

## Part 1: Creating an Application

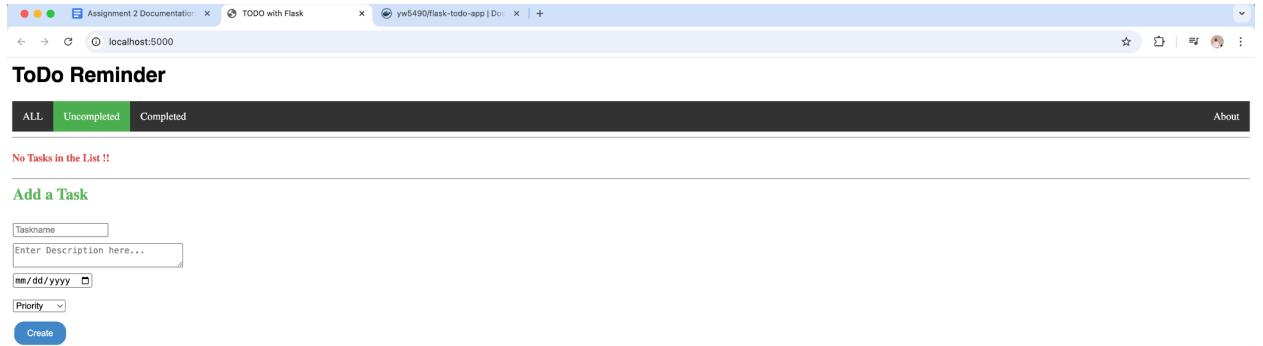
1. Download source code.
2. Create a Python virtual environment using the following command:
  - o `python3 -m venv todo`
  - o `source todo/bin/activate`
  - o `pip install -r requirements.txt`
3. Run the Web Application with: `python app.py`
4. Visit <http://localhost:5000> to check if the Web Application is running successfully as shown in the screenshot below:



## Part 2: Containerizing the Application on Docker

1. Write `Dockerfile` and `docker-compose.yml`.
2. Run the following command to build the Docker image:
  - o `docker build -t yw5490/flask-todo-app:latest .`
3. Create and start Docker containers with: `docker compose up`

- Visit <http://localhost:5000> to check if the Web Application is running successfully as shown in the screenshot below:



- Stop and remove Docker containers with: `docker compose down`
- Push the Docker image to Docker Hub using:
  - `docker push yw5490/flask-todo-app:latest`
  - [Link to Docker Hub Repository](#)

## Part 3: Deploying the Application on Minikube

- Write the `todo-app.yaml` and `mongo.yaml` file. Place the two files into a folder called `kube`.
- Start Minikube using: `minikube start`
- Deploy the Web Application using: `kubectl apply -f kube/`
- Check that the pods have been successfully created using: `kubectl get pods`

```
● yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl get pods
  NAME                  READY   STATUS    RESTARTS   AGE
  mongo-fbcbc47dc-smw65   1/1     Running   0          100s
  todo-app-684ddc77c9-8td4c   1/1     Running   0          100s
```

- Check the status of Deployments and Pods on Kubernetes Dashboard using: `minikube dashboard`

Deployments					
Name	Images	Labels	Pods	Created ↑	
mongo	mongo:latest	-	1 / 1	11.minutes.ago	⋮
todo-app	yw5490/flask-todo-app:latest	-	1 / 1	11.minutes.ago	⋮

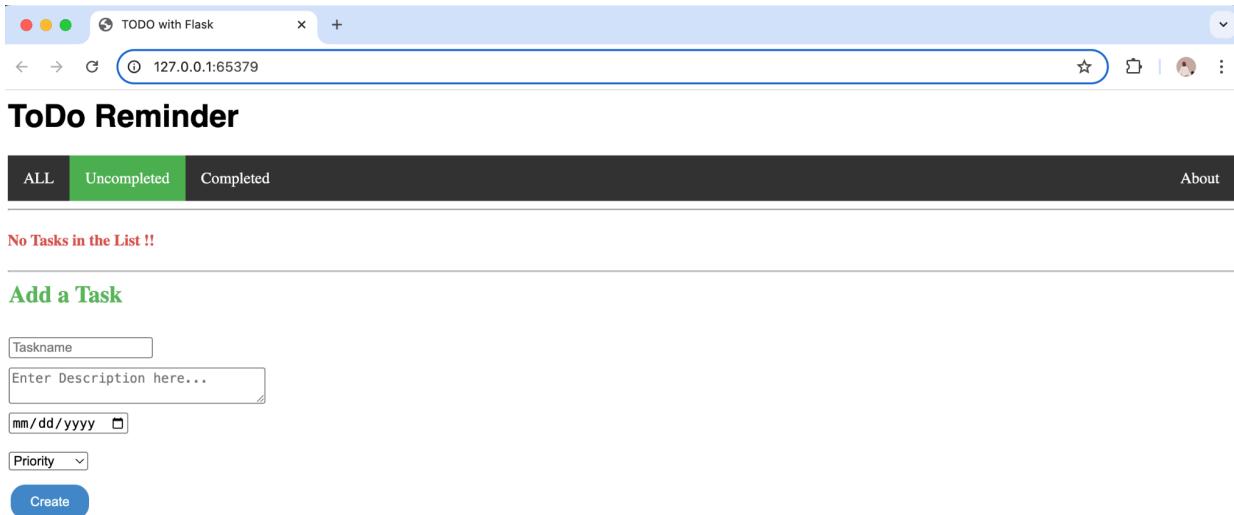
  

Pods							
Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)
							Created ↑
mongo-fbcbe47dc-smw65	mongo:latest	app: mongo pod-template-hash: fbcbe47dc	minikube	Running	0	-	12.minutes.ago
todo-app-684ddc77c9-8td4c	yw5490/flask-todo-app:latest	app: todo-app pod-template-hash: 684ddc77c9	minikube	Running	0	-	12.minutes.ago

6. Test the Web Application by visiting the service URL provided when running the following command: `minikube service todo-app --url`

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % minikube service todo-app --url
http://127.0.0.1:65379
! Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

7. The Web Application should run successfully on Minikube as shown in the screenshot below:



## Part 4: Deploying the Application on AWS EKS

1. Create an AWS EKS cluster using [this guide](#).

The screenshot shows the AWS EKS Cluster Overview page for a cluster named 'todo-app-cluster'. The left sidebar includes links for Amazon Elastic Kubernetes Service, Clusters, Settings (Console settings), Amazon EKS Anywhere (Enterprise Subscriptions), and Related services (Amazon ECR, AWS Batch). The main content area displays cluster details such as Status (Active), Kubernetes version (1.32), Support period (Standard support until March 21, 2026), and Node health issues (0). It also shows API server endpoint, Certificate authority, and IAM role ARN. The EKS Auto Mode section indicates it is disabled. The Kubernetes version settings section shows 'eks-4'. The bottom of the page includes CloudShell, Feedback, and copyright information.

