Improving Deployment Configuration for High Availability

This section focuses on improving a Kubernetes deployment for a simple Go application to ensure high availability, efficient resource utilization, and better monitoring with health probes.

Overview

The application is a basic "Hello World" web server written in Go, with endpoints to handle liveness and readiness probes. This document outlines the improvements made to the deployment configuration to enhance reliability and scalability, as well as changes to the Go application for logging probe requests.

Improvements Made

1. **Increased Replicas:** Enhanced availability and redundancy by increasing the number of pod replicas
2. **Resource Requests and Limits:** Defined resource requests and limits to ensure optimal resource usage.
3. **Liveness and Readiness Probes:** Added health checks to ensure only healthy pods are available.
4. **Rolling Update Strategy:** Configured a rolling update strategy for zero downtime during deployments.
5. **Pod Disruption Budget:** Introduced a Pod Disruption Budget to maintain availability during node maintenance or voluntary disruptions.
6. **Logging for Probes:** Modified the Go application to log incoming liveness and readiness probe requests.

Go Application Changes

The Go application has been updated to include logging for liveness and readiness probes. This helps in monitoring the frequency and success of these checks.

Updated Go Code

```
package main

import (
    "fmt"
    "log"
    "net/http"
)

// helloWorldHandler responds with "Hello World" to any HTTP request.
func helloWorldHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello World")
}

// healthCheckHandler logs and responds to liveness probe requests.
func healthCheckHandler(w http.ResponseWriter, r *http.Request) {
    log.Println("Liveness probe hit")
    fmt.Fprintln(w, "OK")
```

```
}

// readinessCheckHandler logs and responds to readiness probe requests.
func readinessCheckHandler(w http.ResponseWriter, r *http.Request) {
    log.Println("Readiness probe hit")
    fmt.Fprintln(w, "OK")
}

func main() {
    // Register the helloWorldHandler to handle requests to the root URL path.
    http.HandleFunc("/", helloWorldHandler)

    // Register handlers for liveness and readiness probes.
    http.HandleFunc("/health", healthCheckHandler)
    http.HandleFunc("/ready", readinessCheckHandler)

    // Print a message indicating the server is starting on port 8080.
    log.Println("Starting server at port 8080")

    // Start the HTTP server on port 8080 and log any errors if the server fails
to start.
    if err := http.ListenAndServe(":8080", nil); err != nil {
        log.Fatalf("Error starting server: %s", err)
    }
}
```

Dockerfile Changes

The Dockerfile has been updated to use `alpine` instead of `scratch` for easier debugging, including installing bash and curl.

Updated Dockerfile

```
# Stage 1: Build the Go Application
FROM golang:1.22.5-alpine AS builder

# Set the working directory
WORKDIR /app

# Copy the app directory contents into the container at /app
COPY app /app

# Build the Go app
RUN go build -o hello-world .

# Stage 2: Create a lightweight image for running the application
# Use alpine for debugging purposes instead of scratch
FROM alpine

# Set the working directory inside the container
WORKDIR /root/
```

```
# Install bash and curl for debugging
RUN apk add --no-cache bash curl

# Copy the compiled binary from the builder stage to the current stage
COPY --from=builder /app/hello-world .

# Expose port 8080 to the outside world
EXPOSE 8080

# Command to run the executable
CMD ["./hello-world"]
```

Kubernetes Configuration Changes

Updated Deployment Configuration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world2-deployment
spec:
  replicas: 3 # Increased replicas for higher availability
  selector:
    matchLabels:
      app: hello-world2
  strategy:
    type: RollingUpdate # Use rolling update strategy
    rollingUpdate:
      maxUnavailable: 1 # Allow only 1 pod to be unavailable during updates
      maxSurge: 1 # Allow 1 additional pod to be created during updates
  template:
    metadata:
      labels:
        app: hello-world2
    spec:
      containers:
      - name: hello-world2
        image: bhavinprajapti/ha-hello-world:1.0
        ports:
        - containerPort: 8080
        resources:
          requests:
            cpu: "100m" # Request minimum CPU resources
            memory: "128Mi" # Request minimum memory resources
          limits:
            cpu: "500m" # Limit maximum CPU usage
            memory: "256Mi" # Limit maximum memory usage
        livenessProbe:
          httpGet:
            path: /health
```

```
            port: 8080
          initialDelaySeconds: 10 # Wait 10 seconds before starting liveness
checks
          periodSeconds: 10 # Check liveness every 10 seconds
        readinessProbe:
          httpGet:
            path: /ready
            port: 8080
          initialDelaySeconds: 5 # Wait 5 seconds before starting readiness checks
          periodSeconds: 10 # Check readiness every 10 seconds
```

Service Configuration

```
apiVersion: v1
kind: Service
metadata:
  name: hello-world2-service
spec:
  type: NodePort # Use NodePort to expose service on each node's IP
  selector:
    app: hello-world2
  ports:
    - protocol: TCP
      port: 80 # Expose service on port 80
      targetPort: 8080 # Forward to container port 8080
```

Pod Disruption Budget Configuration

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: hello-world2-pdb
spec:
  minAvailable: 2 # Ensure at least 2 pods are available at any time
  selector:
    matchLabels:
      app: hello-world2
```

k3s Installation and Deploy application

This section guides you through installing K3s, a lightweight Kubernetes distribution, and deploying the application on it.

Prerequisites

- A Unix-based system.
- Sudo privileges.

Setup

1. copy the `installAndDeploy2.sh` into a directory on your local machine.
2. Navigate to the project directory.

Running the Script

1. Make the script executable:

```
chmod +x installAndDeploy2.sh
```

2. Run the script:

```
./installAndDeploy2.sh
```

You should see "Hello World" displayed.

3. Access Logs:

```
sudo kubectl logs <pod-name>
```

You should see log entries indicating when the liveness and readiness probes are hit.

4. verify probe status: Check that the application responds correctly to /health and /ready endpoints to ensure that probes are working as expected.

```
curl http://<Node_IP>:<NodePort>/ready
curl http://<Node_IP>:<NodePort>/health
```

You should see "OK" displayed, which indicates that the probes are functioning correctly.