Kubernetes Automatic Scaling and Monitoring Documentation

Table of Contents

1. Overview

This automation tool provides continuous monitoring and automatic scaling of Kubernetes deployments based on resource utilization. It helps maintain optimal performance by automatically adjusting the number of replicas based on CPU and memory usage.

Key Features
- Automatic resource monitoring
- Dynamic scaling based on configurable thresholds
- Real-time metrics dashboard
- Historical metrics logging
- Color-coded status outputs
- Configurable scaling parameters

2. Prerequisites

Required components:
—>Kubernetes CLI

```
kubectl version --client
```

->Basic Calculator (for arithmetic operations)

```
bc -v
```

Required Kubernetes permissions

- kubectl auth can-i scale deployment
- kubectl auth can-i get pods
- kubectl auth can-i get deployments

3. Installation

1. Download the script:

```
curl -O https://raw.githubusercontent.com/your-repo/k8s-auto-scale.sh
```

2. Make it executable:

```
chmod +x k8s-auto-scale.sh
```

3. Verify installation:

```
./k8s-auto-scale.sh --version
```

4. Configuration

->Default Parameters

```
CPU_THRESHOLD=80        # CPU usage percentage threshold
MEMORY_THRESHOLD=80     # Memory usage percentage threshold
MAX_REPLICAS=10         # Maximum number of replicas
MIN_REPLICAS=2          # Minimum number of replicas
CHECK_INTERVAL=30       # Monitoring interval in seconds
```

Custom Configuration
You can override defaults during script execution:

```
Enter CPU threshold % (default 80): 70
Enter Memory threshold % (default 80): 75
Enter minimum replicas (default 2): 3
Enter maximum replicas (default 10): 8
Enter check interval in seconds (default 30): 45
```

5. Usage Guide

->Basic Usage

```
./k8s-auto-scale.sh
```

Required Inputs

```
Enter namespace: production
Enter deployment name: web-app
```

6. Monitoring Features

Real-time Metrics
- CPU usage percentage
- Memory usage percentage
- Current replica count
- Resource utilization trends

Dashboard Display

Current Metrics Summary:

```
----------------------------------------
Last 5 measurements:
2024-11-18 10:30:00 CPU: 45% Memory: 60% Replicas: 3
2024-11-18 10:30:30 CPU: 48% Memory: 62% Replicas: 3
2024-11-18 10:31:00 CPU: 82% Memory: 65% Replicas: 4
2024-11-18 10:31:30 CPU: 75% Memory: 63% Replicas: 4
2024-11-18 10:32:00 CPU: 70% Memory: 61% Replicas: 4
----------------------------------------
```
```` ``` ````

7. Scaling Logic

Scale Up Conditions

```
if (CPU_Usage > CPU_THRESHOLD) OR (Memory_Usage > MEMORY_THRESHOLD):
    if (Current_Replicas < MAX_REPLICAS):
        Scale Up by 1 replica
```

Scale Down Conditions

```
if (CPU_Usage < CPU_THRESHOLD/2) AND (Memory_Usage <
MEMORY_THRESHOLD/2):
    if (Current_Replicas > MIN_REPLICAS):
        Scale Down by 1 replica
```
````

8. Logging and Metrics

Log File Format

```
metrics_${namespace}_${deployment}.log
```

Sample Log Entry

```
2024-11-18 10:30:00 CPU: 45% Memory: 60% Replicas: 3
```

Metrics Collected
- Timestamp
- CPU usage percentage
- Memory usage percentage
- Number of replicas
- Scaling events

9. Troubleshooting

Common Issues and Solutions

1. Permission Issues

Error: "cannot get deployments"
Solution: Ensure proper RBAC permissions

2. Resource Metrics Unavailable

Error: "metrics not available"
Solution: Verify metrics-server is running

3. Scaling Failures

Error: "scaling deployment failed"
Solution: Check deployment configuration and resource quotas

10. Best Practices

Resource Thresholds
- Set CPU threshold based on application behavior
- Consider memory usage patterns
- Adjust thresholds during peak hours

Scaling Configuration
- Set appropriate MIN_REPLICAS for HA
- Configure MAX_REPLICAS based on cluster capacity

- Use reasonable CHECK_INTERVAL (30-60 seconds)

Monitoring
- Regular log review
- Track scaling patterns
- Monitor application performance

Maintenance
- Regular script updates
- Log rotation
- Performance tuning

Health Checks

The script performs the following health checks:

1. Deployment Health

```
kubectl rollout status deployment/$deployment -n $namespace
```

2. Resource Availability

```
kubectl top pods -n $namespace
```

3. Scaling Operations

```
kubectl scale deployment $deployment --replicas=$count -n $namespace
```

Security Considerations

1. RBAC Permissions

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: deployment-scaler
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "update", "patch"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
```

2. Namespace Isolation
- Use namespace-scoped roles
- Limit access to specific deployments

- Monitor scaling operations

Limitations

1. Resource Constraints
- Dependent on metrics-server availability
- May have delayed scaling during heavy load
- Limited by cluster resources

2. Scaling Boundaries
- Minimum replica count enforced
- Maximum replica count enforced
- Scaling step size fixed at 1

Support and Maintenance

Updates and Patches
- Regular script updates
- Bug fixes
- Feature enhancements

Monitoring and Alerts
- Resource usage alerts
- Scaling event notifications
- Error reporting

This documentation provides a detail guide for using the automatic scaling script.