Kubernetes Advanced (Day 03)

Helm Charts

CRDs & Operators

StatefulSets & DaemonSets

Karthikeyan Vaiyapuri

Helm Charts and Package Management

Part 01

What is Helm?

01

Package Manager for Kubernetes

02

Template Engine for K8s manifests

03

Release Management tool

04

Known as the "apt/yum for Kubernetes"

Key Helm Concepts

01

Chart Package containing K8s resource definitions

02

Template Parameterized manifest files

03

Values Configuration parameters

04

Release Running instance of a chart

05

Repository Collection of charts

Helm Architecture

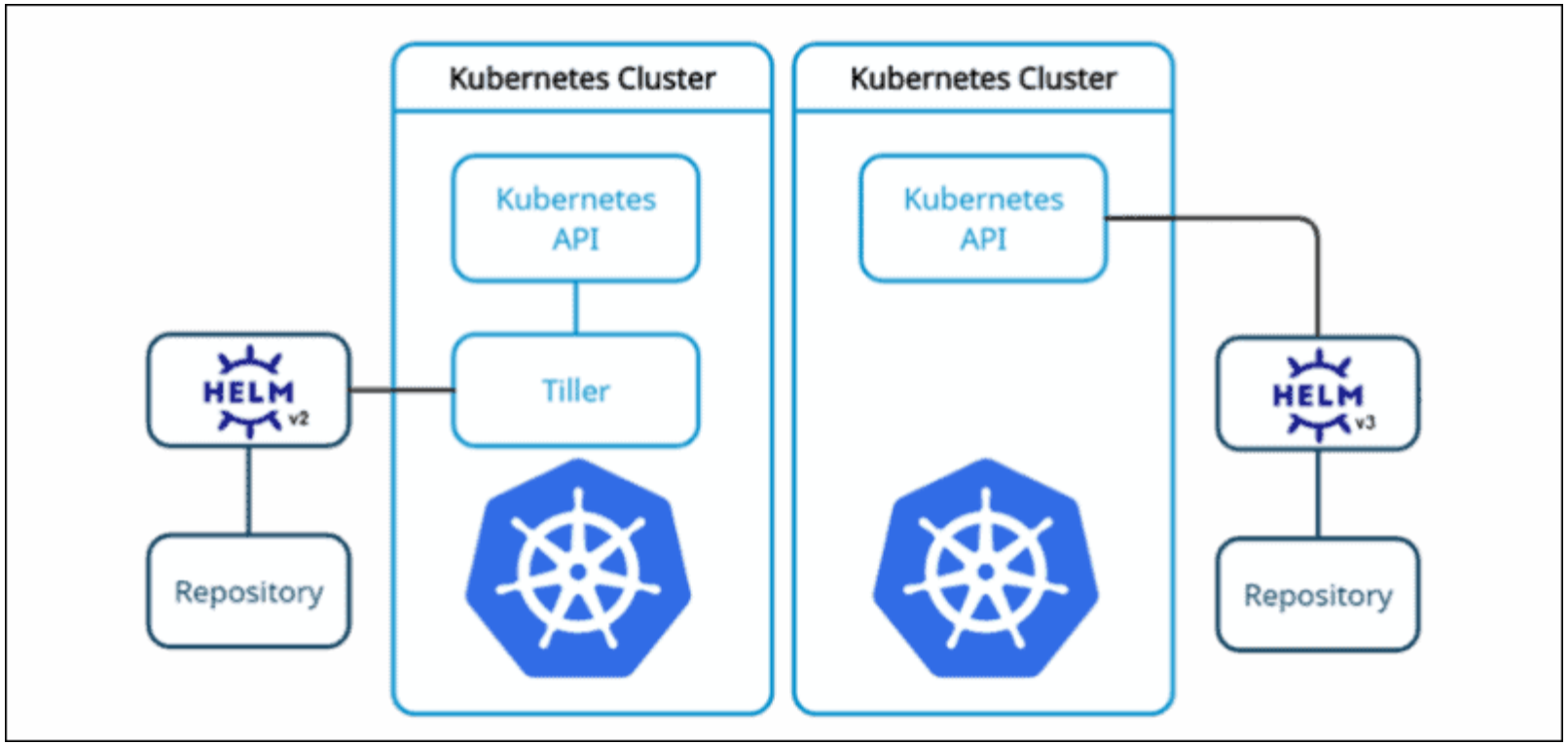


Chart Structure

```
mychart/
├── Chart.yaml          # Chart metadata
├── values.yaml         # Default values
├── templates/          # K8s manifest templates
│   ├── deployment.yaml
│   ├── service.yaml
│   └── _helpers.tpl
└── charts/             # Dependencies
```

Template Functions

Built- in Functions:

{{ .Values.name }}

Sprig Functions:

{{ upper .Values.name }}

Helper Templates:

{{ include "mychart.labels" . }}

Flow Control:

{{ if .Values.enabled }}

Values Hierarchy

Default values.yaml (lowest priority)

Parent chart values

User-supplied values (-f flag)

Command- line parameters (--set flag, highest priority)

Helm Benefits

Reusability:
Template once, deploy many times

01

Versioning:
Track application releases

02

Rollback:
Easy rollback to previous versions

03

Dependency Management:
Handle complex applications

04

Standardization:
Consistent deployment patterns

05

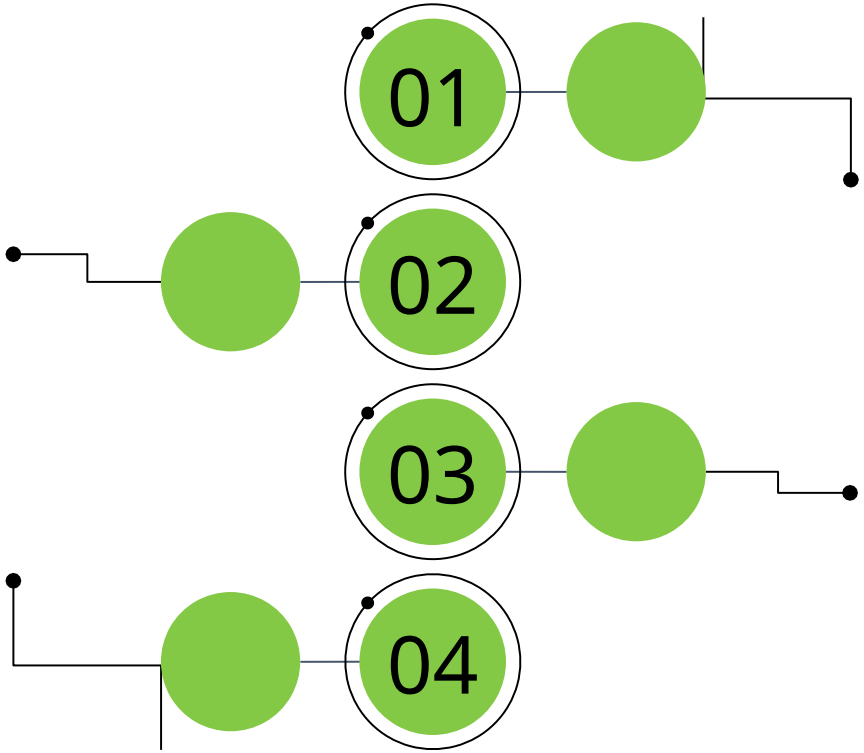
Custom Resource Definitions (CRDs) and Operators

Part 02

Kubernetes Extensibility

Built- in Resources:
Pod, Service, Deployment

Custom Resources:
Extend K8s API



CRDs:
Define new resource types

Controllers:
Watch and act on resources

What are CRDs?

Schema Definition for custom resources

API Extension mechanism

Declarative Specification of custom objects

Cluster-scoped resources

CRD Components

01

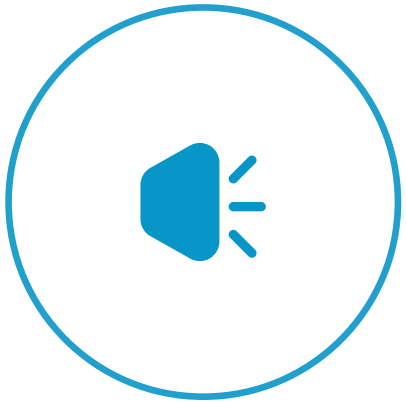
```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: databases.example.com
spec:
  group: example.com
  versions: [...]
  scope: Namespaced
  names:
    plural: databases
    singular: database
    kind: Database
```

Custom Resource Example

01

```yaml
apiVersion: example.com/v1
kind: Database
metadata:
  name: my-postgres
spec:
  engine: postgresql
  version: "13"
  storage: 100Gi
  replicas: 3
```

What are Operators?