2. Configure the Kubernetes CLI (kubectl) to connect to the EKS cluster using: [aws](#)

```
eks update-kubeconfig --region us-east-1 --name todo-app-cluster
```

3. Test the configuration using: [kubectl](#) get svc

```
● yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kubernetes   ClusterIP   10.100.0.1    <none>        443/TCP     23h
```

#### 4. Create EC2 Linux managed node group using [this guide](#).

The screenshot shows the AWS EKS Node Group configuration page. The node group is named 'todo-app-nodegroup'. Key details include:

- Kubernetes version:** 1.32
- AMI type:** AL2023.x86\_64\_STANDARD
- Status:** Active
- Disk size:** 20 GiB
- Subnets:** subnets-04649852f6093d5a1, subnets-067000514b2122eb, subnets-0b74e35359833d1256, subnets-0c4fe7d7e56901fbef
- Capacity type:** On-Demand
- Desired size:** 2 nodes
- Minimum size:** 2 nodes
- Maximum size:** 2 nodes
- Autoscaling group name:** eks-todo-app-nodegroup-5ccac745-5174-2ba5-9eb9-04ce54e6dfb2
- Node IAM role ARN:** arn:aws:iam::202533503880:role/myAmazonEKSNodeRole
- Node auto repair configuration:** Disabled

#### 5. Check that nodes are created and are in “Ready” status using: `kubectl get nodes`

```
● yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl get nodes
  NAME           STATUS    ROLES   AGE     VERSION
  ip-192-168-177-19.ec2.internal   Ready    <none>   4h11m   v1.32.1-eks-5d632ec
  ip-192-168-96-29.ec2.internal   Ready    <none>   4h10m   v1.32.1-eks-5d632ec
```

#### 6. Set up the Amazon EBS CSI driver through the [AWS EKS add-on](#) to utilize Amazon EBS volume as the persistent volume we used to store MongoDB data.

The screenshot shows the AWS EKS Add-on configuration page for 'aws-ebs-csi-driver'. Key details include:

- Add-on details:**
  - Status:** Active
  - Category:** storage
  - Add-on ARN:** arn:aws:eks:us-east-1:202533503880:addon/todo-app-cluster/aws-ebs-csi-driver:bacac8e6-d671-5256-c310-7d21a599f8ef
  - IAM role for service account (IRSA):** arn:aws:iam::202533503880:role/AmazonEKS\_EBS\_CSI\_DriverRole
- EKS Pod Identity (0):** No Pod Identity associations
- Advanced configuration:** (Empty)
- Health issues (0):** (Empty)

7. Make `gp2` the default storage class for any PersistentVolumeClaim that does not specify a storage class: `kubectl patch storageclass gp2 -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'`

```
● yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl get storageclass
  NAME           PROVISIONER      RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
  gp2 (default)  kubernetes.io/aws-ebs  Delete          WaitForFirstConsumer  false                  20h
```

8. Build and push a multi-architecture image to the [existing Docker Hub Repository](#) with the `Dockerfile` written in Part 2 using: `docker buildx build --platform linux/amd64,linux/arm64 -t yw5490/flask-todo-app:latest --push .`
9. Deploy the Web Application using: `kubectl apply -f kube/`
10. Check that the pods have been successfully created using: `kubectl get pods`

```
● yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl get pods
  NAME           READY   STATUS    RESTARTS   AGE
  mongo-fbcbc47dc-j6g8q     1/1     Running   0          33s
  todo-app-684ddc77c9-zwtnj 1/1     Running   0          33s
```

11. Check the status of Deployments and Pods on AWS EKS Console:

# todo-app-cluster

[Delete cluster](#) [View dashboard](#)

## Cluster info

Status: Active | Kubernetes version: 1.32 | Support period: Standard support until March 21, 2026 | Provider: EKS

Cluster health issues: 0 | Upgrade insights: 4 | Node health issues: 0

[Overview](#) [Resources](#) [Compute](#) [Networking](#) [Add-ons](#) [Access](#) [Observability](#) [Update history](#) [Tags](#)

### Resource types

- Workloads
- PodTemplates
- Pods
- ReplicaSets
- Deployments**
- StatefulSets
- DaemonSets
- Jobs
- CronJobs
- PriorityClasses
- HorizontalPodAutoscalers

### Workloads: Deployments (2)

Deployment is an API object that manages a replicated application, typically by running Pods with no local state. [Learn more](#)

	Name	Namespace	Type	Created	Pod count	Status
<a href="#">mongo</a>	default	deployments	5 minutes ago	1	<div style="width: 100%;"><div style="width: 100%;">1 Ready   0 Failed   1 Desired</div></div>	
<a href="#">todo-app</a>	default	deployments	5 minutes ago	1	<div style="width: 100%;"><div style="width: 100%;">1 Ready   0 Failed   1 Desired</div></div>	

# todo-app-cluster

[Delete cluster](#) [View dashboard](#)

## Cluster info Info

Status <span>Active</span>	Kubernetes version   <a href="#">Info</a> 1.32	Support period <a href="#">Standard support until March 21, 2026</a>	Provider EKS
Cluster health issues <span>0</span>	Upgrade insights <span>4</span>	Node health issues <span>0</span>	

[Overview](#) [Resources](#) [Compute](#) [Networking](#) [Add-ons](#) [Access](#) [Observability](#) [Update history](#) [Tags](#)

**Resource types** X

- Workloads
- PodTemplates
- Pods
- ReplicaSets
- Deployments
- StatefulSets
- DaemonSets
- Jobs
- CronJobs
- PriorityClasses
- HorizontalPodAutoscalers

**Workloads: Pods (2)**

Pod is the smallest and simplest Kubernetes object. A Pod represents a set of running containers on your cluster. [Learn more](#)

default	<input type="text"/> Filter Pods by name	<span>&lt;</span> <span>1</span> <span>&gt;</span>
<input type="radio"/>	mongo-fbcbc47dc-j6g8q	5 minutes ago
<input type="radio"/>	todo-app-684ddc77c9-zwtrnj	5 minutes ago

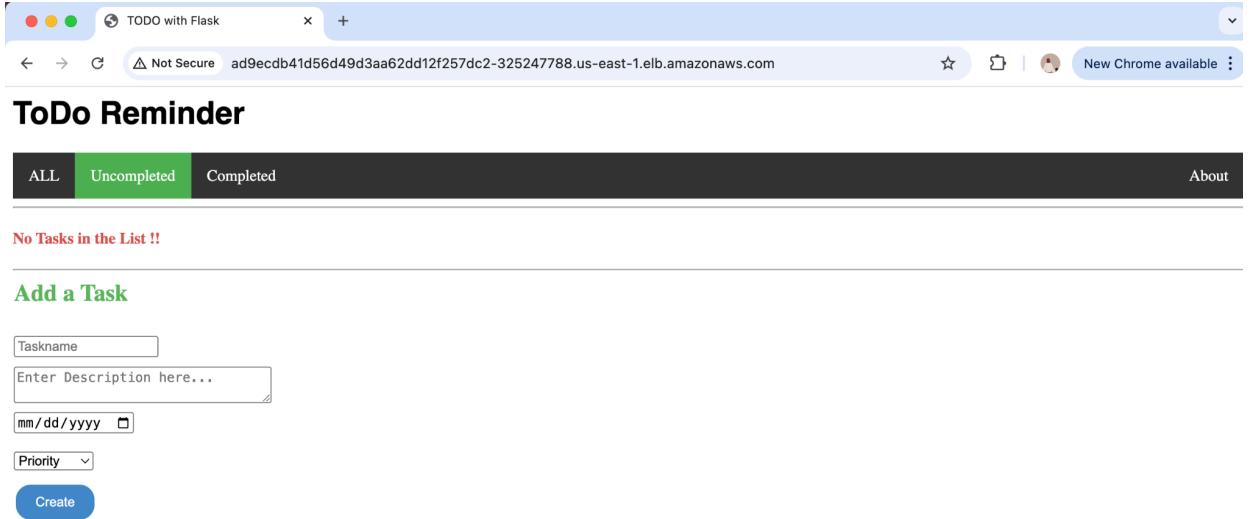
[View details](#)

