Demo Project: Deploy Microservices application in Kubernetes with Production & Security Best Practices

This guide demonstrates deploying an online shop application with **10 microservices** and **Redis** into a Kubernetes cluster hosted on Linode. It also includes production-grade configurations and security best practices.

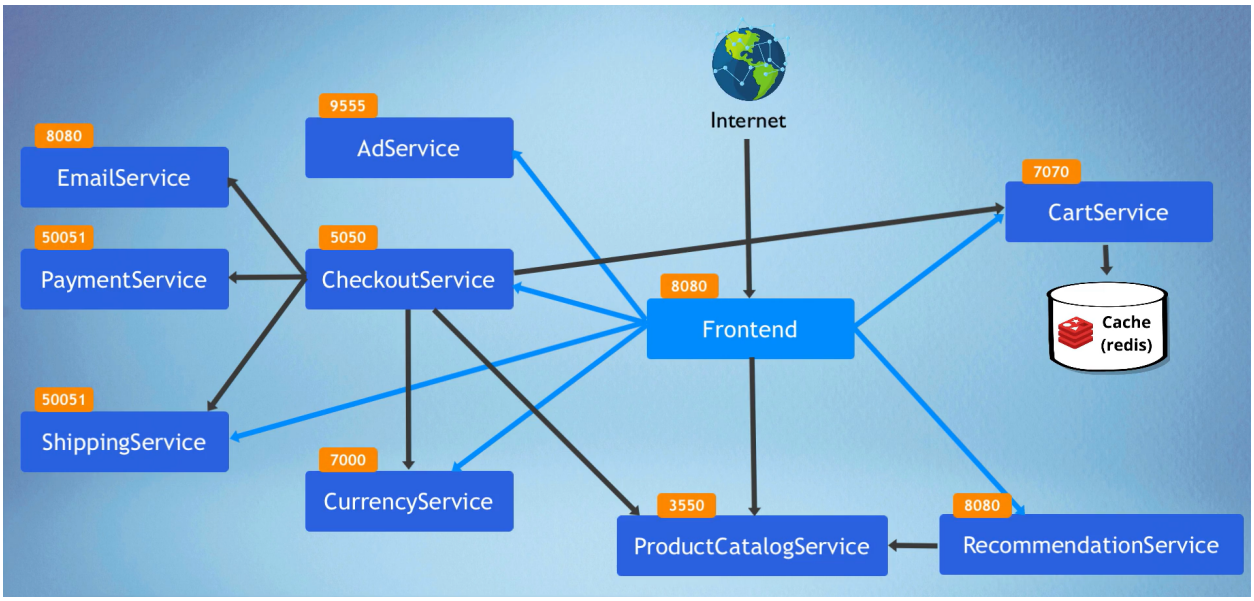
Step 1: Gather Developer-Provided Information

Before starting, ensure you have the following details from the developer:

1. **Microservices Information**:
    - Names of all microservices.
    - Docker image names and versions.
2. **Configuration Details**:
    - Ports exposed by each service.
    - Required environment variables.
3. **Connections**:
    - Dependencies between microservices.
    - External services or databases.
4. **Resources**:
    - CPU and memory requirements.
    - Limits for each container.
5. **Accessibility**:
    - Identify which services will be exposed outside the cluster.

Example Architecture:

The online shop application consists of 10 microservices and Redis for storing shopping cart data. All services are in the same namespace (`microservices`) since one team manages them.



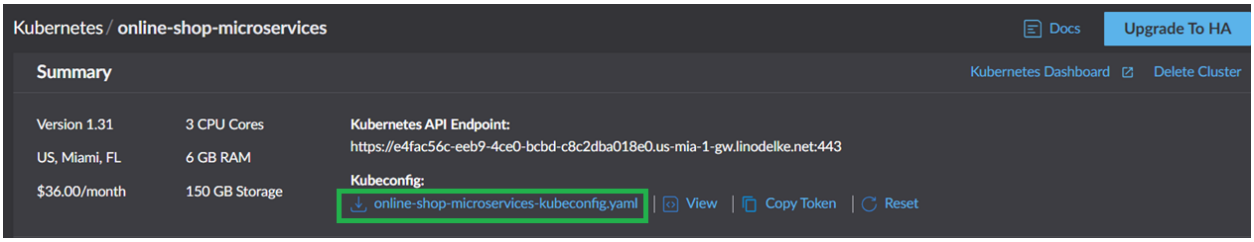
Step 2: Create Deployment & Service Configurations

