Daily Tasks

- Cluster Health Monitoring
  - ✓ Monitor pod health (kubectl get pods --all-namespaces)
  - ✓ Review resource usage (CPU, memory)
- Log Review & Troubleshooting
  - ✓ Inspect logs for failed pods (kubectl logs <pod-name>)
  - ✓ Investigate CrashLoopBackOff or ImagePullBack errors
  - ✓ Use tools like kubectl describe and kubectl events for debugging
- Deployment :
  - ✓ Deploy new application versions using kubectl apply or Helm & new builds.
- Resource Management
  - ✓ Review and adjust resource requests/limits
  - ✓ Clean up unused resources
- Automation & CI/CD Integration
  - ✓ Monitor CI/CD pipelines for Kubernetes deployments
  - ✓ Ensure ArgoCD tools are syncing correctly
  - ✓ Monitor Application Sync Status
  - ✓ Check if applications are in sync with Healthy and Synced status.
- Review and Apply Git Changes
  - ✓ Monitor Git repositories for new commits.
  - ✓ Ensure changes are reflected in the cluster.
  - ✓ Trigger manual sync if auto-sync is disabled
- Handle Sync Failures
  - ✓ Investigate failed syncs (e.g., due to YAML errors, missing resources).
  - ✓ Use argocd app logs or check the UI for error messages.
- Manage Secrets and Configs
  - ✓ Ensure secrets are handled securely (e.g., using Sealed Secrets or External Secrets).
  - ✓ Validate config maps and environment variables.
- Container Health Checks
  - ✓ List running containers

    docker ps
- After exec in any container , later on connect to dev env :
  - ✓ Inspect Logs or Files
  - ✓ View application logs
  - ✓ Resource Usage
- Monitor CPU/memory:

  1.top

2. free -m

Docker image resize :

- Use a Slim Base Image :
  - ✓ Replace python:3.x with python:3.x-slim or python:3.x-alpine.
  - ✓ Alpine is smaller but may require extra setup for dependencies.
    - FROM python:3.9-slim
- Install Only Required Packages
  - ✓ Use a minimal requirements.txt.
  - ✓ Avoid unnecessary packages and dev dependencies.
    - COPY requirements.txt .
    - RUN pip install --no-cache-dir -r requirements.txt
- Remove Cache and Temporary Files
  - ✓ Use --no-cache-dir with pip.
  - ✓ Clean up build tools if installed temporarily.

```
RUN pip install --no-cache-dir -r requirements.txt && \
        apt-get clean && \
        rm -rf /var/lib/apt/lists/*
```

- Use Multi-Stage Builds (Optional)
  - ✓ Build dependencies in one stage, copy only necessary files to final image.

```
# Stage 1: Build
FROM python:3.9-slim as builder
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
# Stage 2: Final image
FROM python:3.9-slim
WORKDIR /app
COPY --from=builder /usr/local/lib/python3.11/site-packages
/usr/local/lib/python3.11/site-packages
COPY . .
CMD ["python", "app.py"]
```

- Minimize Layers
  - ✓ Combine commands to reduce image layers.

```
RUN apt-get update && apt-get install -y \
        build-essential \
        && apt-get clean \
        && rm -rf /var/lib/apt/lists/*
```

- Use .dockerignore File
  - ✓ Prevent copying unnecessary files (e.g., .git, tests, docs) into the image.

```
# .dockerignore
.git
__pycache__/
tests/
docs/
*.md
```

- Tasks to Run a Kubernetes Job in Dev

Step 1: Write the Job YAML

Create a YAML file defining the job in locally and then push into bitbucket

Clone the git repo into VM or We can directly write a file in VM.

Step 2: Apply the Job to the Cluster

Use helm-command to apply the job:

```
helm upgrade --install helm-release -n namespace . --debug -f
values.yaml
```

Step 3: Monitor Job Status

Check if the job is running or completed:

```
kubectl get jobs
kubectl describe job sample-job
```

Step 4: View Logs

Once the pod is created, view its logs:

```
kubectl logs job/sample-job
```

Step 5: Clean Up (Optional)

Delete the job and its pods after completion:

```
kubectl delete job sample-job
```