

Container Architecture

[Author Name]

Université de Toulon, LIS UMR CNRS 7020

2026-10-03

Table des matières

1 Purpose of this Document	1
2 Container Strategy Overview	1
3 Container Platform	1
4 Container Architecture	2
5 Container Orchestration	2
6 Container Deployment Patterns	3
7 Example Kubernetes Deployment	3
8 Resource Management	4
9 Security Considerations	4
10 Operational Considerations	5
11 Container Best Practices	5
12 Example Dockerfile	5
13 Template Validation Checklist	6
14 Related Views	6

1 Purpose of this Document

This document defines the container architecture for [System Name], including containerization strategy, orchestration, and operational considerations.

2 Container Strategy Overview

Containerization Goals:

- [Goal 1, e.g., "Consistent deployment across environments"]
- [Goal 2, e.g., "Improved resource utilization"]
- [Goal 3, e.g., "Simplified scaling"]

Key Principles:

- [Principle 1, e.g., "Immutable infrastructure"]
- [Principle 2, e.g., "Single responsibility"]
- [Principle 3, e.g., "Security by default"]

3 Container Platform

Platform:

Orchestration: [Kubernetes/Nomad/ECS/etc.]

Version: [Platform version]

Distribution: [Distribution, e.g., EKS, AKS, Rancher]

Registry:

Provider: [ECR/Docker Hub/GitHub Container Registry/etc.]

Security: [Image scanning approach]

Access Control: [Access management approach]

Base Images:

- Name: [Image name]

Version: [Version policy]

Source: [Origin]

Purpose: [Use cases]

Security: [Scanning/hardening]

```

approach]

- Name: [Additional base image]
  # Similar structure as above

Standards:
  Image Security:
    - Scanning: [Tools and process]
    - CVE Management: [Approach to
vulnerabilities]
    - Signing: [Image signing
approach]
    - Admission Control: [Image
validation process]

Resource Management:
- Requests: [Standard approach]
- Limits: [Standard approach]
- Quotas: [Usage quotas]

Health Checks:
- Liveness: [Requirements]
- Readiness: [Requirements]
- Startup: [Requirements]

```

4 Container Architecture

```

graph TD
    subgraph "Development"
        A[Source Code]
        B[Build Pipeline]
        C[Container Image]
    end

    subgraph "Container Platform"
        D[Container Registry]
        E[Orchestrator]
        F[Service Mesh]
    end

    subgraph "Workloads"
        G[App Containers]
        H[Sidecar Proxies]
        I[Service Containers]
    end

```

```

end
end

A --> B
B --> C
C --> D
D --> E
E --> G
E --> H
E --> I
F --- H

```

5 Container Orchestration

```

@startuml
package "Container Platform" {
    [Container Registry] as CR
    [Orchestrator] as Orch
    [Service Mesh] as SM
    [Ingress Controller] as IC
    [Config Management] as CM
    [Secret Management] as SM2
}

package "Workloads" {
    [App Containers] as App
    [Sidecar Proxies] as SP
    [Service Containers] as Svc
}

[CI/CD] --> CR
CR --> Orch
Orch --> App
Orch --> SP
Orch --> Svc
SM --> SP
IC --> SP
CM --> App
SM2 --> App
@enduml

```

6 Container Deployment Patterns

Deployment Patterns:

- Pattern: Sidecar
 - Description: Helper container extends main container
 - Use Cases: Logging, monitoring, configuration
 - Example: Proxy sidecar for service mesh
- Pattern: Ambassador
 - Description: Proxy for external services
 - Use Cases: API connection management, rate limiting
 - Example: Database connection pooling
- Pattern: Adapter
 - Description: Standardizes output from main container
 - Use Cases: Monitoring, logging normalization
 - Example: Log format standardization
- Pattern: Init Container
 - Description: Runs before main containers
 - Use Cases: Setup, dependency checks, migrations
 - Example: Database schema migration

7 Example Kubernetes Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-app
  labels:
```

```
    app: example-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: example-app
  template:
    metadata:
      labels:
        app: example-app
    spec:
      containers:
        - name: app
          image: example/app:1.0.0
          ports:
            - containerPort: 8080
          resources:
            requests:
              memory: "128Mi"
              cpu: "100m"
            limits:
              memory: "256Mi"
              cpu: "500m"
          livenessProbe:
            httpGet:
              path: /health
              port: 8080
            initialDelaySeconds: 30
            periodSeconds: 10
          readinessProbe:
            httpGet:
              path: /ready
              port: 8080
            initialDelaySeconds: 5
            periodSeconds: 5
          securityContext:
            runAsNonRoot: true
            allowPrivilegeEscalation:
false
              capabilities:
                drop:
                  - ALL
        - name: sidecar
```

