

The Next Era of Kubernetes Operator Lifecycle Management

Rashmi Gottipati (Red Hat)
Joe Lanford (Red Hat)
Attila Mészáros (Apple)
Jordan Keister (Red Hat)



Presentation team

Rashmi
(rgottipa@redhat.com)



Red Hat

Joe
(jlanford@redhat.com)



Red Hat

Attila



Jordan
(jkeister@redhat.com)



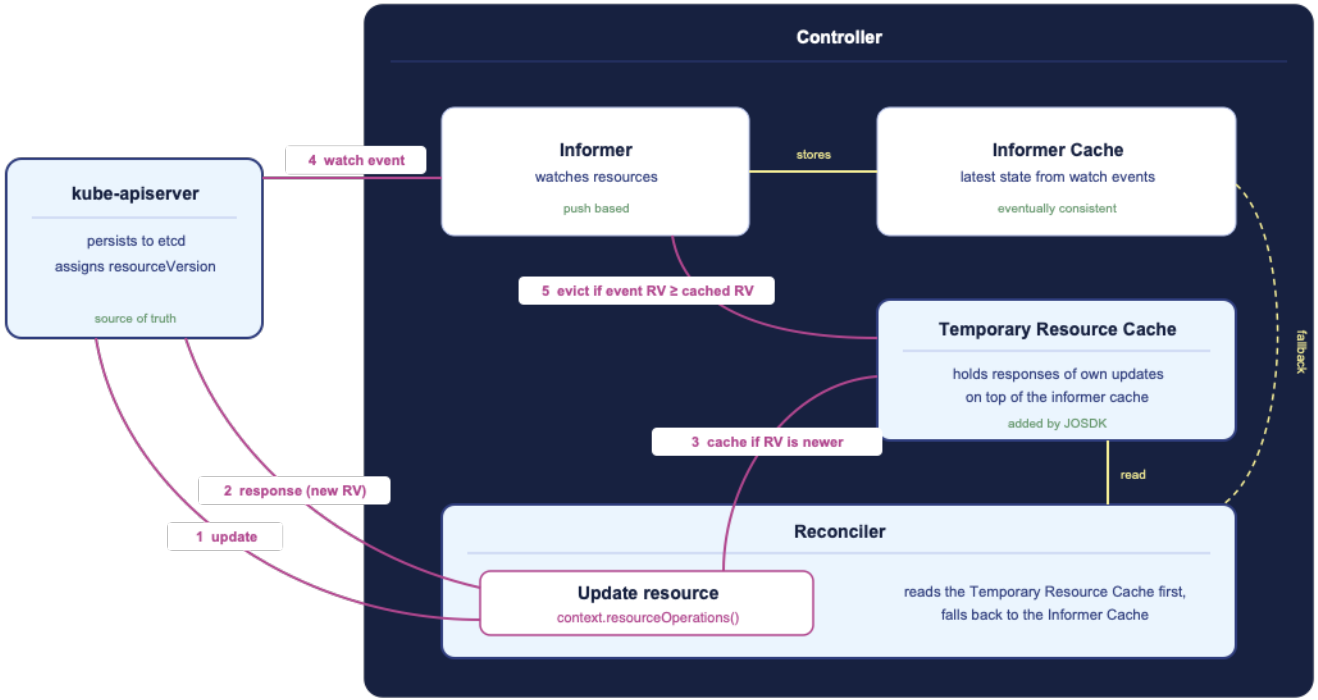
Red Hat

Operator Framework:
Java Operator SDK

Read-Cache-After-Write Consistency

- since v5.3
- Overlay cache vs wait for resource version
- API:
  - UpdateControl.patchStatus(resource)
  - context.**resourceOperations()**.serverSideApply(resource);
- Additional cache brings complexity:
  - Indexers
  - Items store
- Comparable Resource Versions KEP
- Read-cache-after-write consistency blog

