Gmail

Sandhya Babu <sandhya.bca7@gmail.com>

Day25 Challenge: Understanding Deployments and Application Management

1 message

Sagar Utekar <getfitwithsagar2366@gmail.com> Fri, Dec 27, 2024 at 8:00 AM
Bcc: sandhya.bca7@gmail.com

Hello Learners,

Welcome back to another thrilling episode of the DevOps SRE Daily Challenge!
This challenge focuses on **Workloads in Kubernetes**, specifically targeting **deployments**, rolling updates, rollbacks, scaling applications, and creating robust, self-healing application deployments.

Understanding Kubernetes Deployments and Daemonsets

What is a Deployment?

A **Deployment** in Kubernetes is a controller that manages the lifecycle of Pods using a declarative configuration. It ensures your application runs reliably by automatically scaling, updating, and recovering Pods as needed.

Key Features and Use Cases of Deployments

1. **Declarative Updates**
   - Specify the desired application state (e.g., number of replicas, container images).
   - Use Case: Create a Deployment to manage ReplicaSets and ensure Pods match the declared state automatically.
2. **Rolling Updates**
   - Safely deploy updates to applications with minimal downtime.

- Use Case: Gradually roll out a new version of an application, controlling update behavior with `maxUnavailable` and `maxSurge` to maintain availability.

3. **Rollbacks**

- Quickly revert to a previous stable state if issues occur during an update.
- Use Case: Roll back to an earlier Deployment revision when a new release introduces critical errors, ensuring the system remains stable.

4. **Scaling**

- Dynamically adjust the number of replicas to handle varying workloads.
- Use Case: Scale up an application to handle increased traffic during peak hours or scale down during low traffic periods to save resources.

5. **Self-Healing**

- Automatically replace failed Pods to maintain the desired state and ensure application availability.
- Use Case: When a Pod crashes or a node fails, the Deployment automatically recreates the necessary Pods on healthy nodes.

6. **Pause and Resume Rollouts**

- Temporarily halt a Deployment rollout to make multiple changes before resuming.
- Use Case: Pause a rollout to apply fixes to the application’s PodTemplateSpec and test in a controlled environment before resuming the rollout.

7. **Clean Up Old ReplicaSets**

- Automatically manage and clean up unused ReplicaSets.
- Use Case: Retain only the most recent revisions of the Deployment, ensuring the cluster remains uncluttered and resource-efficient.

Deployment YAML File Example

```
apiVersion: apps/v1                    # Specifies the API version for the Deployment
resource.
kind: Deployment                       # Defines the resource type. In this case, it's a
Deployment.
metadata:
  name: my-deployment                  # Unique name of the Deployment within the
namespace.
  namespace: default                   # (Optional) Namespace where the Deployment
resides. Defaults to 'default'.
  labels:
    app: my-app                        # Key-value pair to categorize and identify the
Deployment.
spec:
  replicas: 3                          # Desired number of Pod replicas to maintain.
  selector:
    matchLabels:
      app: my-app                      # Label selector to identify Pods managed by this
Deployment.
  template:
    metadata:
      labels:
        app: my-app                    # Labels assigned to the Pods created by this
Deployment.
    spec:
      containers:
      - name: my-container             # Name of the container.
        image: nginx:latest            # Container image to be used.
        ports:
        - containerPort: 80            # Port the container listens on.
        resources:
          requests:
            cpu: "100m"                # Minimum CPU resources required by the
container.
```

```
            memory: "128Mi"          # Minimum memory required by the container.
          limits:
            cpu: "250m"              # Maximum CPU resources the container can use.
            memory: "256Mi"          # Maximum memory the container can use.
```

Challenge: Real-World Deployments with Kubernetes (Yelb UI Focus)

GitHub Repository for the Challenge:

The challenge uses the Yelb application, which has the following components:

- Yelb UI (frontend - nginx)
- Yelb Appserver (backend logic - Ruby)
- Yelb DB (database - PostgreSQL)
- Redis Cache

Task 1: Clone, Build, and Deploy Yelb (Version 1)

Objective: Build container images and deploy the Yelb application.

- Clone the Yelb repository.
- Build container images for `yelb-ui`, `yelb-appserver`, and `yelb-db` using the Dockerfile provided in the repository.
- Push the images to a container registry (DockerHub, ECR, or GCR).
- Create Kubernetes Deployments and Services to deploy Yelb in the `yelb` namespace.

Deliverable: A fully running Yelb application accessible via a Kubernetes service.

Task 2: Add Health Checks to All Components

Objective: Ensure all Yelb components have proper readiness and liveness probes.