1. **Prepare the `config.yaml`:**
    - The file should include **Deployment** and **Service** manifests for each microservice.
    - For **each microservice**, define:
        - **Environment Variables**.
        - **Ports** for communication.
        - **Resource Requests and Limits** for containers.
        - **Liveness and Readiness Probes**.
2. **Example Configuration for a Single Microservice:**

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: emailservice
spec:
  selector:
    matchLabels:
      app: emailservice
  template:
    metadata:
      labels:
        app: emailservice
    spec:
      containers:
      - name: service
        image: gcr.io/google-samples/microservices-demo/emailservice:v0.8.0
        ports:
        - containerPort: 8080
        env:
        - name: PORT
          value: "8080"
        livenessProbe:
          grpc:
            port: 8080
          periodSeconds: 5
        readinessProbe:
          grpc:
            port: 8080
          periodSeconds: 5
        resources:
          requests:
            cpu: 100m
            memory: 64Mi
          limits:
            cpu: 200m
            memory: 128Mi
---
apiVersion: v1
kind: Service
metadata:
  name: emailservice
  type: ClusterIP
  selector:
    app: emailservice
  ports:
  - protocol: TCP
    port: 5000
    targetPort: 8080
```

Step 3: Set Up the Kubernetes Cluster

1. **Create a Kubernetes Cluster in Linode:**
   - Navigate to **Create → Kubernetes** in the Linode dashboard.
   - **Cluster Details:**
     - **Cluster Label:** `online-shop-microservices`
     - **Region:** Select the closest region.
     - **Kubernetes Version:** Use the latest version.
   - **HA Control Plane:** No
   - **Node Pools:** Add at least **3 nodes** (2GB Shared CPU each).
   - Click **Create Cluster**.
2. **Download the Kubeconfig File:**
   - Save the `online-shop-microservices-kubeconfig.yaml` file after the cluster is created. This file is required for `kubectl` to interact with the cluster.



Step 4: Configure Access to the Cluster

1. **Secure Kubeconfig File:**
   - Place the `online-shop-microservices-kubeconfig.yaml` in your working directory.
   - Restrict access: `chmod 400 online-shop-microservices-kubeconfig.yaml`
2. **Export the file:**

```
export KUBECONFIG=online-shop-microservices-kubeconfig.yaml
```

3. **Verify the Cluster:**
   - Confirm that the nodes are running: `kubectl get node`

Step 5: Deploy Microservices

1. **Create a Namespace for the Application:** `kubectl create ns microservices`
2. **Deploy the microservices:**
   - Apply the `config.yaml` manifest to the `microservices` namespace:

     ```
     kubectl apply -f config.yaml -n microservices
     ```
3. **Verify the Deployment:**
   - Check the status of pods: `kubectl get pod -n microservices`

     Example:

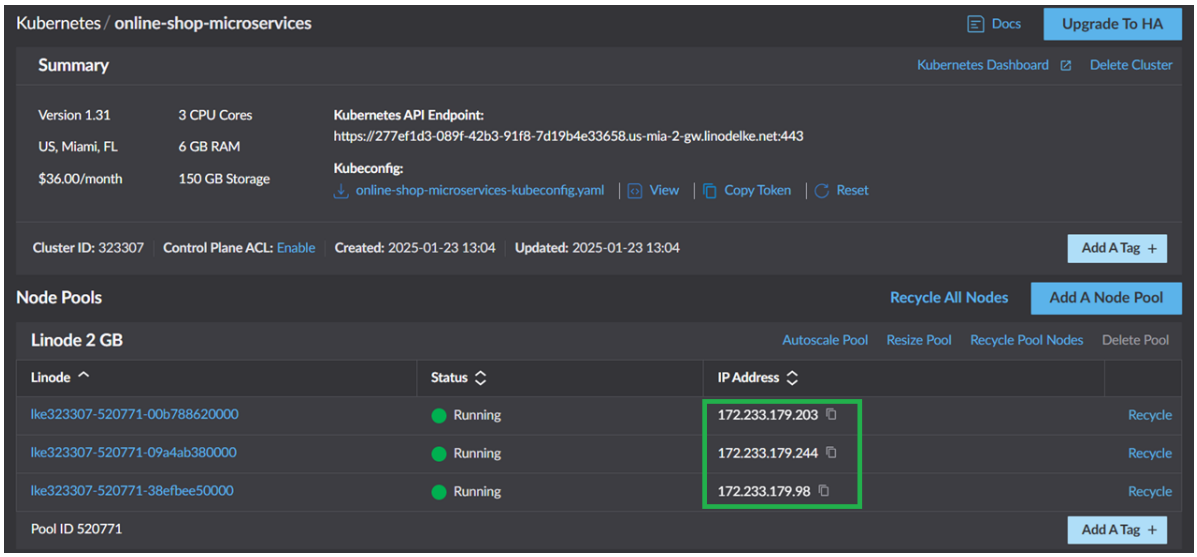
     ```
     kubectl get pod -n microservices
     NAME                                 READY   STATUS    RESTARTS   AGE
     adservice-5c98bd5cb-qkvnn            1/1     Running   0          19h
     cartservice-76c4d44889-s7wf8         1/1     Running   0          19h
     checkoutservice-7cdbddcbb8-t7bvk     1/1     Running   0          19h
     currencyservice-9cc8864fd-qczsg      1/1     Running   0          19h
     emailservice-7c85cfb958-lxbpk        1/1     Running   0          19h
     frontend-6455d5bdc6-72tq5            1/1     Running   0          19h
     frontend-6455d5bdc6-jvfnm            1/1     Running   0          19h
     paymentservice-66f455f78b-tkdf5      1/1     Running   0          19h
     productcatalogservice-6dc5664f4-9kzlp   1/1     Running   0          19h
     recommendationservice-79d9b8d6d4-h5lrh   1/1     Running   0          19h
     redis-cart-7d4fbc49bf-gqqt6          1/1     Running   0          19h
     shippingservice-b468854fb-7r5p5      1/1     Running   0          19h
     ```
   - Check services: `kubectl get svc -n microservices`

     Example:

```
kubectl get svc -n microservices
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
adservice               ClusterIP   10.128.54.59    <none>        9555/TCP         19h
cartservice             ClusterIP   10.128.54.97    <none>        7070/TCP         19h
checkoutservice         ClusterIP   10.128.233.44   <none>        5050/TCP         19h
currencyservice         ClusterIP   10.128.78.119   <none>        7000/TCP         19h
emailservice            ClusterIP   10.128.36.25    <none>        5000/TCP         19h
frontend                NodePort    10.128.241.15   <none>        80:30007/TCP     19h
paymentservice          ClusterIP   10.128.33.250   <none>        50051/TCP        19h
productcatalogservice   ClusterIP   10.128.133.62   <none>        3550/TCP         19h
recommendationservice   ClusterIP   10.128.1.37     <none>        8080/TCP         19h
redis-cart              ClusterIP   10.128.15.109   <none>        6379/TCP         19h
shippingservice         ClusterIP   10.128.122.161  <none>        50051/TCP        19h
```

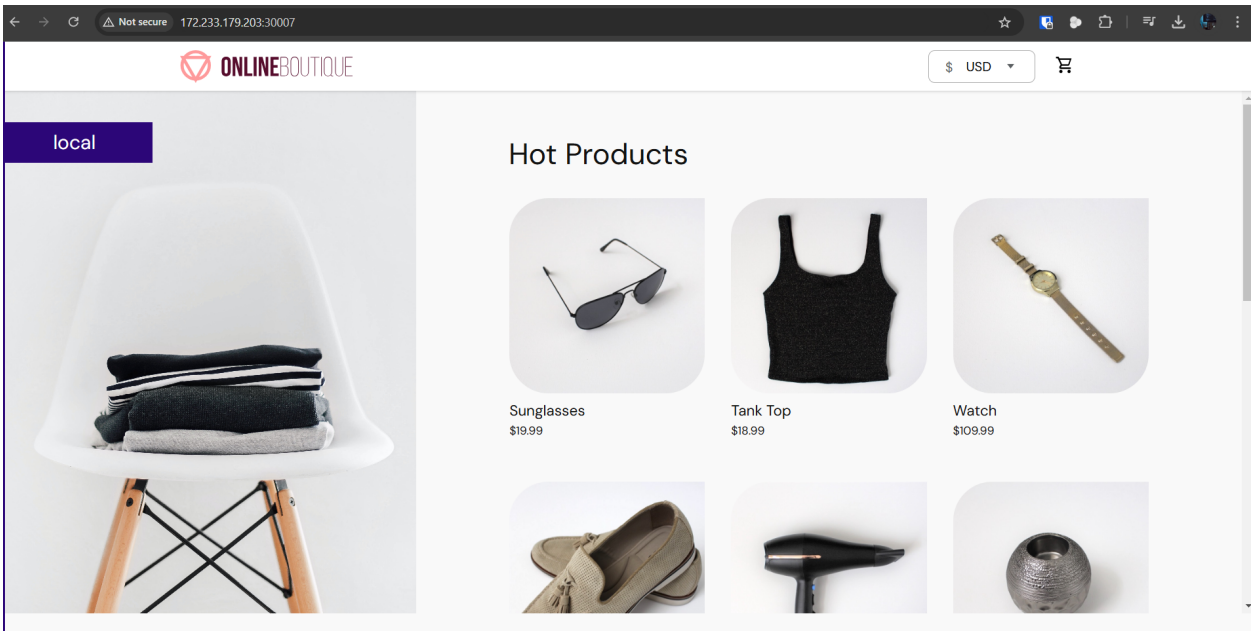
Step 6: Verify Application Access

1. **Obtain an External IP Address:**
   - In the Linode dashboard, navigate to **NodeBalancers** and copy the external IP address of one of the node pools.



2. **Access the Application in a Browser:**
   - Use the IP address and NodePort of the service exposed to the browser (e.g., `30007`):

     `http://<node-pool-ip>:30007`



Step 7: Make Sure to Consider Security Best Practices

1. Container Image Versioning

- **Pin (Tag) Versions** for each container image.
  - This ensures control and visibility of which versions of your applications are running inside the cluster.
  - Avoid using the `latest` tag, as it can lead to unexpected updates and instability.

2. Probes for Application Health

- **Liveness Probe**:
  - Automatically restarts pods when the application crashes or experiences issues.
  - It periodically pings the application endpoint (e.g., every 5 or 10 seconds) to confirm it's responding.

- Configured to run **after** the container has started while the application is running.

- **Readiness Probe**:
  - Verifies that the application has fully started and is ready to receive traffic.
  - Ensures Kubernetes only routes requests to pods that are fully initialized.

---

3. Resource Requests and Limits

- **Why?**
  - Avoid over-allocating or under-allocating cluster resources.
