## 

## **Objective**

The objective of this capstone project is to provide you with an opportunity to apply your knowledge and skills in various technologies and tools studied throughout the course.

The project involves implementing DevOps practices for a Django project, focusing on automation, scalability, and efficient deployment on the AWS cloud using technologies such as Terraform, Docker, and Kubernetes.

**Project - 1**

Team1 (Edwin, Stephan, Xinxin) - <https://github.com/phubui1996/pokeclone>

[Project demo by developers](https://drive.google.com/file/d/1h7MKsujrleeolic1tgTOGKcOM-om9c6T/view?usp=sharing)



****

## **Project Steps**

### **Step 1: Project Setup and Prerequisites**

1. Set up a local development environment with the provided Django project.
2. [Recommended] Test and run the given project manually.
3. Set up the project in your GitHub repo.

### **Step 2: Infrastructure Provisioning**

1. Use Terraform to provision the required infrastructure on the AWS cloud. This may include but is not limited to the following services:
   * An ec2 server for provisioning purposes and shared working.
   * Creating AWS RDS postgres db.
   * Created EKS cluster.
2. Store these terraform files in a separate private repository. We may add additional steps to the project.

### **Step 3: Dockerization**

1. Create a Dockerfile in the project root directory. You must use a multi-stage build.
2. You can use node14:alpine as a base image for the frontend.
3. You can use python:alpine for the backend.
4. Define the necessary dependencies and configurations in the Dockerfile.
5. [Recommended] You should define the ENV and EXPOSE.
6. Build a Docker image for the application.
7. Test the Docker image locally to ensure it is working correctly. You may have to create a database container as well for this.

### **Step 4: Docker compose**

1. Organize your Dockerfile and project directories in the correct structure.
2. Create a docker-compose.yml file in the project directory.
3. Write the necessary services such as app and database. No need to create a frontend service. Django application is also hosting the frontend.
4. Run the docker-compose file and show it running.

### **Step 5: Orchestration with Kubernetes**

1. **[Recommended] Deploy Django app on Minikube**: Write deployment and service YAML configuration files to deploy the Django application and expose it within the Minikube cluster. You can use RDS to test.
2. **Configure Database**: Use ConfigMap and Secrets to pass database configuration to the application. Your database will be deployed on RDS.
3. **Deployments and services:** For your Django app.
4. **Namespace and Quotas**: Create a namespace for the application and apply quotas to it. All deployments should be within this namespace and abide by the quotas.
5. **Probes**: Implement any two of the liveness, readiness, and startup probes for your deployments to manage the application's lifecycle effectively.
6. **Storage class:** Write expandable storage class in EKS.
7. **Persistent Volume and Persistent Volume Claim**: Create a Persistent Volume (PV) and Persistent Volume Claim (PVC) using the **AWS storage class** provisioner.

### **Step 6: Continuous Integration**

1. Create a .github/workflows directory in the project repository.
2. Create a YAML file (e.g., ci-cd.yml) for defining the CI workflow.
3. Set up the necessary workflow triggers, such as push or pull requests.
4. Define the workflow steps:
   1. Check out the code from the repository.
   2. Build and test the Docker image.
   3. Push the Docker image to Docker Hub.

### **Step 7: Continuous Delivery**

1. Configure CICD workflows to fully automate the build, docker push, and deploy process of the Django project.
2. After successful CI, add the CD step to deploy the application on the AWS EKS cluster that you created.

