**Project Title**

Scalable Deployment of Tux Racer JS Using AWS EKS and Terraform-Driven Infrastructure

**Group Members**

* John Price
* Philip Sciortino
* Travis Dumont

**Introduction and Objectives**

This project aims to deploy **Tux Racer JS**, an open-source JavaScript-based racing game (<https://github.com/ebbejan/tux-racer-js>), to the cloud using Amazon Web Services (AWS). The game will be containerized, deployed on an AWS Elastic Kubernetes Service (EKS) cluster, and managed with a modern DevOps pipeline. Our focus is to explore key cloud computing concepts such as Infrastructure-as-Code (IaC), container orchestration, and continuous integration/continuous deployment (CI/CD) while delivering a scalable, secure, and playable game accessible over the internet.

**Objectives**:

1. Build and deploy a fully functional Tux Racer JS game in a cloud environment.
2. Implement a VPC-based EKS cluster using Terraform to demonstrate IaC principles.
3. Establish a CI/CD pipeline with GitHub Actions to automate image building and deployment.
4. Ensure scalability and security through Kubernetes and AWS best practices.
5. Gain hands-on experience with cloud-native tools and workflows.

**Technologies and Tools**

* **AWS Elastic Kubernetes Service (EKS)**: Chosen for container orchestration, providing scalability and management of the game’s deployment. EKS integrates well with AWS services and supports our VPC requirements.
* **Terraform**: Used as our IaC tool to provision the VPC, EKS cluster, and supporting resources. Terraform’s declarative syntax ensures repeatable and version-controlled infrastructure.
* **Docker**: Employed to containerize Tux Racer JS, enabling consistent deployment across environments.
* **AWS Elastic Container Registry (ECR)**: Stores Docker images securely, integrating seamlessly with EKS for image pulls.
* **GitHub Actions**: Automates the CI/CD pipeline, triggering image builds and tagging on code changes.
* **AWS VPC**: Provides a secure network environment with public and private subnets for the EKS cluster.
* **IAM**: Manages permissions for EKS, ECR, and other AWS resources, ensuring least-privilege access.

**Implementation Plan**

1. **Game Preparation ()**:
   * Clone the Tux Racer JS repo and test locally.
   * Create a Dockerfile to containerize the game (e.g., use Nginx to serve the static files).
   * Push initial code to a group GitHub repository.
2. **Infrastructure Setup with Terraform**:
   * Define a VPC with public (for load balancer) and private (for EKS nodes) subnets.
   * Provision an EKS cluster with worker nodes in the private subnet.
   * Set up ECR for image storage and IAM roles for EKS/ECR access.
   * Configure security groups and networking (e.g., NAT Gateway for private subnet outbound traffic).
3. **CI/CD Pipeline with GitHub Actions**:
   * Create a workflow to:
     + Build the Docker image on code push.
     + Tag the image as build:latest and push to ECR.
     + Tag build:latest as release:x.x.x on release events.
4. **Kubernetes Deployment**:
   * Write Kubernetes manifests (e.g., Deployment, Service, Ingress) to run the game pods.
   * Configure EKS to watch the release:x.x.x tag in ECR and roll out updates.
   * Expose the game via a LoadBalancer service accessible over the internet.
5. **Testing and Validation**:
   * Test the game’s accessibility and functionality via the public URL.
   * Simulate traffic to verify scaling and pod updates.

**Splitting up the work**

* **John**: CICD, AWS Infrastructure
* **Travis**: Kubernetes
* **Philip**: Game Preparation

**Expected Outcomes and Metrics for Success**

* **Successful Deployment**: Tux Racer JS is playable via a public URL.
* **Scalability**: EKS auto-scales pods based on CPU/memory usage (e.g., 2-5 pods under load).
* **CI/CD Functionality**: Pushing code triggers an automated image build and deployment within 5 minutes.
* **Cost Efficiency**: Total AWS costs remain under the free tier or a $10 budget (e.g., using t3.micro instances).
* **Security**: Game runs in a private subnet, with only the ALB exposed publicly.

**Metrics**:

* Response time < 500ms under normal load.
* Zero downtime during deployment updates.
* Successful pod rollout on image SHA change.

**Challenges and Learning Objectives**

**Potential Challenges**:

* **EKS Networking**: Configuring VPC subnets and EKS connectivity may require troubleshooting (e.g., NAT Gateway setup). *Solution*: Follow AWS documentation and test incrementally.
* **CI/CD Complexity**: GitHub Actions integration with ECR/EKS might face authentication issues. *Solution*: Use IAM roles and validate workflow steps.
* **Game Containerization**: Tux Racer JS may need tweaks to run in Docker. *Solution*: Test locally first and adjust the Dockerfile as needed.

**Learning Objectives**:

* Master Terraform for provisioning cloud infrastructure.
* Understand Kubernetes orchestration and EKS management.
* Gain practical experience with CI/CD pipelines and containerization.
* Apply cloud security best practices (e.g., VPC isolation).