KEDA Example: Kubernetes Event-Driven Autoscaling Guide

A comprehensive guide for implementing Kubernetes Event-Driven Autoscaling (KEDA) with robust solutions for common Minikube local image pulling issues.

Overview

This guide provides step-by-step instructions to deploy and configure KEDA for event-driven autoscaling in a Minikube environment. The tutorial addresses common issues such as `ErrImagePull` problems with local Docker images and provides reliable solutions for running KEDA-based workloads.

Key Features Demonstrated

- **Event-driven autoscaling** using KEDA ScaledJobs
- **Redis-based message queue** for job triggering
- **CSV processing workload** as a practical example
- **Persistent volume management** for data storage
- **Local Docker image handling** in Minikube

Prerequisites

Before starting, ensure you have the following tools installed:

- **Minikube**: Installed and running (preferably with `--driver=docker`)
- **kubectl**: Configured to communicate with your Minikube cluster
- **git**: For cloning the project repository
- **Docker**: Available and accessible from your terminal

Architecture Overview

The tutorial demonstrates a complete event-driven processing pipeline:

1. **Data Generator**: Populates Redis queue with CSV processing tasks
2. **Redis**: Acts as the message queue for job coordination
3. **KEDA**: Monitors Redis queue and triggers scaling events
4. **CSV Processor**: Scaled job that processes CSV files from the queue
5. **Persistent Volume**: Shared storage for processed data

Step-by-Step Implementation

1. Clone the Project Repository

bash

```
git clone https://github.com/digital-power/tutorial-kubernetes-event-driven-autoscaling.git
cd tutorial-kubernetes-event-driven-autoscaling
```

2. Start Minikube & Configure Docker Environment

Critical Step: Configure your local Docker commands to use Minikube's internal Docker daemon. This ensures built images are immediately available within the cluster.

bash

```
# Start Minikube (if not already running)
minikube start --driver=docker

# IMPORTANT: Configure your shell to use Minikube's Docker daemon
eval $(minikube docker-env)
```

> **Note**: You must run `eval $(minikube docker-env)` in each new terminal session where you want to build Docker images for Minikube.

3. Install KEDA

Install the KEDA core components into your cluster:

bash

```
kubectl apply -f https://github.com/kedacore/keda/releases/download/v2.10.1/keda-2.10.1-core.yaml
```

Verify KEDA installation:

bash

```
kubectl get pods -n keda
# All KEDA pods (e.g., keda-operator, keda-metrics-apiserver) should show 'Running' status
```

4. Deploy Redis (Message Queue)

Deploy Redis as the message queue backend:

bash

```
kubectl apply -f manifests/deployments/redis-deployment.yaml
kubectl apply -f manifests/services/redis-service.yaml
```

Verify Redis deployment:

```bash
kubectl get deployments redis
kubectl get services redis
# Both should show '1/1 Ready' or similar healthy status
```

5. Set up Persistent Volume Claim (PVC)

The CSV processor requires persistent storage for processed files:

```bash
# Apply the Persistent Volume (if needed)
kubectl apply -f manifests/volumes/data-pv.yaml

# Apply the Persistent Volume Claim
kubectl apply -f manifests/volumes/data-pvc.yaml
```

Note: If `data-pv.yaml` doesn't exist, create it with the following content:

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: data-pv
spec:
  capacity:
    storage: 1Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: "/mnt/data" # Minikube provides hostPath on its VM
```

Verify PVC binding:

bash

```
kubectl get pvc data-pvc
# Status should be 'Bound'
```

6. Build Python Docker Image for CSV Processor

Critical Change: Build the `csvprocessor` image directly into Minikube's Docker daemon:

bash

```
# Ensure you are in the project root directory
# Re-run eval $(minikube docker-env) if you opened a new terminal
eval $(minikube docker-env)

docker build -t csvprocessor:latest -f csv_processor.Dockerfile .
```

Verify the image is available:

bash

```
docker images
# You should see 'csvprocessor' with tag 'latest' listed
```

7. Deploy the Scaled Job (with Image Pull Policy Fix)

Critical Change: Modify the ScaledJob manifest to prevent image pull issues.

Edit `manifests/jobs/csv-processor-scaled-jobs.yaml` and locate the containers section. Add `imagePullPolicy: Never` below the image line:

yaml

```
  containers:
  - name: csv-processor
    image: csvprocessor:latest
    imagePullPolicy: Never # <--- ADD THIS LINE!
    env:
    - name: REDIS_HOST
      value: redis
    - name: REDIS_LIST
      value: csvs-to-process
    resources:
      limits:
        cpu: "0.2"
        memory: "100Mi"
      requests:
        cpu: "0.2"
        memory: "100Mi"
    volumeMounts:
    - name: data-volume
      mountPath: /app/data
```

Apply the modified ScaledJob:

```
bash

kubectl apply -f manifests/jobs/csv-processor-scaled-jobs.yaml
```

8. Generate Data and Trigger Scaling

Clean up any existing jobs and trigger the processing workflow:

```
bash

# Delete any old csv-processor jobs
kubectl get jobs -l scaledjob.keda.sh/name=csv-processor -o name | xargs -r kubectl delete

# Delete any old data-generator jobs
kubectl delete job data-generator --ignore-not-found=true

# Apply the data-generator to push messages to Redis
kubectl apply -f manifests/jobs/data-generator.yaml
```

Watch the scaling in action:

```bash
kubectl get pods -w
```

You should observe:

1. A `data-generator` pod running and completing
2. New `csv-processor-...` pods appearing as KEDA scales up
3. Pods transitioning: `Pending` → `ContainerCreating` → `Running` → `Completed`
4. **No `ErrImagePull` or `ImagePullBackOff` errors**

9. Verification and Cleanup

Verify Redis Queue is Empty

```bash
kubectl exec -it $(kubectl get pod -l app=redis -o jsonpath='{.items[0].metadata.name}') -- redis-cli LLEN csvs-to-proces
# Should return (integer) 0 once all processing is complete
```

List All Jobs

```bash
kubectl get jobs
# All relevant jobs (data-generator, csv-processor-...) should be 'Completed'
```

Cleanup Resources

When finished, clean up all deployed resources:

```bash
```

```
kubectl delete -f manifests/jobs/data-generator.yaml --ignore-not-found=true
kubectl delete -f manifests/jobs/csv-processor-scaled-jobs.yaml --ignore-not-found=true
kubectl delete -f manifests/services/redis-service.yaml --ignore-not-found=true
kubectl delete -f manifests/deployments/redis-deployment.yaml --ignore-not-found=true
kubectl delete -f manifests/volumes/data-pvc.yaml --ignore-not-found=true
kubectl delete -f manifests/volumes/data-pv.yaml --ignore-not-found=true
kubectl delete -f https://github.com/kedacore/keda/releases/download/v2.10.1/keda-2.10.1-core.yaml --ignore-not-fo

# Optionally stop/delete minikube
minikube stop
# or
minikube delete
```

Key Solutions Implemented

This guide addresses common issues encountered in KEDA tutorials:

1. Local Image Pull Problems

- **Solution**: Use `eval $(minikube docker-env)` to build images directly into Minikube's Docker daemon
- **Implementation**: Set `imagePullPolicy: Never` in Kubernetes manifests

2. Job Re-execution Issues

- **Solution**: Properly delete existing jobs before reapplying
- **Implementation**: Use `kubectl delete` with appropriate selectors

3. Registry Configuration Confusion

- **Solution**: No need for `minikube addons enable registry` when using `minikube docker-env`
- **Implementation**: Direct Docker daemon usage eliminates registry complexity

Troubleshooting

Common Issues and Solutions

1. **ErrImagePull errors**:
   - Ensure `eval $(minikube docker-env)` was run in the current terminal
   - Verify `imagePullPolicy: Never` is set in the manifest
   - Check that the image exists with `docker images`
2. **Jobs not scaling**:

- Verify KEDA pods are running: `kubectl get pods -n keda`
- Check Redis connectivity: `kubectl logs -l app=redis`
- Ensure ScaledJob is applied correctly: `kubectl get scaledjobs`

3. **PVC binding issues**:
   - Check PV availability: `kubectl get pv`
   - Verify storage class: `kubectl get storageclass`
   - Review PVC status: `kubectl describe pvc data-pvc`

Best Practices

1. **Always use** `eval $(minikube docker-env)` in terminal sessions where you build images
2. **Set** `imagePullPolicy: Never` for locally built images
3. **Clean up jobs** before re-running to avoid conflicts
4. **Monitor pod status** with `kubectl get pods -w` to observe scaling behavior
5. **Verify component health** at each step before proceeding

Conclusion

This guide provides a robust foundation for implementing KEDA-based event-driven autoscaling in Kubernetes. By following these steps and understanding the key solutions for common issues, you'll have a working example that demonstrates the power of event-driven scaling in cloud-native applications.

The tutorial showcases how KEDA can automatically scale workloads based on external metrics (Redis queue length), providing an efficient and cost-effective approach to handling variable workloads in Kubernetes environments.