Task 3: Scaling Concerns and Implementations

This document outlines the current implementations and best practice recommendations related to scaling and security for both the Application and Database layers within our Kubernetes environment.

Application Layer

Implemented Features

1. **Horizontal Pod Autoscaling (HPA)**
    - Automatically adjusts the number of application pods based on CPU utilization to maintain optimal resource usage and service availability during varying traffic loads.
    - Requires the Kubernetes Metrics Server, which is already deployed and operational.
2. **Init Container for Dependency Management**
    - Ensures that application pods start only after the successful completion of database migration jobs.
    - This guarantees that the required schema and data are available, preventing runtime errors related to missing dependencies.
3. **Ingress Controller for External Access**
    - Configured with the NGINX Ingress Controller to manage incoming traffic to the application.
    - Provides critical features such as routing, SSL termination, and centralized traffic control.
    - The Ingress Controller is automatically installed via the `setup.sh` script if it is not already present in the cluster.
4. **Scoped Access via Role & RoleBinding**
    - Application pods are assigned dedicated ServiceAccounts with limited permissions governed by Role-Based Access Control (RBAC).
    - This restricts pod access within the cluster to only necessary resources, minimizing potential security risks.

Recommendations

- **Multi-AZ Deployments**
  To improve resilience and fault tolerance, distribute application replicas across multiple Availability Zones using Kubernetes features such as `topologySpreadConstraints` or cloud provider-specific configurations.
- **Security Context for Pods and Containers**
  Enhance container isolation and security by enforcing best practices like:
    - Running containers as non-root users (`runAsNonRoot: true`)
    - Disabling privilege escalation
    - Using read-only root file systems
- **File-Based Secrets Management**
  Replace environment variable secrets with Kubernetes Secrets mounted as files. This

approach reduces accidental exposure of sensitive information through logs or memory dumps and aligns with security best practices.

Database Layer (PostgreSQL)

Implemented Features

1. **StatefulSet for PostgreSQL**
   - Deploys PostgreSQL pods with stable network identities and persistent storage, ensuring ordered and reliable startup and scaling.
2. **PersistentVolumeClaim (PVC)**
   - Provides durable storage attached to the StatefulSet, preserving database state and data through pod restarts or rescheduling.
3. **ClusterIP Service**
   - Exposes PostgreSQL internally within the Kubernetes cluster via DNS, preventing external access by default and reducing attack surface.
4. **Credentials Stored in Kubernetes Secrets**
   - Database credentials (username and password) are securely stored in Kubernetes Secrets and injected into pods, minimizing the risk of credential leakage.

Recommendations

- **Use Managed PostgreSQL Services for Production**
  For production workloads, consider migrating to managed database services such as AWS RDS, Google Cloud SQL, or Azure Database for PostgreSQL. These platforms offer features like automated backups, high availability (HA), monitoring, and easier vertical scaling.
- **Automated Backup and Restore Strategy**
  Implement regular backups using tools such as `pg_dump` combined with Kubernetes CronJobs and external object storage solutions (e.g., AWS S3, Google Cloud Storage). Consider using volume snapshot tools like Velero for more comprehensive backups. Regularly test restore procedures to ensure data integrity.
- **Vertical Scaling for Non-Production Environments**
  Since PostgreSQL horizontal scaling is complex, non-production environments can benefit from vertical scaling (increased CPU/RAM). For production, use connection pooling (e.g., PgBouncer) or advanced techniques like partitioning or sharding when handling high load.
- **Database Pod Security Context and Hardening**
  Apply security hardening on PostgreSQL pods by enforcing:
  - `runAsNonRoot: true`
  - `readOnlyRootFilesystem: true`
  - Dropping unnecessary Linux capabilities to minimize the attack surface.