Read-Cache-After-Write Consistency

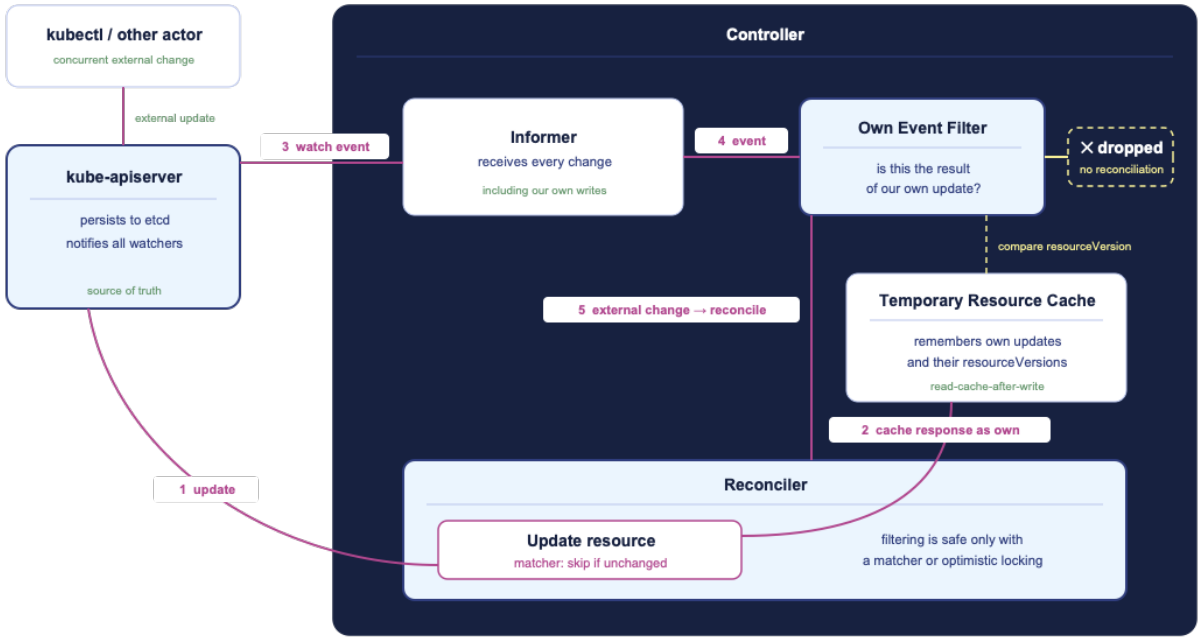


Event Filtering - API & Challenges

- Filter out events for our own updates
- Challenges
  a. Informer lost event problem
  b. Our update did not change the resource but other actor did concurrently
- `context.resourceOperations().serverSideApply(resource);`

Event Filtering

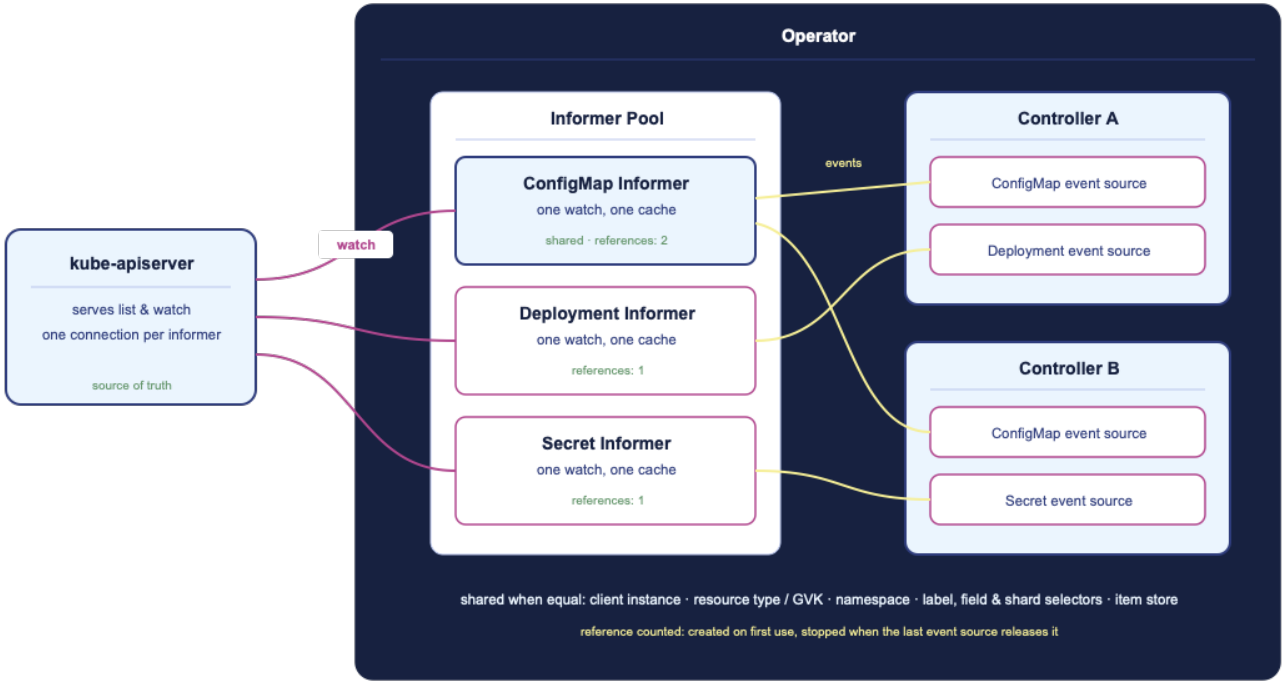
for Read-Cache-After-Write-Consistency



Informer Pools

- Sharing underlying informers between controller
- Matches the informer configuration
  - Except list limit
  - Details in InformerClassifier
- Supports **dynamic** Informer registration and de-registration

