
# Use Case: Deploying Odoo E-Commerce Solution on Scaleway Kubernetes Kapsule

## Objective

Deploy Odoo, an open-source ERP and e-commerce platform, on Scaleway Kubernetes Kapsule to create a scalable, reliable, and secure e-commerce solution. Use `kubectl` and `scw` to manage the deployment, configure a PostgreSQL database, and expose the application via a Scaleway Load Balancer.

## Prerequisites

- Scaleway account with Owner or IAM permissions for KubernetesFullAccess.
- Scaleway CLI (`scw`) installed and configured with API keys (`scw init`).
- `kubectl` installed on your local machine.
- Cert-Manager: Installed in your Kubernetes cluster for automatic TLS certificate management using Let's Encrypt.
- NGINX Ingress Controller: Deployed in your cluster to handle ingress traffic.
- DNS Configuration: A domain (e.g., `odoo-demo.example.com`) pointed to your Scaleway Kubernetes cluster's ingress controller IP.
- Scaleway Managed Database for PostgreSQL: A running PostgreSQL instance with a database created for Odoo.
- Basic understanding of Kubernetes concepts (Pods, Deployments, Services, Ingress, Persistent Volumes).

## Step-by-Step Deployment

1. Create a Kubernetes secret for database credentials

Odoo requires a PostgreSQL database. Store the database credentials in a Kubernetes secret for secure access.

Run the following command, replacing the placeholders with your Scaleway Managed Database details:

```shell
kubectl create secret generic odoo-db-secret \
  --from-literal=host=<your-db-host> \
  --from-literal=port=<your-db-port> \
  --from-literal=user=<your-db-user> \
  --from-literal=password=<your-db-password> \
  --from-literal=dbname=<your-db-name>
```

- `<your-db-host>`: The hostname of your Scaleway Managed PostgreSQL instance (e.g., `postgresql-12345678.database.cloud.scaleway.com`).
- `<your-db-port>`: The port (e.g., `5432`).
- `<your-db-user>`: The database user (e.g., `odoo_user`).
- `<your-db-password>`: The user's password.
- `<your-db-name>`: The database name (e.g., `odoo_db`).

Verify the secret:

```shell
- kubectl get secret odoo-db-secret -o yaml
```

2. Deploy Persistent Volume Claim (PVC)

Odoo requires persistent storage for session data and custom modules. Apply the PVC configuration to create a 5Gi block storage volume on Scaleway.

Save the following as `odoo-pvc.yaml`:

```none
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: odoo-data-pvc
spec:
  accessModes:
    - ReadWriteOnce
```

```
  resources:
    requests:
      storage: 5Gi
  storageClassName: sbs-5k
```

Apply the configuration:

```
None
kubectl apply -f odoo-pvc.yaml
```

Verify the PVC:

```
None
kubectl get pvc odoo-data-pvc
```

3. Configure Let's Encrypt `ClusterIssuer`

To secure Odoo with TLS, configure a Let's Encrypt `ClusterIssuer` for cert-manager.

Save the following as `letsencrypt.yaml`:

```
None
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    email: contact@example.com
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: letsencrypt-prod-key
```

```
    solvers:
    - http01:
        ingress:
          class: nginx
```

- Replace `contact@example.com` with your email address for Let's Encrypt notifications.

Apply the configuration:

```
None
kubectl apply -f letsencrypt.yaml
```

Verify the `ClustterIssuer`:

```
None
kubectl get clusterissuer letsencrypt-prod
```

4. Deploy the Odoo application

Deploy Odoo using a Kubernetes Deployment. The configuration sets correct permissions for the persistent volume and environment variables to connect to the PostgreSQL database.