```
image: example/sidecar:1.0.0
# Sidecar configuration
```

8 Resource Management

Container Policies:

Resources:

CPU:

Requests: [Standard request values by container type]

Limits: [Standard limit values by container type]

Ratio: [Guideline for limit:request ratio]

Memory:

Requests: [Standard request values by container type]

Limits: [Standard limit values by container type]

Ratio: [Guideline for limit:request ratio]

Storage:

Volumes:

- Type: [Volume type]

Use Case: [When to use]

Sizing: [Guidelines]

- Type: [Additional volume type]
Similar structure as above

Networking:

Policy: [Network policy approach]

CNI: [Container Network Interface plugin]

Security:

- Ingress/Egress rules
- Microsegmentation approach
- Service Mesh integration

Monitoring:

Metrics:

Collector: [Prometheus/Datadog/etc.]

Standard Metrics: [Key metrics collected]

Custom Metrics: [Application-specific metrics]

Logging:

Collector: [Fluentd/Logstash/etc.]

Format: [Standard log format]

Destination: [Where logs are sent]

9 Security Considerations

Container Security:

Build Time:

- Base Image Selection: [Criteria and approach]

- Dependency Scanning: [Tool and process]

- Secret Detection: [Tool and process]

- Image Scanning: [Tool and process]

Deploy Time:

- Image Signatures: [Verification approach]

- Admission Control: [Policy enforcement]

- Secret Management: [Approach to secrets]

- RBAC: [Access control model]

Runtime:

- Security Contexts: [Required settings]

- Pod Security Standards: [Applied standards]

- Runtime Protection: [Runtime security tools]

- Network Policies: [Isolation approach]

10 Operational Considerations

Day 2 Operations:

Scaling:

- Horizontal Pod Autoscaler: [Configuration approach]
- Vertical Pod Autoscaler: [Configuration approach]
- Cluster Autoscaler: [Configuration approach]

Updates:

- Rolling Updates: [Strategy]
- Canary Deployments: [Strategy]
- Blue/Green Deployments: [Strategy]

Reliability:

- Pod Disruption Budgets: [Approach]
- Liveness/Readiness: [Best practices]
- Anti-affinity: [Strategy]
- Topology Spread: [Approach]

Monitoring:

- Container Health: [Key metrics]
- Application Health: [Key metrics]
- Alerting: [Critical conditions]

11 Container Best Practices

1. Image Size

- Use minimal base images (Alpine, Distroless)
- Multi-stage builds to reduce size
- Remove unnecessary tools and packages

2. Configuration

- Externalize configuration
- Use environment variables or config maps
- Avoid hardcoded configs

3. Security

- Run as non-root user
- Use read-only file systems where possible
- Drop unnecessary capabilities
- Scan images for vulnerabilities

4. Resource Management

- Set appropriate resource requests and limits
- Use init containers for setup tasks
- Configure liveness and readiness probes

5. Logging

- Log to stdout/stderr
- Use structured logging (JSON)
- Include correlation IDs for tracing

12 Example Dockerfile

```
# Build stage
FROM node:16-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm ci --production
COPY . .
RUN npm run build

# Production stage
FROM node:16-alpine
WORKDIR /app
COPY --from=build /app/dist ./dist
COPY --from=build /app/node_modules ./node_modules
COPY package.json ./

# Set non-root user
USER node

# Set environment variables
ENV NODE_ENV=production
ENV PORT=8080
```

```
# Expose ports
EXPOSE 8080

# Define health check
HEALTHCHECK    --interval=30s    --
timeout=3s \
    CMD wget -q -O - http://localhost:
8080/health || exit 1

# Define entrypoint
CMD ["node", "dist/main.js"]
```

13 Template Validation Checklist

- ☐ Container strategy defined
- ☐ Platform components identified
- ☐ Base image standards documented
- ☐ Resource management approach specified
- ☐ Security considerations addressed
- ☐ Operational considerations covered
- ☐ Example implementations provided
- ☐ Best practices described
- ☐ Team review completed

14 Related Views

- Infrastructure Architecture - Overall infrastructure
- Component Architecture - Service components
- Deployment Architecture - Deployment approach