12. Test the Web Application by accessing the EXTERNAL-IP provided when running the following command: `kubectl get svc`

● yirongwang@Yirongs-MacBook-Pro cloud\_computing\_hw2 % kubectl get s

Name	Type	IP	Port	Target IP
kubernetes	ClusterIP	10.100.0.1	<none>	
mongo	ClusterIP	10.100.237.30	<none>	
todo-app	LoadBalancer	10.100.31.66		ad9ecdb41d56d49d3aa62dd1

13. The Web Application should run successfully on AWS EKS as shown in the screenshot below:



## Part 5: Replication Controller Feature

1. Write the `todo-app-rc.yaml` file, specify that 3 replicas of the application are always running.
2. Place the file into a folder called `kube-rc`, together with the `mongo.yaml` file we wrote in Part 3 and a `todo-app-load-balancer.yaml` file defining the Load Balancer Service, copied from Part 3.
3. Start Minikube using: `minikube start`
4. Check the current Kubernetes context using: `kubectl config get-contexts`

```
● vironwang@ironwang-MacBook-Pro ~ % kubectl config get-contexts
CURRENT   NAME         CLUSTER          AUTHINFO           NAMESPACE
*         minikube     arn:aws:eks:us-east-1:202533503880:cluster/todo-app-cluster   minikube
                                         arn:aws:eks:us-east-1:202533503880:cluster/todo-app-cluster   default
```

5. If the current context is not Minikube, set it to Minikube using: `kubectl config use-context minikube`
6. Deploy the Web Application with Replication Controller using: `kubectl apply -f kube-rc/`

7. Verify that 3 replicas of the application are created and running using: `kubectl`

```
get pods
```

NAME	READY	STATUS	RESTARTS	AGE
mongo-fbcfc47dc-wf2mq	1/1	Running	0	18s
todo-app-with-rc-dd2sp	1/1	Running	0	18s
todo-app-with-rc-dkn5n	1/1	Running	0	18s
todo-app-with-rc-gjtj2	1/1	Running	0	18s

8. Test the Replication Controller by intentionally deleting one of the pods: `kubectl`

```
delete pod <pod-name>
```

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl delete pod todo-app-with-rc-dd2sp  
pod "todo-app-with-rc-dd2sp" deleted
```

9. Use `kubectl get pods` to verify that a new pod is automatically created by the Replication Controller to maintain that 3 replicas are running for the application:

NAME	READY	STATUS	RESTARTS	AGE
mongo-fbcfc47dc-wf2mq	1/1	Running	0	60s
todo-app-with-rc-dkn5n	1/1	Running	0	60s
todo-app-with-rc-gjtj2	1/1	Running	0	60s
todo-app-with-rc-tfx55	1/1	Running	0	5s

10. Update the `todo-app-rc.yaml` file to set the desired number of replicas to 5.

11. Apply the changes to the running Replication Controller using: `kubectl apply`

```
-f kube-rc/todo-app-rc.yaml
```

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl apply -f kube-rc/todo-app-rc.yaml  
replicationcontroller/todo-app-with-rc configured
```

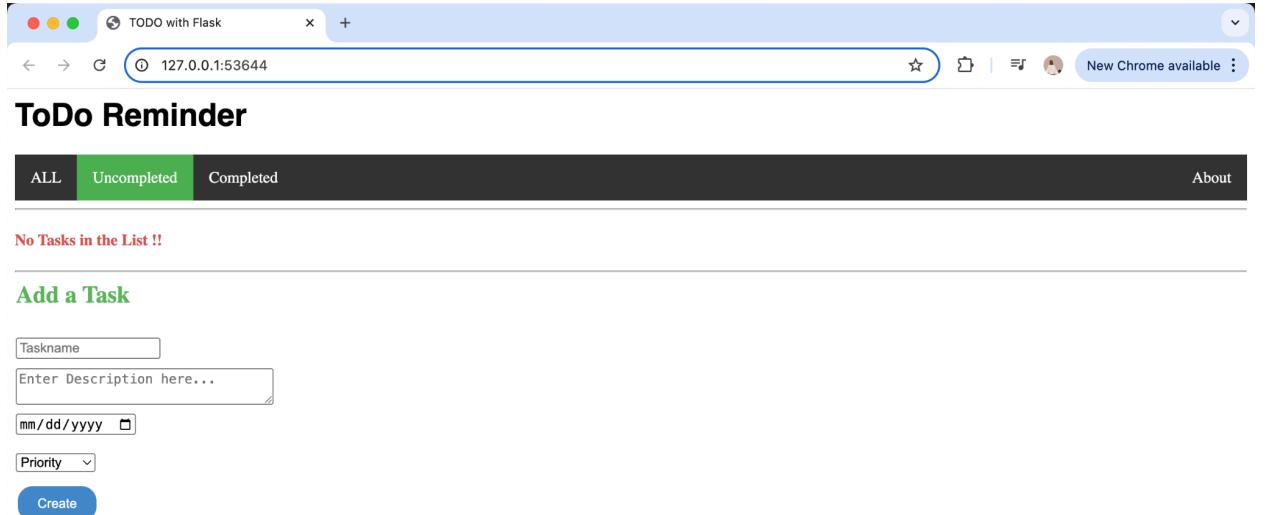
12. Use `kubectl get pods` to verify that the Replication Controller scales up the number of replicas to 5:

NAME	READY	STATUS	RESTARTS	AGE
mongo-fbcfc47dc-wf2mq	1/1	Running	0	2m38s
todo-app-with-rc-2jx9z	1/1	Running	0	15s
todo-app-with-rc-5vx8t	1/1	Running	0	15s
todo-app-with-rc-dkn5n	1/1	Running	0	2m38s
todo-app-with-rc-gjtj2	1/1	Running	0	2m38s
todo-app-with-rc-tfx55	1/1	Running	0	103s

13. Test the Web Application by visiting the service URL provided when running the following command: `minikube service todo-app --url`

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % minikube service todo-app --url  
http://127.0.0.1:53644  
! Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

14. The Web Application should run successfully on Minikube as shown in the screenshot below:



## Part 6: Rolling Update Strategy

1. Update the `todo-app.yaml` file to set the update strategy to Rolling Update, and set the maximum number of pods that can be unavailable during the update to 1.
2. Push a new version of the Docker image to the [existing Docker Hub Repository](#) using the following command:
  - o `docker tag yw5490/flask-todo-app:latest yw5490/flask-todo-app:v2`
  - o `docker push yw5490/flask-todo-app:v2`
3. Start Minikube using: `minikube start`
4. Deploy the Web Application using: `kubectl apply -f kube/`

5. Check that the pods have been successfully created using: `kubectl get pods`

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
mongo-fbcfc47dc-x2nn6     1/1     Running   0          5s
todo-app-684ddc77c9-2cnpx 1/1     Running   0          4s
todo-app-684ddc77c9-jr47b 1/1     Running   0          4s
todo-app-684ddc77c9-nb7fd 1/1     Running   0          4s
```

6. Check the current version of the Docker image used by the deployment: `kubectl`