01 **Custom Controllers** for CRDs

02 **Domain-specific Knowledge** encoded in software

03 **Automate Operations**
Install, upgrade, backup, scale

04 **Kubernetes-native** applications

Operator Pattern

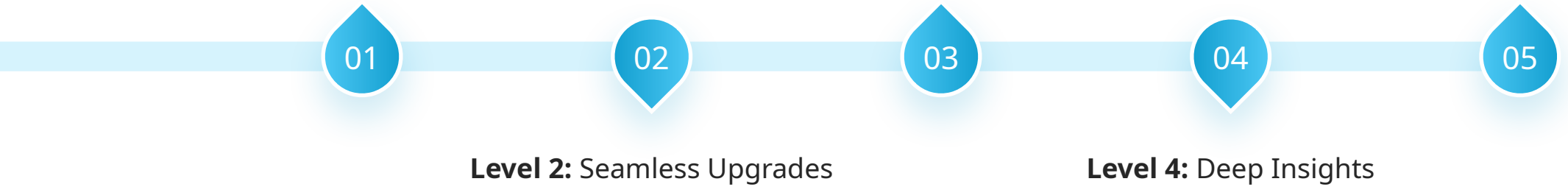
01
Custom Resource (CR)

03
Kubernetes Resources

02
Operator (Controller)

04
Application State

Operator Capabilities Levels



Operator Framework Tools

Operator SDK: Build operators easily

OperatorHub: Discover operators

Operator Lifecycle Manager (OLM): Manage operators

Kubebuilder: Alternative operator framework

CRD vs ConfigMap

CRD	ConfigMap
Structured schema	Unstructured data
API validation	No validation
Custom Controllers	Manual Processing
Versioning support	No versioning
Full K8s integration	Basic storage

StatefulSets and DaemonSets in Detail

Part 03

StatefulSets Overview

Ordered Deployment and scaling

Stable Network Identity

Persistent Storage per instance

Ordered Rolling Updates

Use case: Databases, distributed systems

StatefulSet Characteristics

Stable Pod Names: web-0, web-1, web-2

Persistent Volumes: Survive pod restarts

Ordered Creation: Sequential startup

Headless Service: Direct pod access

Ordered Termination: Reverse order shutdown

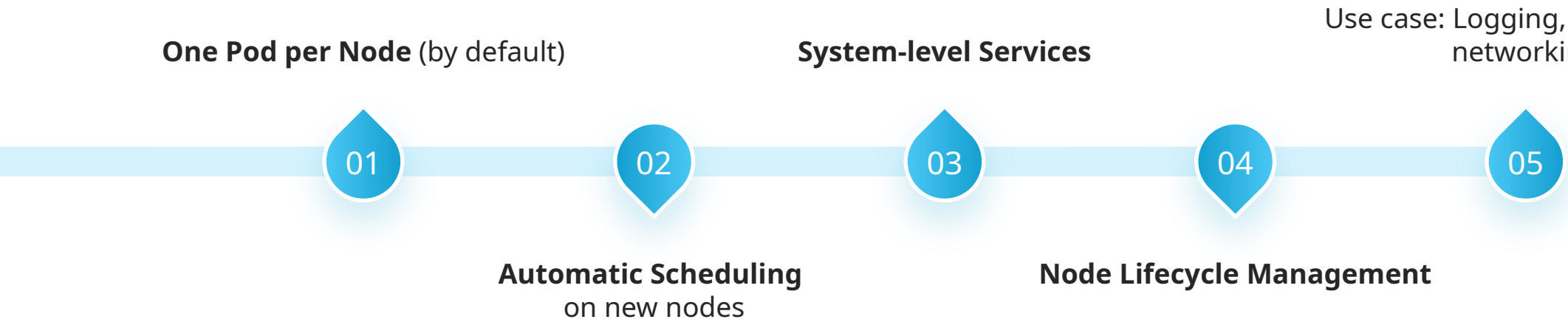
StatefulSet vs Deployment

StatefulSet	Deployement
Ordered pods	Unordered pods
Stable identities	Random names
Persistent storage	Ephemeral storage
Sequential operations	Parallel operations
Stateful apps	Stateless apps

StatefulSet Components

```yaml
apiVersion: apps/v1
kind: StatefulSet
spec:
  serviceName: "nginx-headless"  # Headless service
  replicas: 3
  volumeClaimTemplates:          # PVC template
  - metadata:
      name: data
    spec:
      resources:
        requests:
          storage: 1Gi
```

DaemonSets Overview



DaemonSet Use Cases

01

Log Collection: Fluentd, Logstash

02

Monitoring Agents: Node Exporter, DataDog

03

Networking: Calico, Flannel CNI

04

Storage: GlusterFS, Ceph

05

Security: Falco, Twistlock agents

DaemonSet Features

01 **Node Selector:** Target specific nodes

02 **Tolerations:** Run on tainted nodes

03 **Update Strategy:** Rolling or OnDelete

04 **Resource Limits:** Per-pod constraints

05 **Affinity Rules:** Advanced scheduling

DaemonSet vs Deployment

DaemonSet	Deployement
One pod per node	Multiple replicas
Node-centric	Application-centric
System services	User applications
Node lifecycle tied	Independent lifecycle
No replica count	Configurable replicas