Informer Pools



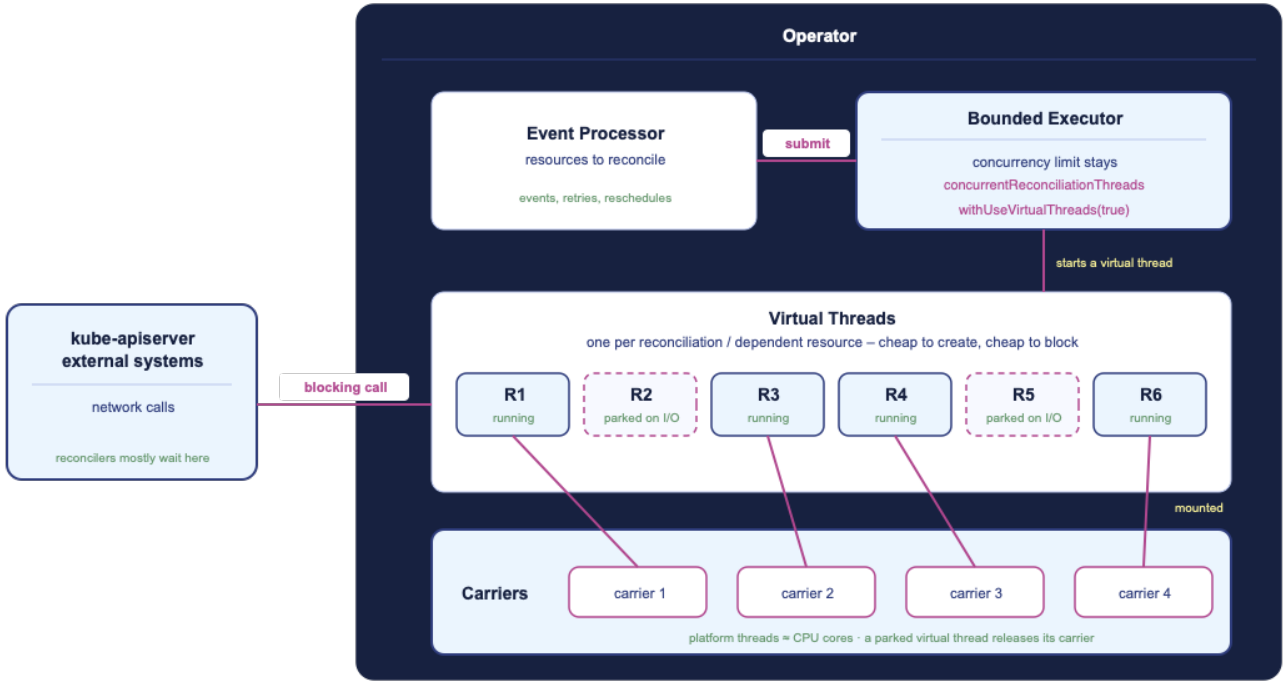
Observability stack enhancements

- Facelift over Observability stack
- Improved logging and Mapped Diagnostic Context (MDC)
  - Putting event and primary resource to MDC if available
- Metrics API refined
  - MicrometerMetricsV2
  - Generic Grafana Dashboard
  - Sample with OTel + Prometheus + Grafana
    - E2E test

Java Virtual Threads (WIP)

- Planned for v5.7.0
- We will officially support this for Java 25+ (but will work with Java 21+)
- Replacing thread pools with virtual thread based one
  - Create a new virtual thread always
  - Support also for polling event sources
- Virtually unlimited number of concurrent reconciliation
  - The explicit limit applies though

Java Virtual Threads (WIP)

