**Docker, CI/CD and Kubernetes setup Instructions**

**Developer → GitHub**

* **Technology:** git
* **How it works:** You push code with git push. GitHub stores your repo and triggers a CI/CD workflow.  
  CI/CD is configures using CI/CD .yml file that contains the relevant command to create the images and push to the GitHub Container Registry(GHCR)
* **Communication:** Local git client → GitHub servers (HTTPS/SSH).

**GitHub CI → GitHub Container Registry (GHCR)**

* **Technologies:** GitHub Actions, Docker, GHCR (OCI-compliant registry)
* **How it works:**

1. GitHub runner checks out your code.
2. Runner logs in to GHCR with GHCR\_USERNAME + GHCR\_TOKEN These secrets are created and added in the Github Secrets.
3. docker build creates container images for frontend + backend and any other involved services like database layer in the future.
4. docker push uploads images to GHCR.

* **Communication:** Runner → GHCR via HTTPS REST API.

**GitHub CD → Oracle Kubernetes Engine (OKE)**

* **Technologies:** kubectl, kubeconfig, Kubernetes API server.
* **How it works:**

1. GitHub runner installs **OCI CLI** because your kubeconfig uses exec: oci.

* Command in workflow: Bash  
  curl -L https://raw.githubusercontent.com/oracle/oci-cli/master/scripts/install/install.sh | bash -s -- --accept-all-defaults --install-dir /usr/local/bin

1. Runner sets up kubeconfig from KUBECONFIG\_B64 secret.
2. When you run kubectl apply -f k8s/, kubectl executes oci ce cluster generate-token behind the scenes to get a short-lived auth token.
3. Requests go to OKE’s **API server** over HTTPS.

**Communication:** kubectl (via runner + OCI CLI) → OKE API server → etcd.

**OKE Scheduler → Worker Nodes**

* Technologies: Scheduler, kubelet, containerd.
* How it works:
* Scheduler decides which node runs each pod.
* Kubelet on that node pulls the image and starts the container.
* Communication: API server → kubelet → containerd on OCI Compute nodes.

**Worker Nodes → GHCR**

* Technologies: containerd, Kubernetes imagePullSecrets.
* How it works:  
  + Node tries to pull the image from ghcr.io.
  + If registry is private, Kubernetes injects credentials from ghcr-secret.
* Communication: Worker node → GHCR over HTTPS, authenticated with base64-encoded PAT.

**Frontend Pods → Backend Pods**

* Technologies: ClusterIP Service, CoreDNS, kube-proxy.
* How it works:  
  + Frontend pods call http://backend → CoreDNS resolves name → ClusterIP.
  + kube-proxy routes request to one backend pod.
* Communication: Pod → Service (ClusterIP) → kube-proxy → Pod (TCP/HTTP).

**Users → Application (via OCI Load Balancer)**

* Technologies: Service of type LoadBalancer, OCI Cloud Controller Manager.
* How it works:  
  + A LoadBalancer service in Kubernetes triggers OCI CCM to provision an OCI Load Balancer.
  + OCI LB gets a public IP and forwards traffic to NodePorts on worker nodes → frontend pods.
* Communication: Browser → OCI Load Balancer → NodePort → kube-proxy → Pod.

**OCI CLI in CI/CD Runners**

* *Why it matters:*  
  If your kubeconfig uses exec: oci, kubectl cannot authenticate to OKE without the **OCI CLI** installed and configured.
* *What happens:*
  + When kubectl connects, it calls oci ce cluster generate-token.
  + The CLI uses your API key credentials (tenancy OCID, user OCID, fingerprint, private key) to mint a short-lived access token.
  + This token is presented to the OKE API server for RBAC validation.
* *CD impact:*  
  In GitHub Actions you must:
  + Install OCI CLI (curl -L … install.sh).
  + Configure credentials from secrets (OCI\_TENANCY, OCI\_USER, OCI\_REGION,   
     OCI\_FINGERPRINT, OCI\_PRIVATE\_KEY). This information you can get from your   
     cluster Kubernetes. Click on Access Cluster in the Oracle Console.
  + Export kubeconfig (KUBECONFIG\_B64) that expects exec: oci.
* **Alternative:** Use a **Service Account kubeconfig** with a bearer token → no OCI CLI needed in runners.

**Security & Secrets**

* **ImagePullSecrets:** Required for GHCR private images.
* **App secrets:** Use Kubernetes Secret objects for database passwords, API keys, etc. Mount them as env vars or volumes.
* **RBAC:** Restrict Service Accounts used in CI/CD to only have rights to what they need (avoid cluster-admin for production).

**OCI Networking (OKE Specifics)**

* **VCN Subnets:**
  + Worker nodes typically in **private subnets**.
  + Load Balancers in **public subnets** (for Internet-facing apps).
* **Gateways:**
  + **NAT Gateway** → lets private nodes pull images/packages.
  + **Internet Gateway** → needed for load balancers in public subnets.
* **Why it matters:** If your NAT gateway or routes are missing, pods will fail pulling images from GHCR.

**Observability & Debugging**

* **Important commands:**
  + kubectl get pods -w → watch rollout in real-time.
  + kubectl describe pod <pod> → check events (image pull/auth issues, scheduling errors).
  + kubectl logs <pod> → application logs.
  + kubectl get events --sort-by=.metadata.creationTimestamp → cluster-wide recent issues.
* **Why it matters:** Most CI/CD failures are not in the apply step, but in pod scheduling (ImagePullBackOff, CrashLoopBackOff).