get svc argocd-server -n argocd -o json | jq --raw-output '.status.loadBalancer.ingress[0].hostname''
[cloudshell-user@ip-10-132-64-34 ~]$ echo $ARGOCD_SERVER
a8dc6ff664cc2460e90266f23746db74-1798670777.ap-south-1.elb.amazonaws.com
[cloudshell-user@ip-10-132-64-34 ~]$ export ARGO_PWD='kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d'
[cloudshell-user@ip-10-132-64-34 ~]$ echo $ARGO_PWD
vsAShGwQAJiLSg3E
[cloudshell-user@ip-10-132-64-34 ~]$
```

## Task 2: Creating the GitOps Pipeline

### 1. Dockerize the Application

#Dockerized our simple web application by creating a Dockerfile. This file contains instructions for building a Docker image that encapsulates our application code and dependencies.

#Built the Docker image using the docker build command and tagged it with a version number.

#Pushed the Docker image to a public container registry (in this case, Docker Hub) to make it accessible to our AWS EKS cluster.

## 2. Deploy the Application Using Argo CD

#Updated the Kubernetes manifests (Deployment and Service) in our GitHub repository to reference the Docker image we pushed to Docker Hub. These manifests define how our application should be deployed and exposed within the AWS EKS cluster.

#Configured Argo CD to watch our GitHub repository for changes. When changes are detected, Argo CD automatically synchronizes the cluster's state with the desired state defined in the repository, ensuring that our application is deployed and updated accordingly.

### Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tetris-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tetris
  template:
    metadata:
      labels:
        app: tetris
    spec:
      containers:
        - name: tetris
          image: nasi101/tetris
          ports:
            - containerPort: 80
```

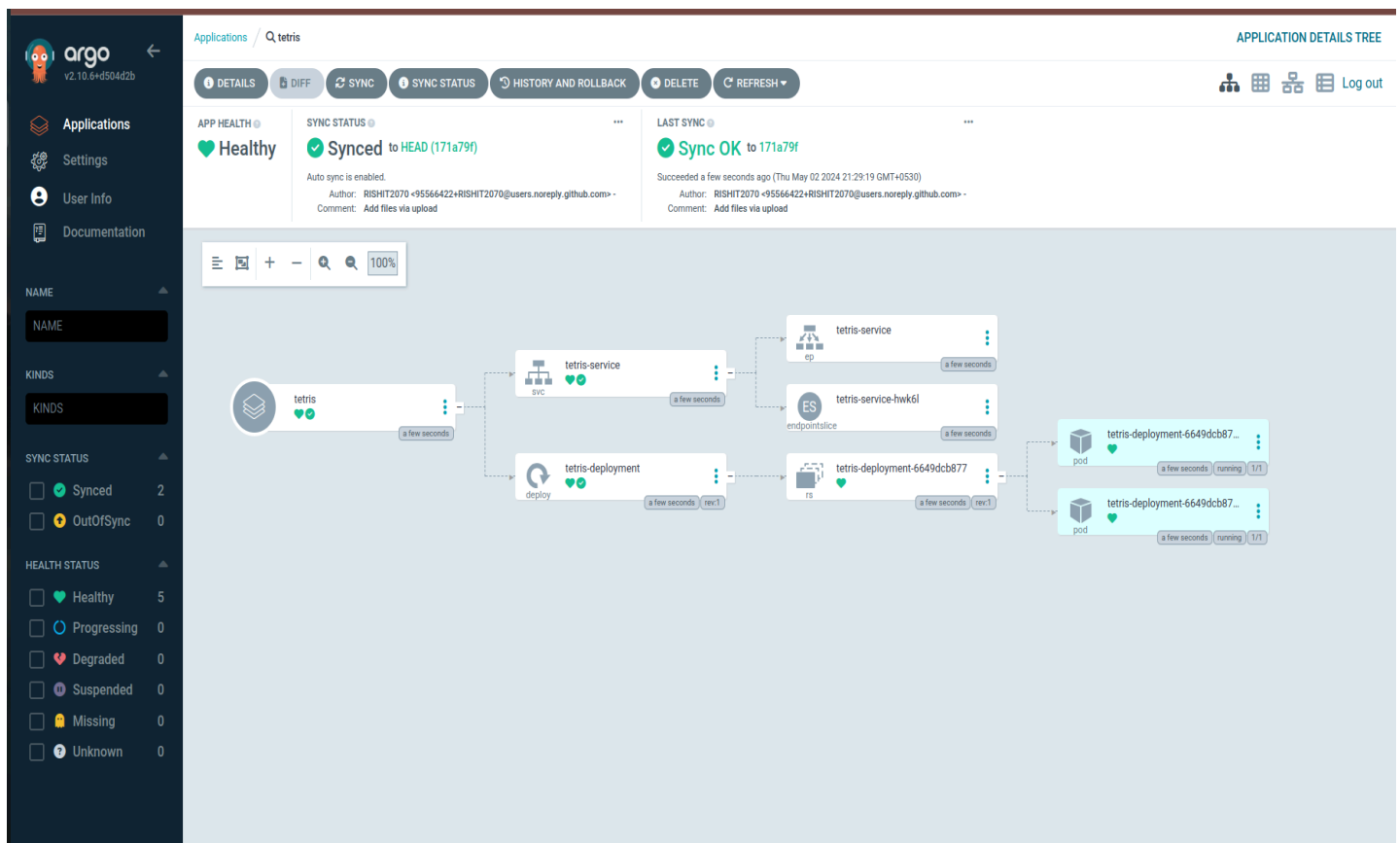
### Service.yaml

```
---
apiVersion: v1
kind: Service
metadata:
  name: tetris-service
spec:
  selector:
    app: tetris
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

## Rollout.yaml

```
apiVersion: argoproj.io/v1alpha1
kind: Rollout
metadata:
  name: tetris-rollout
  namespace: default
spec:
  template:
    metadata:
      labels:
        app: tetris
    spec:
      containers:
        - name: tetris
          image: nasi101/tetris
  selector:
    matchLabels:
      app: tetris
  replicas: 10
  strategy:
    canary:
      steps:
        - setWeight: 10
        - pause: {}
```

## DEPLOYMENT OF CI/CD PIPELINE WITHOUT ROLLOUT



## Task 3: Implementing a Canary Release with Argo Rollouts

### 1. Define a Rollout Strategy

Modified our application's Deployment to use Argo Rollouts instead of the standard Kubernetes Deployment object. This allows us to define a canary release strategy for our application rollout.

Specified the canary release strategy in the rollout definition, including parameters such as traffic weights and rollout steps.

### 2. Trigger a Rollout

Made a change to our application code, such as adding a new feature or fixing a bug.

Built and pushed a new version of the Docker image to Docker Hub, tagged with a new version number.

Updated the rollout definition in our GitHub repository to reference the new Docker image version. This triggers a new rollout process using the updated image.

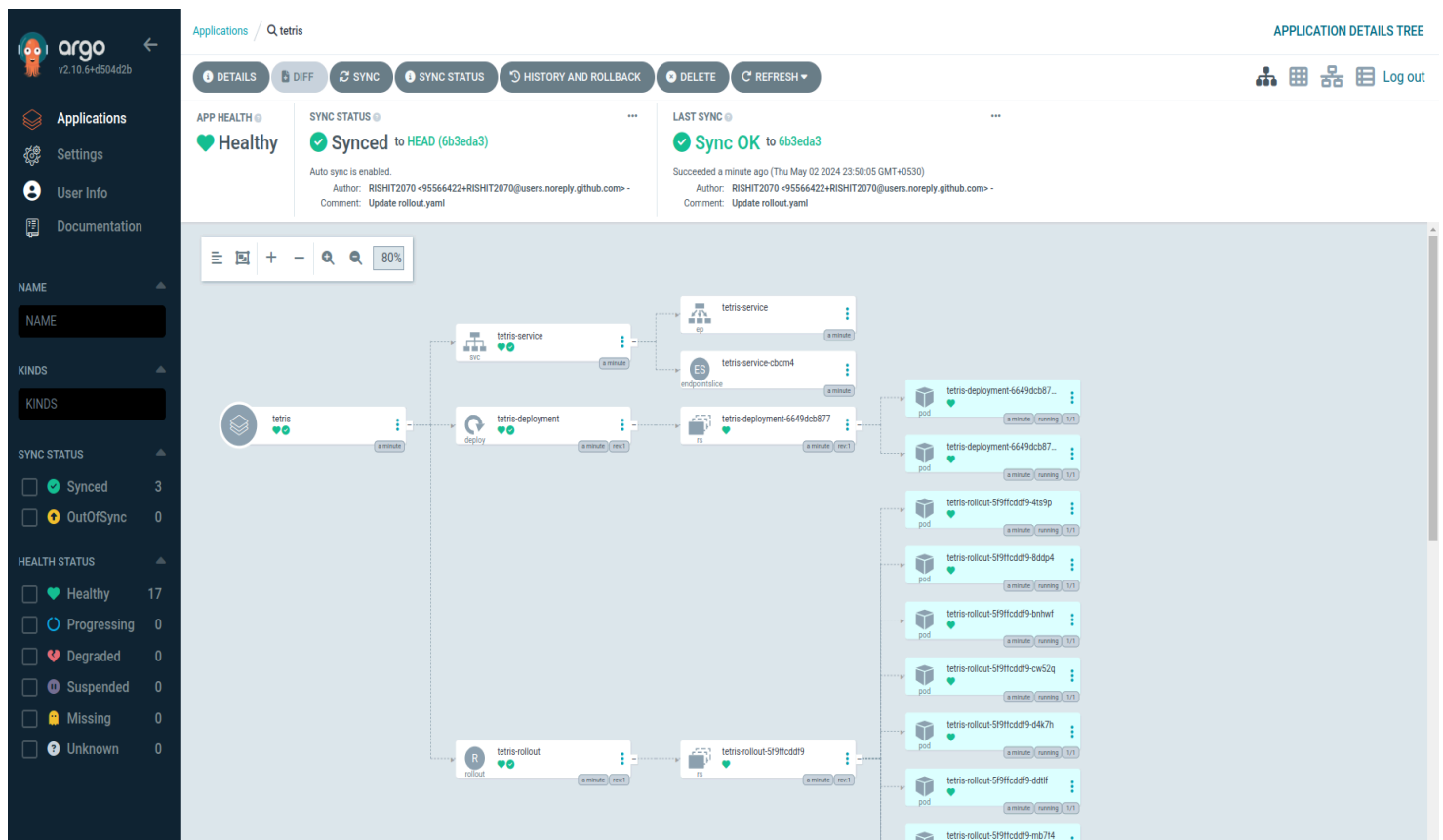
### 3. Monitor the Rollout

Used Argo Rollouts to monitor the deployment of the new version of our application.

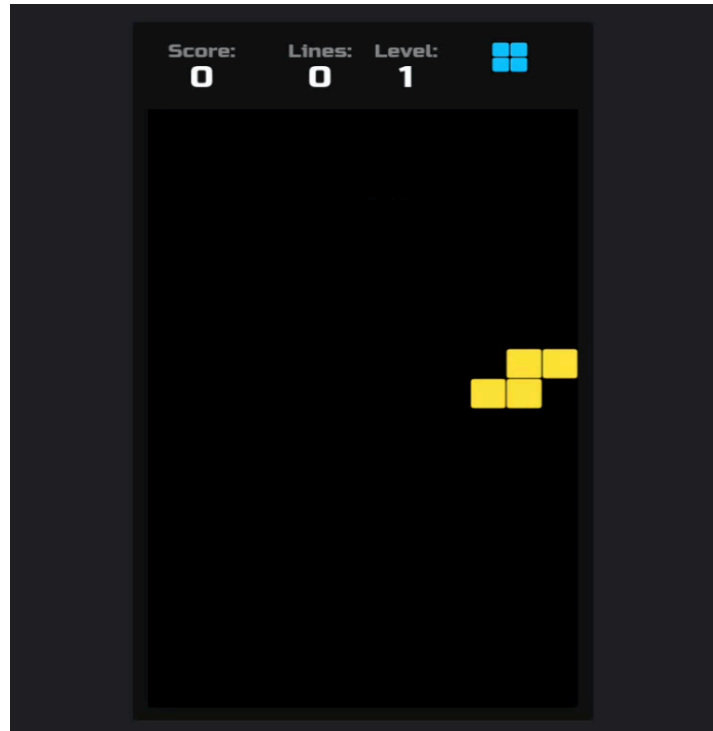
Monitored key metrics such as traffic weights and rollout progress in the Argo Rollouts UI or CLI.