  - Prevent a single container from consuming all available resources.
- **Example Configuration:**

Note: Ensure the requested values fit within the capacity of your cluster nodes. If the requested resources exceed the node size, the pod will not be scheduled.

---

4. High Availability

- **Replicas for Deployments**:
  - Use more than one replica to ensure availability. If one pod crashes, the application remains accessible.
  - Example: `replicas: 2`

- **Multiple Worker Nodes**:
  - Distribute replicas across multiple worker nodes to prevent a **Single Point of Failure**.
  - Reasons for server unavailability:
    - Server crashes.
    - Server reboots (updates or maintenance).
    - Hardware failures.

---

5. Organizational Practices

- **Use Labels for Resources**:
  - Group resources (e.g., pods, services) using labels for better organization and reference in service components.
- **Namespaces**:
  - Isolate resources using namespaces.
  - Simplifies management and enables role-based access control (RBAC) for different teams.

---

6. Security Measures

1. **Vulnerability-Free Images**:
   - Perform vulnerability and security scans on images manually or through your CI/CD pipeline.
2. **No Root Access**:
   - Avoid running containers as root to reduce the risk of resource abuse or damage to the host.
   - Official images usually avoid root access, but always verify third-party images.
3. **Update Kubernetes**:

- Regularly update Kubernetes to the latest version to address security vulnerabilities and bugs.

---

7. Networking Security

- **Avoid NodePorts**:
  - Exposing a NodePort opens a port on all worker nodes, increasing the attack surface.
  - Instead, use **LoadBalancer** type services or an **Ingress Controller** for a secure single entry point.
- **Example Transition**:
  - Change frontend service type from **NodePort** to **LoadBalancer**:

    Before:

    ```
    type: NodePort
    nodePort: 30007
    ```

    After:

    ```
    type: LoadBalancer
    ```

Step 8: Clean-Up

1. **Delete the Application**:
   - Remove all resources for the microservices:
     `kubectl delete -f config.yaml -n microservices`
2. **Testing LoadBalancer Transition**:
   - Apply changes to use **LoadBalancer** instead of **NodePort**, and update the deployment to use `replicas: 2` instead of `replicas: 1` for improved high availability:

- Update the frontend service configuration:

```
type: LoadBalancer
```

- Update the deployment configuration:

```
replicas: 2
```

- Reapply the updated configuration:

`kubectl apply -f config.yaml -n microservices`

3. **Verify the Deployment:**

   Check the status of pods: `kubectl get pod`

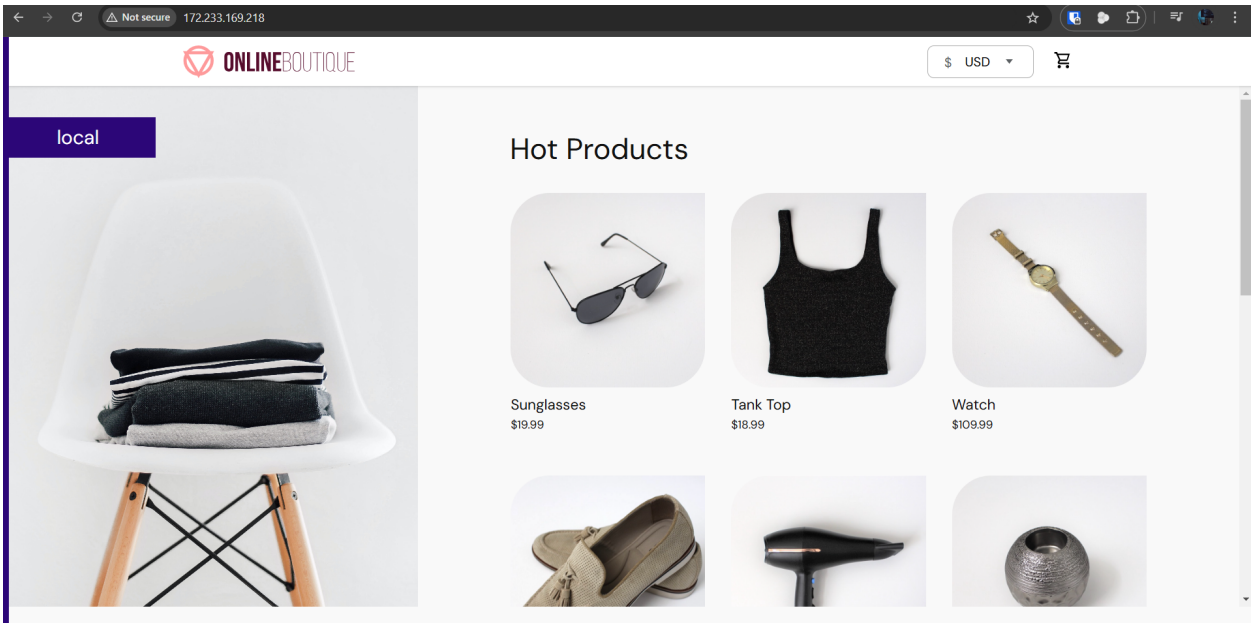
   Example:

```
kubectl get pod
NAME                                 READY   STATUS    RESTARTS   AGE
adservice-54b74b55b7-ft6zg           1/1     Running   0          4m52s
adservice-54b74b55b7-xz8st           1/1     Running   0          4m53s
cartservice-dd4f7764f-wqpd7          1/1     Running   0          4m52s
cartservice-dd4f7764f-zj2l8          1/1     Running   0          4m51s
checkoutservice-767cc65db4-8w8cf     1/1     Running   0          4m54s
checkoutservice-767cc65db4-9bj4k     1/1     Running   0          4m54s
currencyservice-7dcb6cb45f-blmtd     1/1     Running   0          4m51s
currencyservice-7dcb6cb45f-n46qf     1/1     Running   0          4m51s
emailservice-6d488b67f8-5lt7z        1/1     Running   0          4m52s
emailservice-6d488b67f8-rhxrr        1/1     Running   0          4m53s
frontend-696858b68d-47q2w            1/1     Running   0          4m54s
frontend-696858b68d-nftjt            1/1     Running   0          4m54s
paymentservice-795d5d4bd8-7ggbx      1/1     Running   0          4m54s
paymentservice-795d5d4bd8-7m7zf      1/1     Running   0          4m53s
productcatalogservice-7cf54479b6-nlnc4   1/1     Running   0          4m52s
productcatalogservice-7cf54479b6-wxkr4   1/1     Running   0          4m53s
recommendationservice-76887dcc55-j8vvm   1/1     Running   0          4m54s
recommendationservice-76887dcc55-xx7ck   1/1     Running   0          4m54s
redis-cart-6dfd6dfd75-68pk7          1/1     Running   0          4m54s
shippingservice-d84b79bb-5ck2p       1/1     Running   0          4m54s
shippingservice-d84b79bb-kgdsv       1/1     Running   0          4m54s
```

4. **Verify the application:**

   - Get the **LoadBalancer IP** from Linode NodeBalancer.

- Access the application in the browser using: `http://<LoadBalancer-IP>`



6. **Final Clean-Up**:
   - Once testing is complete, delete all resources:
     `kubectl delete -f config.yaml -n microservices`

GitHub Security Tip:

- Exclude the `online-shop-microservices-kubeconfig.yaml` from version control by adding it to `.gitignore`

  Why?

  - This file contains sensitive credentials and certificates that can compromise your cluster if exposed.