- Add a readinessProbe for each Deployment to ensure Pods are ready before accepting traffic.
- Add a livenessProbe to restart unresponsive containers.
- Deploy the updated configurations and verify that the probes work as expected.

Deliverable: Deployments with health checks implemented and validated using logs.

Task 3: Enable RBAC and Service Accounts

Objective: Restrict access to Yelb Deployments and ensure secure operations.

- Create a service account `yelb-sa` in the `yelb` namespace.
- Create an RBAC Role allowing access to Deployments in the namespace.
- Bind the Role to `yelb-sa`.
- Modify the Deployments to use the `yelb-sa` service account.

Deliverable: Secure Yelb Deployments using RBAC and service accounts.

Task 4: Deploy Yelb Using a Multi-Container Pod

Objective: Deploy Yelb UI and a sidecar container for logging in the same Pod.

- Modify the Yelb UI Deployment to include a sidecar container (e.g., Fluentd) for log aggregation.
- Use an `emptyDir` volume for log sharing between the main and sidecar containers.
- Deploy the updated configuration and verify logs aggregation in the sidecar.

Deliverable: A multi-container Pod for Yelb UI with centralized logging.

Task 5: Perform a Rolling Update for Yelb UI

Objective: Update Yelb UI (frontend) with a new version and ensure the update is applied with zero downtime using a rolling update strategy.

- Modify the Yelb UI code to include a new feature or UI change (e.g., update the title or add a section).
- Build and push the updated `yelb-ui:v2` image.
- Update the Deployment to use the new image and perform a rolling update.
- Monitor the rollout process to ensure no downtime occurs during the update.

Deliverable: A smoothly updated Yelb UI deployment with no downtime for users.

References:

- **Kubernetes Deployment Documentation**

Kubernetes Deployments: Essential Commands

1. Deployment Management

- **Create a Deployment (imperative):**

  ```
  kubectl create deployment my-deployment --image=nginx:latest
  ```

- **Apply a Deployment file (declarative):**

  > kubectl apply -f deployment.yaml

- **View details of the Deployment:**

  > kubectl describe deployment my-deployment

- **List Deployments:**

  > kubectl get deployments

- **Delete a Deployment:**

  > kubectl delete deployment my-deployment

2. Scaling a Deployment

- **Scale using an imperative command:**

  > kubectl scale deployment my-deployment --replicas=5

- **Scale using declarative YAML updates:**
  Modify `replicas` in `deployment.yaml` and apply changes:

  > `kubectl apply -f deployment.yaml`

3. Rolling Updates

- **Update Deployment (imperative):**

  > kubectl set image deployment my-deployment my-container=nginx:1.21 --record

- **Check rollout status:**

  > kubectl rollout status deployment my-deployment

4. Rollbacks

- **View Deployment history:**

  > kubectl rollout history deployment my-deployment

- **Roll back to the previous revision:**

  > kubectl rollout undo deployment my-deployment

- **Roll back to a specific revision:**

  > kubectl rollout undo deployment my-deployment --to-revision=2

5. Pause and Resume a Rollout

- **Pause a rollout:**

  > kubectl rollout pause deployment my-deployment

- **Resume a rollout:**

  > kubectl rollout resume deployment my-deployment

6. Debugging and Cleanup

- **View ReplicaSets managed by a Deployment:**

  > kubectl get rs

- **Delete unused ReplicaSets:**

  > kubectl delete rs <replicaset-name>

Submission Guidelines

Theory Section

- Explain key concepts like Kubernetes Deployments, rolling updates, and self-healing.
- Describe the use of RBAC and Service Accounts in securing workloads.

Screenshots or Outputs

1. **Deployment Commands:**
   - Screenshots of `kubectl create deployment`, `kubectl apply`, `kubectl get deployments`, and `kubectl rollout status`.
2. **Multi-Container Pod:**
   - Imperative and YAML-based steps for deploying a sidecar container.
   - `kubectl describe pod` showing the sidecar and shared volume.
3. **Probes & Resource Limits:**

- Screenshots of liveness/readiness probes and resource limits in YAML.
- `kubectl describe pod` validating probes and limits.

4. **Rolling Update/Rollback:**
    - Successful rollout (`kubectl rollout status`), history (`kubectl rollout history`), and rollback (if needed).

Documentation

- Briefly document your approach and challenges faced with multi-container Pods, rolling updates, and RBAC.

Social Media Post

- Share your progress with hashtags:
  #getfitwithsagar #SRELife #DevOpsForAll #ckawithsagar

If you missed any previous challenges, you can catch up by reviewing the problem statements on GitHub.

Best regards,
Sagar Utekar