```
describe deployment todo-app
```

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl describe deployment todo-app
Name:           todo-app
Namespace:      default
CreationTimestamp: Sun, 16 Mar 2025 17:20:14 -0400
Labels:          <none>
Annotations:    deployment.kubernetes.io/revision: 1
Selector:        app=todo-app
Replicas:       3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 1 max unavailable, 25% max surge
Pod Template:
  Labels:  app=todo-app
  Containers:
    todo-app:
      Image:      yw5490/flask-todo-app:latest
      Port:       5000/TCP
      Host Port:  0/TCP
      Limits:
        cpu:  1
        memory: 512Mi
      Environment:
        MONGO_HOST: mongo
        MONGO_PORT: 27017
      Mounts:  <none>
      Volumes: <none>
      Node-Selectors: <none>
      Tolerations: <none>
  Conditions:
    Type      Status  Reason
    ----      ----   -----
    Available  True    MinimumReplicasAvailable
    Progressing True    NewReplicaSetAvailable
    OldReplicaSets: <none>
    NewReplicaSet:  todo-app-684ddc77c9 (3/3 replicas created)
  Events:
    Type      Reason          Age     From               Message
    ----      ----   ----   ----   -----
    Normal   ScalingReplicaSet 9s    deployment-controller  Scaled up replica set todo-app-684ddc77c9 from 0 to 3
```

7. Trigger the Rolling Update by updating the deployment with the new Docker

```
image version using: kubectl set image deployments/todo-app
```

```
todo-app=yw5490/flask-todo-app:v2
```

8. Monitor the Rolling Update progress using: `kubectl rollout status`

```
deployments/todo-app --watch
```

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl rollout status deployments/todo-app --watch
Waiting for deployment "todo-app" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "todo-app" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "todo-app" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "todo-app" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "todo-app" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "todo-app" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "todo-app" rollout to finish: 2 of 3 updated replicas are available...
deployment "todo-app" successfully rolled out
```

9. Check that the pods have been successfully updated using: `kubectl get pods`

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl get pods
NAME                  READY   STATUS    RESTARTS   AGE
mongo-fbcfc47dc-x2nn6   1/1     Running   0          88s
todo-app-698d85c55d-cs4kt   1/1     Running   0          58s
todo-app-698d85c55d-qtjjt   1/1     Running   0          57s
todo-app-698d85c55d-wt8wp   1/1     Running   0          56s
```

10. Check that the updated application deployment is running with the new Docker

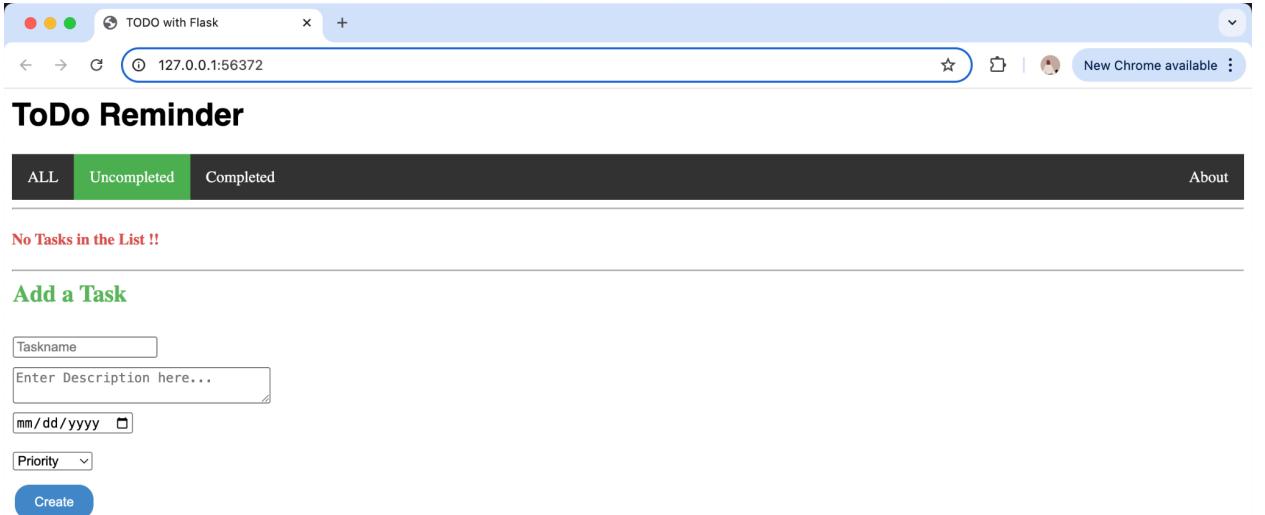
```
image version: kubectl describe deployment todo-app
```

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl describe deployment todo-app
Name:           todo-app
Namespace:      default
CreationTimestamp: Sun, 16 Mar 2025 17:20:14 -0400
Labels:          <none>
Annotations:    deployment.kubernetes.io/revision: 2
Selector:        app=todo-app
Replicas:       3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 1 max unavailable, 25% max surge
Pod Template:
  Labels:  app=todo-app
  Containers:
    todo-app:
      Image:      yw5490/flask-todo-app:v2
      Port:       5000/TCP
      Host Port: 0/TCP
      Limits:
        cpu: 1
        memory: 512Mi
      Environment:
        MONGO_HOST: mongo
        MONGO_PORT: 27017
      Mounts:  <none>
      Volumes: <none>
      Node-Selectors: <none>
      Tolerations: <none>
  Conditions:
    Type        Status  Reason
    ----        ----  -----
    Available   True    MinimumReplicasAvailable
    Progressing True    NewReplicaSetAvailable
  OldReplicaSets: todo-app-684ddc77c9 (0/0 replicas created)
  NewReplicaSet:  todo-app-698d85c55d (3/3 replicas created)
Events:
  Type      Reason          Age   From            Message
  ----      ----          ---   ----            -----
  Normal   ScalingReplicaSet 98s   deployment-controller  Scaled up replica set todo-app-684ddc77c9 from 0 to 3
  Normal   ScalingReplicaSet 69s   deployment-controller  Scaled up replica set todo-app-698d85c55d from 0 to 1
  Normal   ScalingReplicaSet 69s   deployment-controller  Scaled down replica set todo-app-684ddc77c9 from 3 to 2
  Normal   ScalingReplicaSet 68s   deployment-controller  Scaled up replica set todo-app-698d85c55d from 1 to 2
  Normal   ScalingReplicaSet 67s   deployment-controller  Scaled down replica set todo-app-684ddc77c9 from 2 to 1
  Normal   ScalingReplicaSet 67s   deployment-controller  Scaled up replica set todo-app-698d85c55d from 2 to 3
  Normal   ScalingReplicaSet 67s   deployment-controller  Scaled down replica set todo-app-684ddc77c9 from 1 to 0
```

11. Test the Web Application by visiting the service URL provided when running the following command: `minikube service todo-app --url`

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % minikube service todo-app --url  
http://127.0.0.1:56372  
! Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

12. The Web Application should run successfully on Minikube as shown in the screenshot below:



## Part 7: Health Monitoring

1. Update the `todo-app.yaml` file to set up a liveness Probe and a readiness Probe for the pods.
2. Start Minikube using: `minikube start`
3. Deploy the Web Application using: `kubectl apply -f kube/`
4. Check that the pods have been successfully created using: `kubectl get pods`

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl get pods  
NAME                  READY   STATUS    RESTARTS   AGE  
mongo-fbcfc47dc-gdrg5  1/1     Running   0          8s  
todo-app-54c6868ff5-7fpcv 1/1     Running   0          8s  
todo-app-54c6868ff5-mc8gx 1/1     Running   0          8s  
todo-app-54c6868ff5-wbkdc 1/1     Running   0          8s
```