Update Strategies

StatefulSet

RollingUpdate: Sequential updates
OnDelete: Manual pod deletion required

DaemonSet

RollingUpdate: Node-by-node updates
OnDelete: Manual pod deletion required

Best Practices Summary

Part 04

Helm

Use Semantic Versioning

Validate templates before deployment

Keep values.yaml organized

CRDs & Operators

01

Design clear API schemas

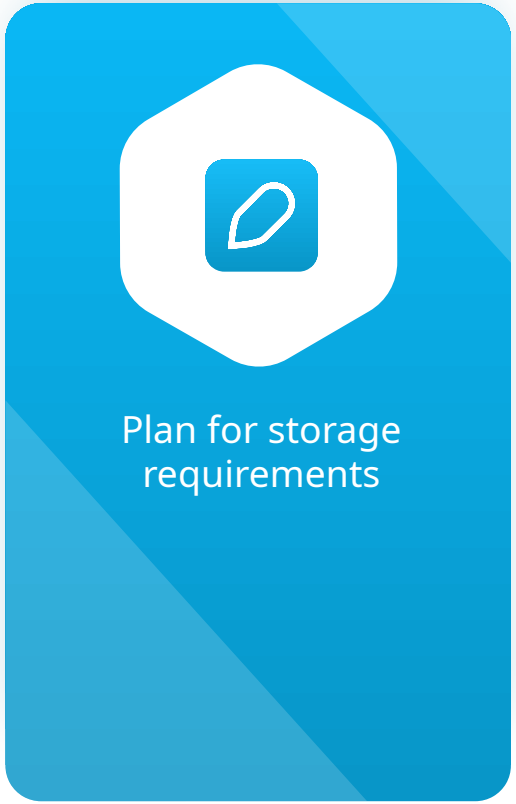
02

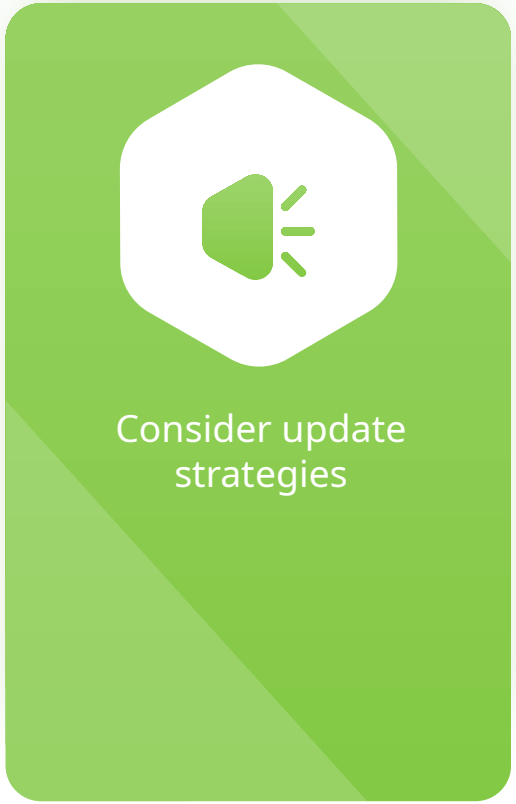
Implement proper validation

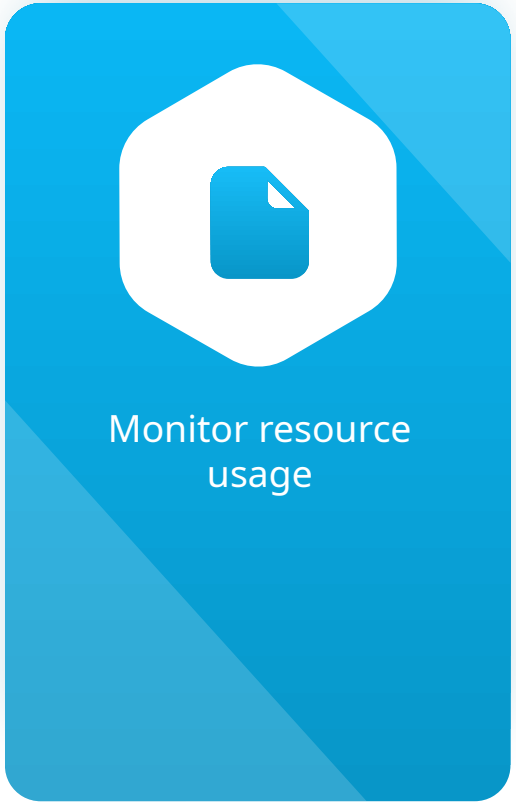
03

Handle edge cases gracefully

StatefulSets & DaemonSets







Thanks

Karthikeyan Vaiyapuri