Save the following as `odoo-deployment.yaml`:

```
None
apiVersion: apps/v1
kind: Deployment
metadata:
  name: odoo
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
    app: odoo
template:
  metadata:
    labels:
      app: odoo
  spec:
    securityContext:
      runAsUser: 1000
      runAsGroup: 1000
      fsGroup: 1000
    containers:
      - name: odoo
        image: odoo:18
        ports:
          - containerPort: 8069
        env:
          - name: HOST
            valueFrom:
              secretKeyRef:
                name: odoo-db-secret
                key: host
          - name: PORT
            valueFrom:
              secretKeyRef:
                name: odoo-db-secret
                key: port
          - name: USER
            valueFrom:
              secretKeyRef:
                name: odoo-db-secret
                key: user
          - name: PASSWORD
            valueFrom:
              secretKeyRef:
                name: odoo-db-secret
                key: password
          - name: DB_NAME
            valueFrom:
              secretKeyRef:
                name: odoo-db-secret
```

```
              key: dbname
        - name: PGHOST
          valueFrom:
            secretKeyRef:
              name: odoo-db-secret
              key: host
        - name: PGPORT
          valueFrom:
            secretKeyRef:
              name: odoo-db-secret
              key: port
        - name: PGUSER
          valueFrom:
            secretKeyRef:
              name: odoo-db-secret
              key: user
        - name: PGPASSWORD
          valueFrom:
            secretKeyRef:
              name: odoo-db-secret
              key: password
      volumeMounts:
        - name: odoo-data
          mountPath: /var/lib/odoo
  volumes:
    - name: odoo-data
      persistentVolumeClaim:
        claimName: odoo-data-pvc
```

Apply the configuration:

```
None
kubectl apply -f odoo-deployment.yaml
```

Verify the deployment:

```
None
kubectl get deployments odoo
kubectl get pods -l app=odoo
```

5. Expose Odoo with a Service

Expose the Odoo deployment using a `LoadBalancer` service to make it accessible externally.

Save the following as `odoo-service.yaml`:

```
None
apiVersion: v1
kind: Service
metadata:
  name: odoo
spec:
  type: LoadBalancer
  selector:
    app: odoo
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8069
```

Apply the configuration:

```
None
kubectl apply -f odoo-service.yaml
```

Check the service status to get its external IP address:

```
None
kubectl get svc odoo
```

6. Configure Ingress for secure access

Set up an Ingress resource to route traffic to Odoo and enable TLS with Let's Encrypt.

Save the following as `odoo-ingress.yaml`:

```
None
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: odoo
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTP"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - odoo-demo.example.com
    secretName: odoo-tls
  rules:
  - host: odoo-demo.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: odoo
            port:
              number: 80
```

- Replace `odoo-demo.example.com` with your domain.

Apply the configuration:

```
None
kubectl apply -f odoo-ingress.yaml
```

Verify the ingress:

```
None
kubectl get ingress odoo
```

Wait for cert-manager to issue the TLS certificate (this may take a few minutes):

```
None
kubectl get certificate odoo-tls
```

7. Access and configure Odoo

- Open a browser and navigate to `https://odoo-demo.example.com` (replace with your domain).
- Follow Odoo's setup wizard to configure the initial database and admin user.
- Log in and set up your e-commerce store using Odoo's web interface.

8. Cleanup

To remove the Odoo deployment run the following commands:

```
Shell
kubectl delete -f odoo-ingress.yaml
kubectl delete -f odoo-service.yaml
kubectl delete -f odoo-deployment.yaml
kubectl delete -f odoo-pvc.yaml
kubectl delete secret odoo-db-secret
```

## Expected Outcomes

- **Persistent storage:** A 5Gi Scaleway block storage volume (`odoo-data-pvc`) ensures data persistence for Odoo's session data and modules across pod restarts.
- **Security**: Using Secrets for credentials and enabling HTTPS via Ingress ensures secure data handling.
- **Cost Efficiency**: Scaleway's pay-for-usage model charges only for running instances and Load Balancers.

## Troubleshooting

- Check Pod logs for potential error messages:

```
Shell
kubectl logs -l app=odoo
```

- Verify database connectivity:
  Ensure the Odoo pod can connect to the PostgreSQL database by checking logs for errors.

- Monitor TLS certificate issuance:

```
None
kubectl describe certificate odoo-tls
```

- Test the Ingress Controller:
  Ensure `https://odoo-demo.example.com` resolves and redirects to HTTPS.