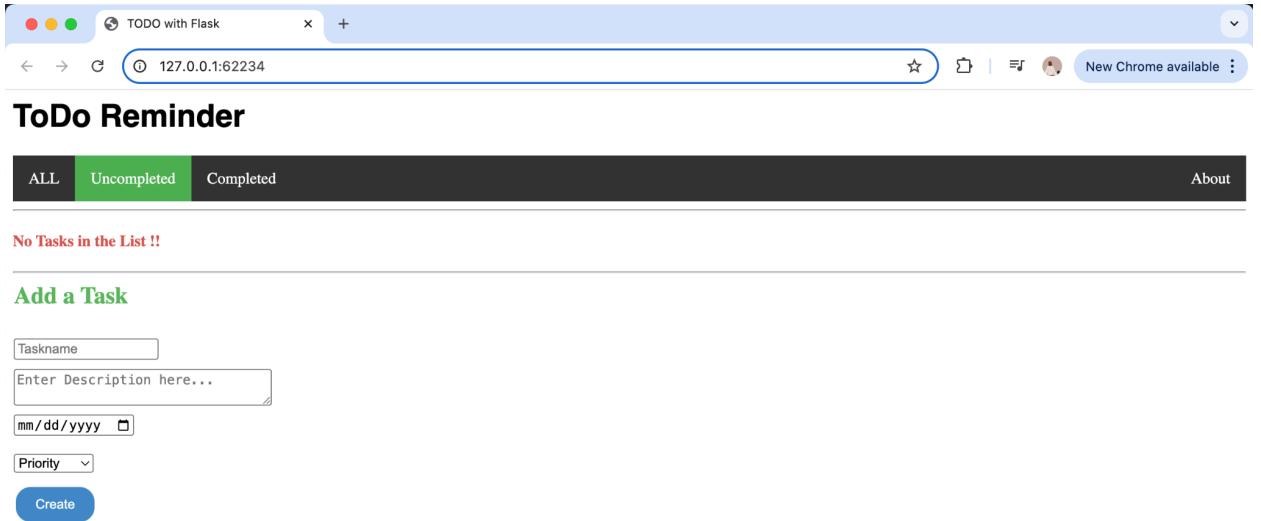
5. Check that the liveness Probe and the readiness Probe has been successfully configured using: `kubectl describe deployment todo-app`

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl describe deployment todo-app
Name:           todo-app
Namespace:      default
CreationTimestamp:  Sun, 16 Mar 2025 23:28:01 -0400
Labels:          <none>
Annotations:    deployment.kubernetes.io/revision: 1
Selector:        app=todo-app
Replicas:        3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 1 max unavailable, 25% max surge
Pod Template:
  Labels:  app=todo-app
  Containers:
    todo-app:
      Image:      yw5490/flask-todo-app:latest
      Port:       5000/TCP
      Host Port: 0/TCP
      Limits:
        cpu: 1
        memory: 512Mi
      Liveness:  http-get http://:5000/ delay=5s timeout=1s period=5s #success=1 #failure=3
      Readiness:  http-get http://:5000/ delay=3s timeout=1s period=3s #success=1 #failure=3
      Environment:
        MONGO_HOST: mongo
        MONGO_PORT: 27017
      Mounts:      <none>
      Volumes:     <none>
      Node-Selectors: <none>
      Tolerations:  <none>
  Conditions:
    Type      Status  Reason
    ----      ----  -----
    Available  True    MinimumReplicasAvailable
    Progressing  True    NewReplicaSetAvailable
  OldReplicaSets: <none>
  NewReplicaSet:  todo-app-54c6868ff5 (3/3 replicas created)
  Events:
    Type      Reason          Age     From            Message
    ----      ----          ----  ----            -----
    Normal   ScalingReplicaSet  30s    deployment-controller  Scaled up replica set todo-app-54c6868ff5 from 0 to 3
```

6. Test the Web Application by visiting the service URL provided when running the following command: `minikube service todo-app --url`

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % minikube service todo-app --url
http://127.0.0.1:62234
! Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

7. The Web Application should run successfully on Minikube as shown in the screenshot below:



8. To test the health monitoring system, add an intentional failure to the application code as below:

```
@app.route("/test")
def test():
    raise Exception("Intentional failure for testing probes. ")
```

9. Build and push a new Docker image version to the [existing Docker Hub Repository](#) with the updated application code using: `docker buildx build --platform linux/amd64,linux/arm64 -t yw5490/flask-todo-app:v3 --push .`
10. Update the deployment with the new Docker image version using: `kubectl set image deployments/todo-app todo-app=yw5490/flask-todo-app:v3`

11. Check that the new image version has been rolled out successfully using:

```
kubectl rollout status deployments/todo-app
```

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % kubectl rollout status deployments/todo-app
deployment "todo-app" successfully rolled out
```

12. Test the new application code by visiting the service URL provided when running the following command: `minikube service todo-app --url`

```
yirongwang@Yirongs-MacBook-Pro cloud_computing_hw2 % minikube service todo-app --url
http://127.0.0.1:62639
! Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

13. The Web Application should run into an Exception when visiting the /test route:



14. Temporarily modify `todo-app.yaml` to set the Docker image version to `yw5490/flask-todo-app:v3` (so that it doesn't conflict with the Docker image version update we completed above) and explicitly hit the failing endpoint `/test` as shown below:

```
livenessProbe:  
  httpGet:  
    # path: /  
    path: /test  
    port: 5000  
  initialDelaySeconds: 5  
  periodSeconds: 5  
readinessProbe:  
  httpGet:  
    # path: /  
    path: /test  
    port: 5000  
  initialDelaySeconds: 3  
  periodSeconds: 3
```

15. Apply the temporary changes using: `kubectl apply -f kube/todo-app.yaml`

16. Monitor the health of the pods using: `kubectl get pods --watch`

NAME	READY	STATUS	RESTARTS	AGE
mongo-fbcdbc47dc-gdrg5	1/1	Running	0	56m
todo-app-85fc55f495-j7qhx	1/1	Running	0	31m
todo-app-85fc55f495-kbjv4	1/1	Running	0	31m
todo-app-85fc55f495-ltgbg	1/1	Running	0	31m
todo-app-5d8b6496c5-snkm	0/1	Pending	0	0s
todo-app-85fc55f495-j7qhx	1/1	Terminating	0	31m
todo-app-5d8b6496c5-snkm	0/1	Pending	0	0s
todo-app-5d8b6496c5-s4k5d	0/1	Pending	0	0s
todo-app-5d8b6496c5-snkm	0/1	ContainerCreating	0	0s
todo-app-5d8b6496c5-s4k5d	0/1	Pending	0	0s
todo-app-5d8b6496c5-s4k5d	0/1	ContainerCreating	0	0s
todo-app-85fc55f495-j7qhx	0/1	Completed	0	31m
todo-app-85fc55f495-j7qhx	0/1	Completed	0	31m
todo-app-85fc55f495-j7qhx	0/1	Completed	0	31m
todo-app-5d8b6496c5-snkm	0/1	Running	0	2s
todo-app-5d8b6496c5-s4k5d	0/1	Running	0	2s
todo-app-5d8b6496c5-s4k5d	0/1	Running	1 (1s ago)	21s
todo-app-5d8b6496c5-snkm	0/1	Running	1 (1s ago)	22s
todo-app-5d8b6496c5-s4k5d	0/1	Running	2 (1s ago)	41s
todo-app-5d8b6496c5-snkm	0/1	Running	2 (0s ago)	41s
todo-app-5d8b6496c5-s4k5d	0/1	Running	3 (1s ago)	61s
todo-app-5d8b6496c5-snkm	0/1	Running	3 (0s ago)	61s
todo-app-5d8b6496c5-snkm	0/1	CrashLoopBackOff	3 (1s ago)	81s
todo-app-5d8b6496c5-s4k5d	0/1	Running	4 (0s ago)	81s
todo-app-5d8b6496c5-s4k5d	0/1	CrashLoopBackOff	4 (0s ago)	101s
todo-app-5d8b6496c5-snkm	0/1	Running	4 (29s ago)	109s
todo-app-5d8b6496c5-snkm	0/1	Running	5 (2s ago)	2m7s
todo-app-5d8b6496c5-snkm	0/1	CrashLoopBackOff	5 (0s ago)	2m26s
todo-app-5d8b6496c5-s4k5d	0/1	Running	5 (55s ago)	2m36s
todo-app-5d8b6496c5-s4k5d	0/1	CrashLoopBackOff	5 (1s ago)	2m57s

17. Verify that Kubernetes takes the appropriate action when a Probe fails using:

`kubectl describe pod <pod-name>`

Events:				
Type	Reason	Age	From	Message
Normal	Scheduled	3m49s	default-scheduler	Successfully assigned default/todo-app-5d8b6496c5-snkm to minikube
Normal	Pulled	3m48s	kubelet	Successfully pulled image "yw5490/flask-todo-app:v3" in 397ms (397ms including waiting). Image size: 171048631 bytes.
Normal	Pulled	3m28s	kubelet	Successfully pulled image "yw5490/flask-todo-app:v3" in 285ms (295ms including waiting). Image size: 171048631 bytes.
Normal	Pulled	3m8s	kubelet	Successfully pulled image "yw5490/flask-todo-app:v3" in 245ms (476ms including waiting). Image size: 171048631 bytes.
Warning	Unhealthy	2m59s (x7 over 3m39s)	kubelet	Liveness probe failed: HTTP probe failed with statuscode: 500
Warning	Unhealthy	2m58s (x18 over 3m45s)	kubelet	Readiness probe failed: HTTP probe failed with statuscode: 500
Normal	Pulled	2m48s	kubelet	Successfully pulled image "yw5490/flask-todo-app:v3" in 199ms (199ms including waiting). Image size: 171048631 bytes.
Normal	Pulling	2m1s (x5 over 3m48s)	kubelet	Pulling image "yw5490/flask-todo-app:v3"
Normal	Created	2m1s (x5 over 3m48s)	kubelet	Created container: todo-app
Normal	Started	2m1s (x5 over 3m48s)	kubelet	Started container todo-app
Normal	Pulled	2m1s	kubelet	Successfully pulled image "yw5490/flask-todo-app:v3" in 530ms (531ms including waiting). Image size: 171048631 bytes.
Normal	Killing	104s (x5 over 3m29s)	kubelet	Container todo-app failed liveness probe, will be restarted

## Part 8: Alerting

1. Install Prometheus locally using the following command:

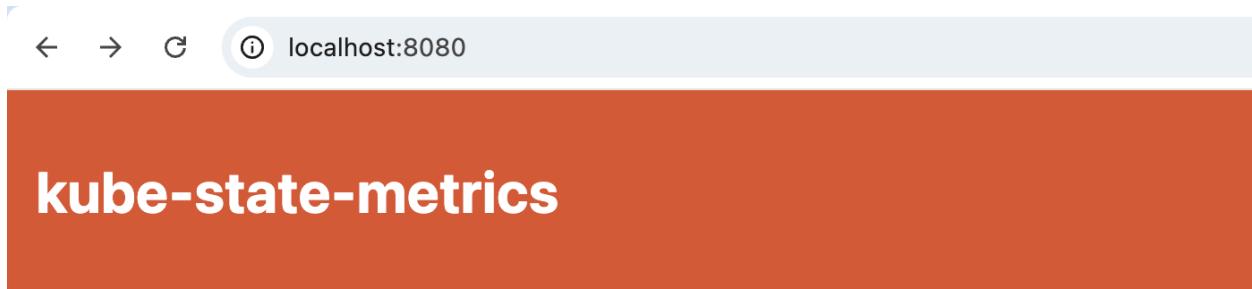
- `curl -LO`  
<https://github.com/prometheus/prometheus/releases/download/v3.2.1/prometheus-3.2.1.darwin-arm64.tar.gz>
  - `tar -xvzf prometheus-3.2.1.darwin-arm64.tar.gz`
2. `cd` `prometheus-3.2.1.darwin-arm64` into the Prometheus folder.
    - Create the custom alert file using: `touch custom-alert.yml`
    - Write the `custom-alert.yml` file that defines the rule for our alert.
    - Update the `prometheus.yml` config file.
  3. Create a webhook in our selected Slack workspace channel “#assignment-2” and obtain the webhook URL using [these instructions](#).
  4. Confirm that the webhook is working properly using: `curl -X POST -H 'Content-type: application/json' --data '{"text":"Test message from Alertmanager"}'`  
<https://hooks.slack.com/services/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>



**Assignment 2** APP 4:58 AM  
Test message from Alertmanager

5. Install Alertmanager locally using the following command:
  - `curl -LO`  
<https://github.com/prometheus/alertmanager/releases/download/v0.28.1/alertmanager-0.28.1.darwin-arm64.tar.gz>
  - `tar -xvzf alertmanager-0.28.1.darwin-arm64.tar.gz`
6. `cd` `alertmanager-0.28.1.darwin-arm64` into the Alertmanager folder. Update the `alertmanager.yml` config file.
7. Deploy `kube-state-metrics` using the following command:
  - `git clone`  
<https://github.com/kubernetes/kube-state-metrics.git>
  - `cd kube-state-metrics`

- `kubectl apply -f examples/standard`
8. Confirm that `kube-state-metrics` has been successfully deployed and running using: `kubectl get pods -n kube-system | grep kube-state-metrics`
- ```
yirongwang@Yirongs-MacBook-Pro kube-state-metrics % kubectl get pods -n kube-system | grep kube-state-metrics
kube-state-metrics-79b5f8c986-5gf8z 1/1 Running 0 2m2s
```
9. Expose `kube-state-metrics` on port 8080 using: `kubectl port-forward -n kube-system svc/kube-state-metrics 8080:8080`
10. Verify that the `kube-state-metrics` Dashboard is loading successfully on <http://localhost:8080/>.



## Metrics for Kubernetes' state

Version: (version=v2.15.0, branch=, revision=unknown)

- [Metrics](#)
- [Healthz](#)
- [Livez](#)

Download a detailed report of resource usage (pprof format, from the Go runtime):

- [heap usage \(memory\)](#)
- [CPU usage \(60 second profile\)](#)

To visualize and share profiles you can upload to [pprof.me](http://pprof.me)

11. Start Prometheus in the `prometheus-3.2.1.darwin-arm64` folder using:

```
./prometheus --config.file=prometheus.yml &
```

12. Load the Prometheus Dashboard through: <http://localhost:9090>.

13. Verify that the custom alert is showing up on the Prometheus Dashboard.

The screenshot shows the Prometheus UI at [localhost:9090/alerts](http://localhost:9090/alerts). The top navigation bar has tabs for 'Query', 'Alerts' (which is selected), and 'Status'. Below the navigation is a search bar for 'Filter by rule name or labels'. A table lists a single alert rule:

| pod-health-alerts                                                    |                                                                                     | custom-alert.yml | INACTIVE (1) |
|----------------------------------------------------------------------|-------------------------------------------------------------------------------------|------------------|--------------|
| <b>PodCrashLooping</b>                                               |                                                                                     |                  |              |
| <pre>rate(kube_pod_container_status_restarts_total[5m]) &gt; 0</pre> |                                                                                     |                  |              |
| <pre>for: 2m</pre>                                                   |                                                                                     |                  |              |
| <pre>severity="critical"</pre>                                       |                                                                                     |                  |              |
| description                                                          | The pod {{ \$labels.pod }} has restarted {{ \$value }} times in the last 5 minutes. |                  |              |
| summary                                                              | Pod {{ \$labels.pod }} in namespace {{ \$labels.namespace }} is crash looping       |                  |              |

14. Verify that the `kube-state-metrics` scraping target is up on <http://localhost:9090/targets>.

The screenshot shows the Prometheus UI at [localhost:9090/targets](http://localhost:9090/targets). The top navigation bar has tabs for 'Query', 'Alerts', 'Status' (selected), and 'Target health'. Below the navigation is a search bar for 'Filter by target health'. The page displays two target sections:

| kube-state-metrics                                                        |                                                    | 1 / 1 up    |          |
|---------------------------------------------------------------------------|----------------------------------------------------|-------------|----------|
| Endpoint                                                                  | Labels                                             | Last scrape | State    |
| <a href="http://localhost:8080/metrics">http://localhost:8080/metrics</a> | instance="localhost:8080" job="kube-state-metrics" | 14.518s ago | 185ms UP |

| prometheus                                                                |                                            | 1 / 1 up    |         |
|---------------------------------------------------------------------------|--------------------------------------------|-------------|---------|
| Endpoint                                                                  | Labels                                     | Last scrape | State   |
| <a href="http://localhost:9090/metrics">http://localhost:9090/metrics</a> | instance="localhost:9090" job="prometheus" | 5.617s ago  | 13ms UP |

15. Start Alertmanager in the `alertmanager-0.28.1.darwin-arm64` folder using:

```
./alertmanager --config.file=alertmanager.yml &
```

16. Load the Alertmanager Dashboard through: <http://localhost:9093>.

17. Verify that the Cluster Status is ready.

The screenshot shows a web browser window with the URL `localhost:9093/#/status`. The page has a header with links for Alertmanager, Alerts, Silences, Status, Settings, and Help. The main content starts with a large 

# Status

 heading. Below it, the **Uptime:** is listed as 2025-03-22T09:50:27.779Z. The next section is **Cluster Status**, which includes a **Name:** field containing the value 01JPYM5MDZEK0242YTKD7WJ5RR. Under **Status:**, there is a green button labeled "ready". The **Peers:** section lists one peer with the name 01JPYM5MDZEK0242YTKD7WJ5RR and address 192.168.1.55:9094. The final section is **Version Information**, which details the Branch (HEAD), BuildDate (20250307-15:04:11), BuildUser (root@2a402d3106ba), GoVersion (go1.23.7), Revision (b2099eaa2c9ebc25edb26517cb9c732738e93910), and Version (0.28.1).

# Status

**Uptime:** 2025-03-22T09:50:27.779Z

## Cluster Status

**Name:** 01JPYM5MDZEK0242YTKD7WJ5RR

**Status:** ready

**Peers:**

- **Name:** 01JPYM5MDZEK0242YTKD7WJ5RR  
**Address:** 192.168.1.55:9094

## Version Information

**Branch:** HEAD

**BuildDate:** 20250307-15:04:11

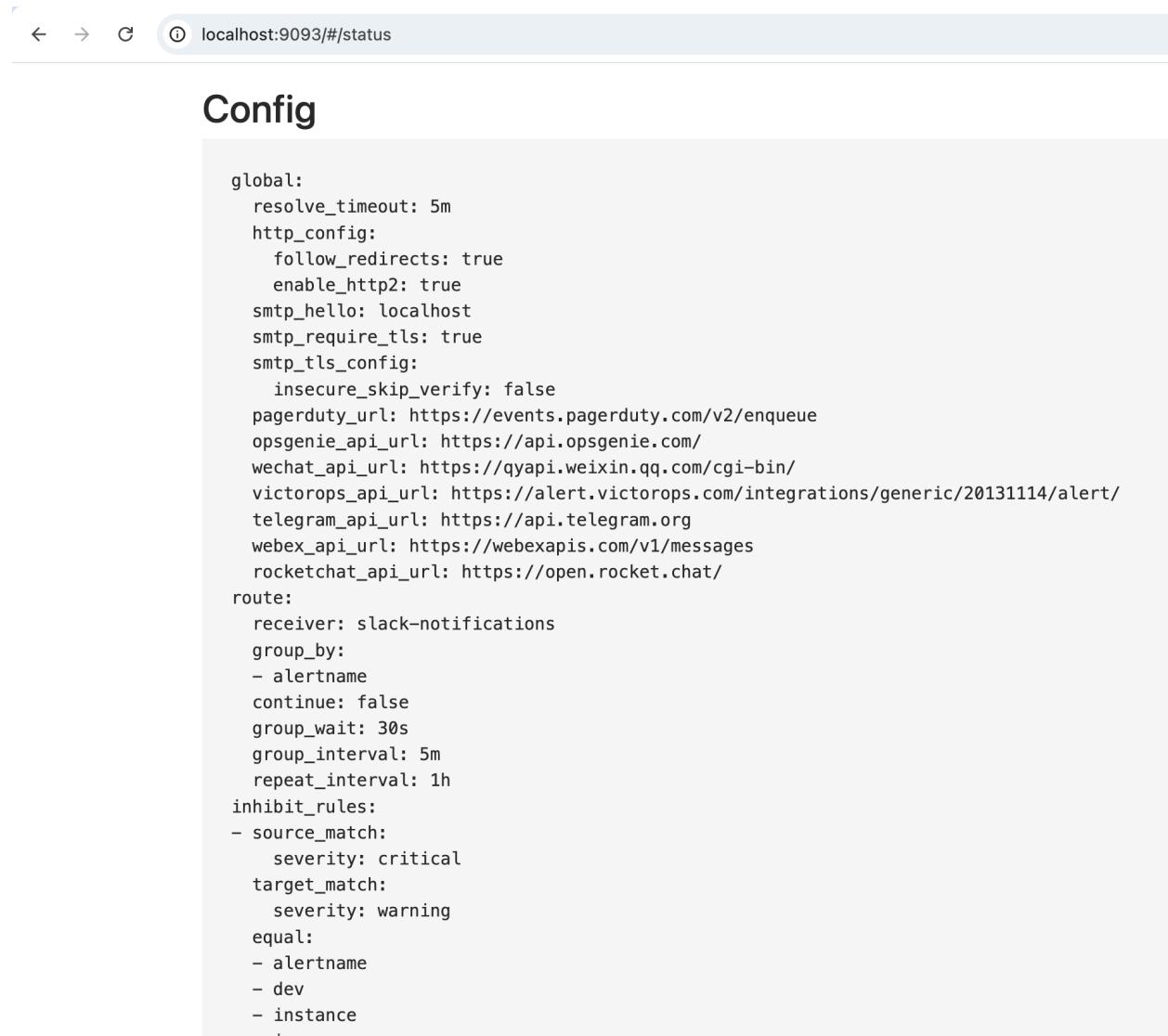
**BuildUser:** root@2a402d3106ba

**GoVersion:** go1.23.7

**Revision:** b2099eaa2c9ebc25edb26517cb9c732738e93910

**Version:** 0.28.1

18. Verify that the Alertmanager has been hooked to Slack Channel through Config.