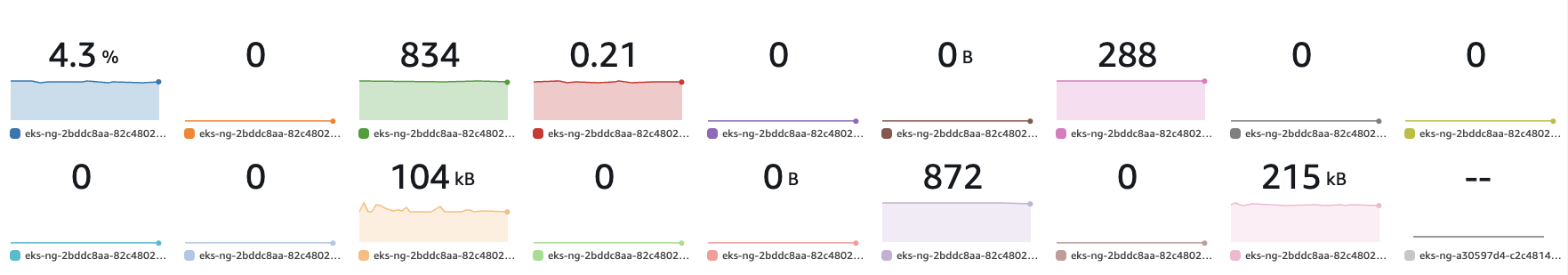
### **Step 8: Monitoring dashboard**

1. Utilize AWS CloudWatch to monitor the EKS cluster.
2. Monitoring Dashboard EKS cluster - Include metric numbers



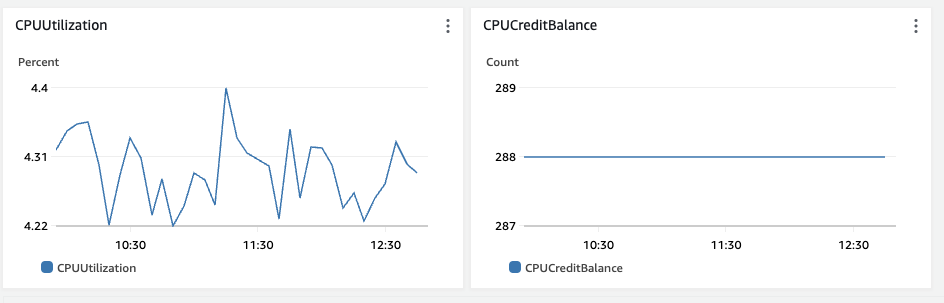
* 1. ListNodeGroups
  2. ListClusters
  3. IncomingLogEvents
  4. IncomingBytes

1. Monitoring Dashboard EKS EC2 Nodes - Include metric numbers



* 1. CPUUtilization
  2. DiskReadOps
  3. NetworkPacketsOut
  4. DiskWriteBytes
  5. StatusCheckFailed\_Instance
  6. DiskWriteOps
  7. NetworkOut
  8. DiskReadBytes
  9. NetworkPacketsIn
  10. NetworkIn

Monitoring Dashboard EKS EC2 Nodes - Include line → Metric graph



* 1. CPUUtilization
  2. CPUCreditBalance

### **Step 9: Backup**

1. Utilize Amazon EventBridge Rules to create an automated backup every midnight using a snapshot of AWS RDS.
2. [Optional] Now, write another rule to delete such snapshots every week.
3. Utilize Amazon EventBridge Rules to create an automated backup every midnight using a snapshot of AWS EC2 volume used in the EKS cluster.
4. [Optional] Now, write another rule to delete such snapshots every week.

### **Step 10: Alerting**

1. Write AWS SNS notifications to send email after every successful RDS snapshot is taken.
2. Configure alarms and send SNS notifications whenever AWS EKS node CPU utilization for any node ec2 goes above 50% or any number.

### **Step 11: Autoscaling**

1. Manually scale up the EKS nodes to two.
2. Implement target tracking “Average CPU utilization” to 50 or any number of your choice.
3. [Optional] Implement dynamic simple scaling based on autoscaling “EC2 > By Auto Scaling Group” CPU utilization.

### **Step 12: DNS and route53**

1. Write code and steps for Route53 hosted zone DNS records to deploy both projects using Terraform. (Don’t do this task. As it costs direct money. Just write steps/code)
   1. Sub-domain can be projectname.codeplatoonprojects.org or teamname.codeplatoonprojects.org
   2. Using terraform for this task is recommended.

**Key Guidance**

* Divide the project into smaller tasks and allocate specific milestones and deadlines for each task to each member.
* Encourage team members to document their work, including the setup process, configuration files, and deployment steps.
* Follow the best practices for each technology and tool, including security considerations and optimization techniques.
* Adopt collaboration and teamwork, allowing each member to work in groups and share knowledge and resources. But there should be one team leader touchpoint for instructors.
* Organize regular check-ins with instructors or TA staff members to showcase your progress, share challenges, and seek guidance.
* Both codes are in raw format. You are the owner of your choice of deployment.
* Please report if any issues or errors are found.

**Capstone limitations**

We want to make the capstone as realistic as possible but here are some limitations for multiple reasons.

* You must use region **us-east-1.**
* You must use user **east1-user** for all deployment works on EKS.
* Since the region will be shared, name your cluster with the team name.
* You can create an access key and secret access key for this user. AWS supports two keys so both teams can have one. Keep that secret. No GitHub checkins of these values.
* To save costs make sure everything is working on minikube then jump to do IaC for EKS and deploy that.
* Let’s minimize the use of RDS. Only for demo recording.

**Instructor’s advice:**

* Pay attention to EKS volume permission for SC.
* Pay attention to Django migrations on containers.

By completing this capstone project, you will gain hands-on experience in implementing DevOps practices for a Django project, demonstrating your proficiency in various technologies and tools.