***Beginner Level: Setting Up the Foundation***

1. Kubernetes Basics

   - Objective: Set up a basic Kubernetes cluster on your local machine.

   - Steps:

     - Install Minikube or Kind (Kubernetes in Docker) for local Kubernetes clusters.

     - Get familiar with `kubectl` commands (e.g., `kubectl get`, `kubectl describe`, `kubectl apply`).

     - Create simple YAML files for Pods, Services, and Deployments.

   - Outcome: Successfully deploy a single-container application (e.g., Nginx or a simple Node.js app) on the Kubernetes cluster.

2. Basic Service Deployment

   - Objective: A microservice based application is provided to you.

Project: <https://github.com/UnpredictablePrashant/Slab.ai.git>

   - Steps:

     - Write a simple REST API for listing products (e.g., using Node.js or Python).

     - Create Docker images for the application.

     - Expose the service with a LoadBalancer or NodePort Service.

   - Outcome: Access the Product Catalog service via a public URL or IP.

***Intermediate Level: Scaling and Networking Microservices***

3. Deploy Additional Microservices and Networking

   - Objective: Add more services and enable internal communication.

   - Steps:

     - Deploy additional microservices:

     - Use Kubernetes Services to enable these microservices to communicate.

     - Configure Ingress resources for external access and route traffic based on paths.

   - Outcome: Deploy an Ingress controller (e.g., Nginx Ingress) and configure rules to route traffic to different microservices.

4. ConfigMaps and Secrets

   - Objective: Manage configuration and sensitive data.

   - Steps:

     - Use ConfigMaps to manage environment variables and application configuration.

     - Use Secrets to securely store sensitive data like API keys and passwords.

     - Update your services to consume these ConfigMaps and Secrets.

   - Outcome: Configurations and sensitive data are stored securely, and services use them dynamically.

5. Auto-scaling and Load Management

   - Objective: Implement horizontal pod autoscaling based on resource utilization.

   - Steps:

     - Configure resource limits for each service.

     - Set up Horizontal Pod Autoscalers (HPAs) to scale based on CPU utilization.

   - Outcome: Kubernetes scales services automatically based on load, ensuring efficient resource usage.

***Advanced Level: Production-Ready Enhancements***

6. Stateful Applications and Storage

   - Objective: Deploy a Database (e.g., MongoDB or MySQL) and manage data persistence.

   - Steps:

     - Deploy a database using StatefulSets and PersistentVolumeClaims for data persistence.

     - Configure services to connect to the database.

     - Implement a backup strategy for your database data.

   - Outcome: Your database persists data even when Pods are deleted or restarted.

7. CI/CD Integration with GitOps

   - Objective: Implement a CI/CD pipeline to automate deployments.

   - Steps:

     - Set up a Git repository to store your Kubernetes manifests.

     - Use a tool like ArgoCD for GitOps-based deployment.

     - Configure a CI/CD pipeline in Jenkins, GitHub Actions, or GitLab CI/CD to automate builds and deployments to the Kubernetes cluster.

   - Outcome: Every change to the repository automatically deploys to the Kubernetes cluster.

8. Monitoring and Logging (Optional)

   - Objective: Implement monitoring and logging for application and cluster insights.

   - Steps:

     - Deploy Prometheus and Grafana for monitoring, setting up dashboards for CPU, memory, and request metrics.

     - Integrate a logging solution (e.g., EFK stack - Elasticsearch, Fluentd, and Kibana). [Optional]

     - Configure alerts in Prometheus based on thresholds.

   - Outcome: Gain insights into the performance and health of your services and the Kubernetes cluster.

9. Service Mesh Integration

   - Objective: Add a service mesh like Istio or Linkerd for advanced networking features.

   - Steps:

     - Install a service mesh and configure it to manage traffic between microservices.

     - Enable features like mutual TLS for security, traffic splitting, and circuit-breaking for resilience.

     - Observe telemetry and distributed tracing for inter-service calls.

   - Outcome: Enhanced microservices communication with security and resilience.

Project Wrap-Up: Deploying to a Managed Kubernetes Cluster

10. Deploying to a Cloud Provider

    - Objective: Move your local Kubernetes setup to a managed cloud Kubernetes service.

    - Steps:

      - Choose a cloud provider (AWS EKS, Google GKE, or Azure AKS) and create a Kubernetes cluster.

      - Use kubectl to deploy all your configurations to the cloud cluster.

      - Configure cloud-specific features like managed storage and load balancers.

    - Outcome: Your application is running on a managed Kubernetes cluster, ready for production.