```
global:
  resolve_timeout: 5m
  http_config:
    follow_redirects: true
    enable_http2: true
  smtp_hello: localhost
  smtp_require_tls: true
  smtp_tls_config:
    insecure_skip_verify: false
  pagerduty_url: https://events.pagerduty.com/v2/enqueue
  opsgenie_api_url: https://api.opsgenie.com/
  wechat_api_url: https://qyapi.weixin.qq.com/cgi-bin/
  victorops_api_url: https://alert.victorops.com/integrations/generic/20131114/alert/
  telegram_api_url: https://api.telegram.org
  webex_api_url: https://webexapis.com/v1/messages
  rocketchat_api_url: https://open.rocket.chat/
route:
  receiver: slack-notifications
  group_by:
  - alertname
  continue: false
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 1h
  inhibit_rules:
  - source_match:
      severity: critical
      target_match:
        severity: warning
      equal:
      - alertname
      - dev
      - instance
  receivers:
```

19. Use the `todo-app.yaml` version in Part 7 to explicitly hit the failing endpoint

`/test` in the two Probes.

20. Apply the intentional failure using: `kubectl apply -f kube/`

## 21. Check the Prometheus Dashboard to see if the custom alert is firing.

The screenshot shows the Prometheus Alerts interface at `localhost:9090/alerts`. A single alert is listed under the heading `pod-health-alerts` (`custom-alert.yml`):

**PodCrashLooping**

`rate(kube_pod_container_status_restarts_total[5m]) > 0`

`for: 2m`

`severity="critical"`

**description** The pod {{ \$labels.pod }} has restarted {{ \$value }} times in the last 5 minutes.

**summary** Pod {{ \$labels.pod }} in namespace {{ \$labels.namespace }} is crash looping

**Alert labels**

|                                                        | State  | Active Since | Value                             |
|--------------------------------------------------------|--------|--------------|-----------------------------------|
| <code>alertname="PodCrashLooping"</code>               | FIRING | 3m 28.603s   | <code>0.016684444698416328</code> |
| <code>container="todo-app"</code>                      |        |              |                                   |
| <code>instance="localhost:8080"</code>                 |        |              |                                   |
| <code>job="kube-state-metrics"</code>                  |        |              |                                   |
| <code>namespace="default"</code>                       |        |              |                                   |
| <code>pod="todo-app-dcd747687-r6pfr"</code>            |        |              |                                   |
| <code>severity="critical"</code>                       |        |              |                                   |
| <code>uid="1fe3dc83-e118-4c1e-9dbd-f248f1dddbc"</code> |        |              |                                   |

**description** The pod todo-app-dcd747687-r6pfr has restarted 0.016684444698416328 times in the last 5 minutes.

**summary** Pod todo-app-dcd747687-r6pfr in namespace default is crash looping

**Alert labels**

|                                                         | State  | Active Since | Value                             |
|---------------------------------------------------------|--------|--------------|-----------------------------------|
| <code>alertname="PodCrashLooping"</code>                | FIRING | 3m 28.603s   | <code>0.016684444698416328</code> |
| <code>container="todo-app"</code>                       |        |              |                                   |
| <code>instance="localhost:8080"</code>                  |        |              |                                   |
| <code>job="kube-state-metrics"</code>                   |        |              |                                   |
| <code>namespace="default"</code>                        |        |              |                                   |
| <code>pod="todo-app-dcd747687-9hvfd"</code>             |        |              |                                   |
| <code>severity="critical"</code>                        |        |              |                                   |
| <code>uid="1ce99b46-1e2a-4ec7-8f10-ad0d6c46f3f4"</code> |        |              |                                   |

## 22. Check the Alertmanager Dashboard to see if the Alert fired by Prometheus has been successfully send to Alertmanager.

The screenshot shows the Alertmanager dashboard at `localhost:9093/#/alerts`. Three alerts are listed under the heading `slack-notifications` (`alertname="PodCrashLooping"`):

**2025-03-22T09:56:52.405Z** [+ Info](#) [Source](#) [Silence](#) [Link](#)

`container="todo-app"` `instance="localhost:8080"` `job="kube-state-metrics"` `namespace="default"` `pod="todo-app-dcd747687-2fqvb"`

`severity="critical"` `uid="6cbd18a5-3afa-43db-a017-b9ec00bd37a1"`

**2025-03-22T09:56:52.405Z** [+ Info](#) [Source](#) [Silence](#) [Link](#)

`container="todo-app"` `instance="localhost:8080"` `job="kube-state-metrics"` `namespace="default"` `pod="todo-app-dcd747687-9hvfd"`

`severity="critical"` `uid="1ce99b46-1e2a-4ec7-8f10-ad0d6c46f3f4"`

**2025-03-22T09:56:52.405Z** [+ Info](#) [Source](#) [Silence](#) [Link](#)

`container="todo-app"` `instance="localhost:8080"` `job="kube-state-metrics"` `namespace="default"` `pod="todo-app-dcd747687-r6pfr"`

`severity="critical"` `uid="1fe3dc83-e118-4c1e-9dbd-f248f1dddbc"`

23. Check the Slack Channel to see if the Alert was successfully send by Alertmanager.



### Assignment 2 APP 5:57 AM

[FIRING:3] PodCrashLooping (todo-app localhost:8080 kube-state-metrics default critical)

Pod todo-app-dcd747687-2fqvb in namespace default is crash looping - The pod todo-app-dcd747687-2fqvb has restarted 0.013353689974576423 times in the last 5 minutes.

Pod todo-app-dcd747687-9hvfd in namespace default is crash looping - The pod todo-app-dcd747687-9hvfd has restarted 0.013353689974576423 times in the last 5 minutes.

Pod todo-app-dcd747687-r6pfr in namespace default is crash looping - The pod todo-app-dcd747687-r6pfr has restarted 0.013353689974576423 times in the last 5 minutes.