Ensured that the canary release successfully completed and that the new version of the application was deployed without any issues.

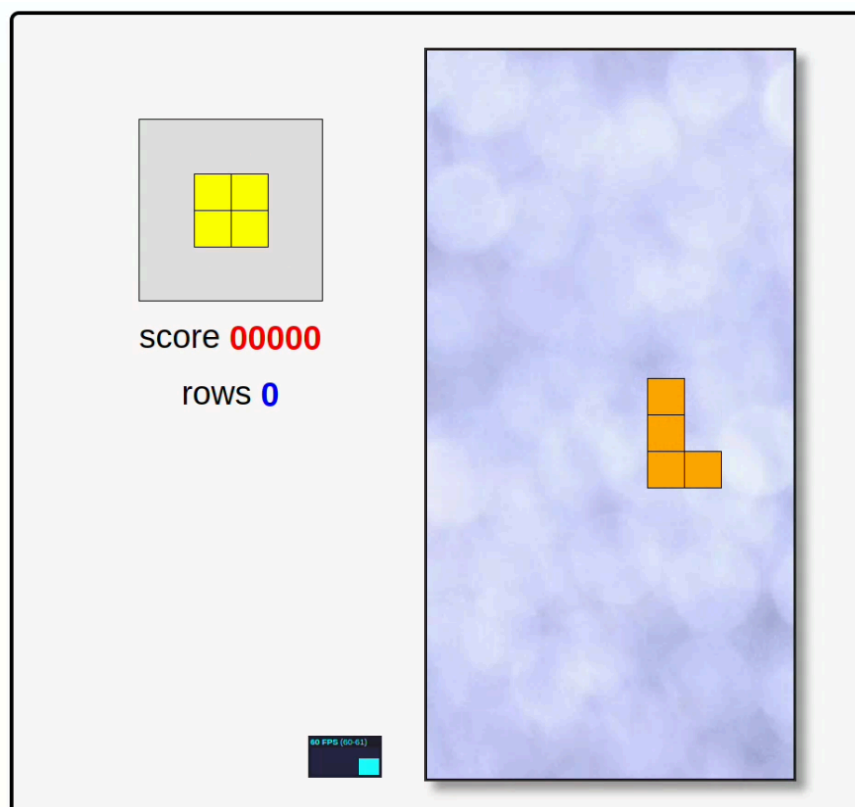
## DEPLOYMENT OF CI/CD PIPELINE WITH ARGO CD ROLLOUT



1.Game Deployed : With 1st Docker Image Before updating The 2nd Docker Image In Github Repo :



2. After Updating The 2nd Docker Image In Github Repo :



Monitoring the change Up-to-date in the docker image -:

```
[cloudshell-user@ip-10-132-65-4 ~]$  
[cloudshell-user@ip-10-132-65-4 ~]$ kubectl get rollout tetris-rollout  
NAME          DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE  
tetris-rollout  10       10       10          10         88m  
[cloudshell-user@ip-10-132-65-4 ~]$ kubectl get rollout tetris-rollout  
NAME          DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE  
tetris-rollout  10       10       10          10         88m  
[cloudshell-user@ip-10-132-65-4 ~]$ kubectl get rollout tetris-rollout  
NAME          DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE  
tetris-rollout  10       10       10          10         3m31s  
[cloudshell-user@ip-10-132-65-4 ~]$ kubectl get rollout tetris-rollout  
NAME          DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE  
tetris-rollout  10       10       1           10         9m54s  
[cloudshell-user@ip-10-132-65-4 ~]$ kubectl get rollout tetris-rollout  
NAME          DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE  
tetris-rollout  10       10       1           10         12m  
[cloudshell-user@ip-10-132-65-4 ~]$
```

Feedback

#### NOTE:

1. I worked On AWS EKS and AWS EC2 for the kubernetes part and after completing the Game deployment , I deleted the node group which I created In the EKS Cluster named "Gitops" . same goes with the cluster.
2. Terminated the EC2 Instance from AWS & If any service is active in Cost & Billing section
3. Created an Alert group on AWS for a minimum threshold cost & Billing.