**Cloud Computing and Big Data Systems - Assignment 2 Implementation Documentation**

**Part 2: Containerizing the Application using Docker**

1. Building: Using docker build -t todo-app . to create the Docker image based on the Python 3.9-slim base image
2. Tagging: Using docker tag todo-app chestnutchen/todo-app:v1.1 to prepare the image for Docker Hub
3. Pushing: Using docker push chestnutchen/todo-app:v1.1 to upload the image to my Docker Hub repository

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Part 3: Deploying the Application on Minikube**

**A screenshot of a computer program

AI-generated content may be incorrect.**

* Two pods being created: the Flask application pod and MongoDB pod
* Three services configured: flask-app-service (LoadBalancer), kubernetes (ClusterIP), and mongodb-service (ClusterIP)
* The deployment and replicaset for the application

The kubectl get pods -o wide command provides additional details showing the MongoDB pod is successfully running while the Flask app pod is still being created. The final command minikube service flask-app-service --url exposes the application's URL (<http://127.0.0.1:50766>).

**A black line on a white background

AI-generated content may be incorrect.**

**Part 4: Deploying the Application on AWS EKS**

**A screen shot of a computer

AI-generated content may be incorrect.**

The kubectl get svc command displays the services running on the EKS cluster:

* flask-app-service configured as a LoadBalancer with an external IP/DNS (aae9d4a9160bd49a28c004b0d3538517-1781435710.us-east-1.elb.amazonaws.com) and port 30266
* Kubernetes service running as ClusterIP on internal IP 10.100.0.1
* mongodb-service configured as ClusterIP on internal IP 10.100.192.233

**Part 5: Replication Controller Feature**

1. A replication controller was created using kubectl apply -f flask-replication-controller.yaml, which created 3 replicas as shown by kubectl get rc
2. The replication controller's self-healing capability was tested by:

* Verifying that 3 Flask app pods were running
* Deliberately deleting one pod (kubectl delete pod flask-app-rc-4n4cc)
* Confirming that the replication controller automatically created a new replacement pod to maintain the desired count of 3

1. The ability to scale the application was tested by:

* Scaling up to 5 replicas using kubectl scale rc flask-app-rc --replicas=5
* Verifying the successful scaling with kubectl get rc showing DESIRED=5, CURRENT=5
* Observing all 5 pods running with kubectl get pods

1. Finally, scaling down was tested by:

* Reducing to 2 replicas using kubectl scale rc flask-app-rc --replicas=2
* Confirming the scale-down with kubectl get rc showing DESIRED=2, CURRENT=2
* Verifying only 2 Flask app pods remained running

A screen shot of a computer

AI-generated content may be incorrect.

**Part 6: Rolling Update Strategy**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

1. **Preparing a new version**:

* Created a new version of the application image (v4) with docker tag chestnutchen/todo-app:v3 chestnutchen/todo-app:v4
* Pushed the new image to Docker Hub with docker push chestnutchen/todo-app:v4

2. **Performing the rolling update**:

* Updated the deployment to use the new image version with kubectl set image deployment/todo-app todo-app=chestnutchen/todo-app:v4
* Monitored the rollout status with kubectl rollout status deployment/todo-app
* Successfully completed the rolling update as indicated by "deployment "todo-app" successfully rolled out"

3. **Verifying the update**:

* Confirmed all pods are running the new version with kubectl get pods -w
* The output shows three pods in the "Running" state with very similar ages (13s and 15s), indicating they were updated nearly simultaneously but in a controlled manner