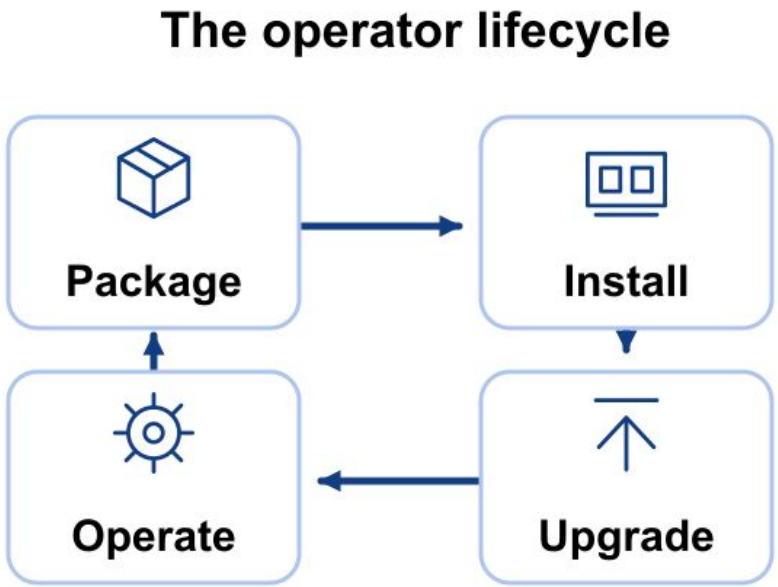


Some honorable mentions…

- EventRecorder API
- ShardSelector support
- ConfigLoader - extensible mechanism to configure your operator with properties files and env variables, etc
- Documentation improvements - operations section

Operator Framework:
~~Operator~~ Lifecycle Manager

Resource?

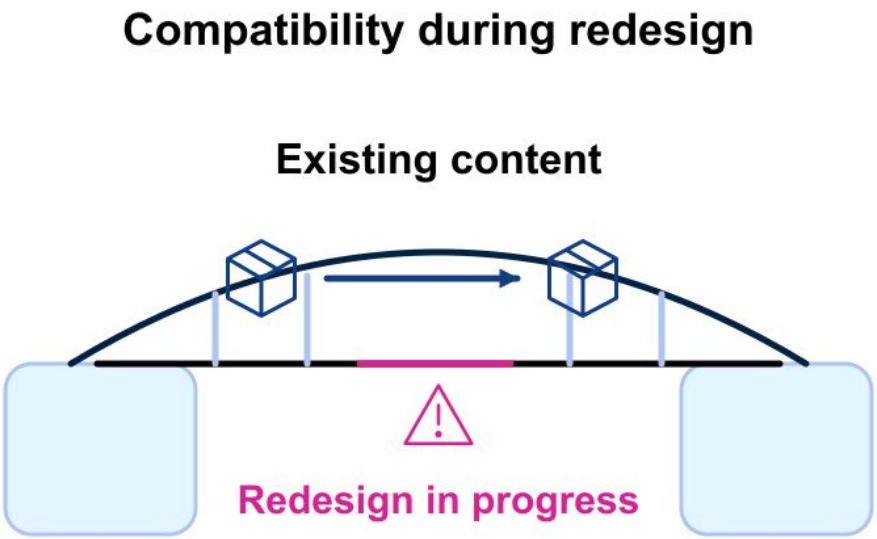


Compatibility and lifecycle foundations

FOUNDATIONAL WORK

OLM evolution builds on existing packages and operational experience.

- OLMv0 established lifecycle management
- Existing registry+v1 content shapes compatibility
- Operational experience informs OLMv1



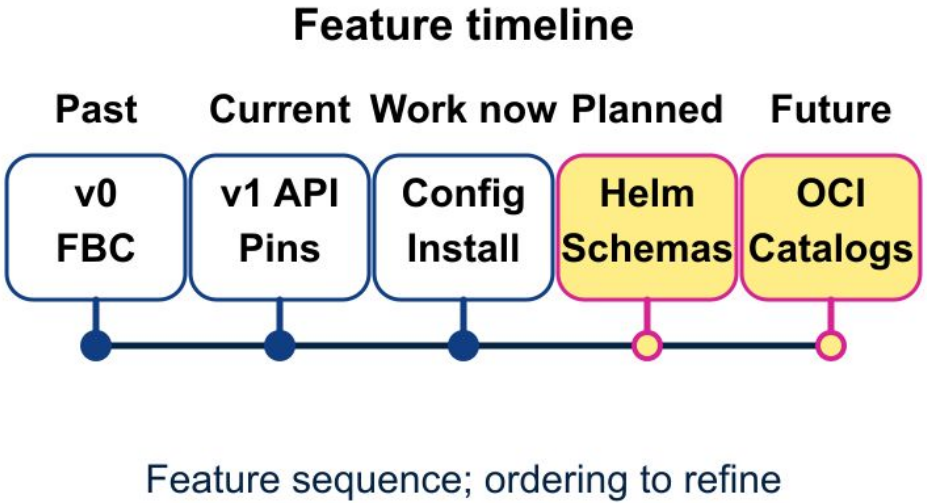
Existing content and users shape lifecycle implementation.

OLM: past, current and future

EVOLUTION / PLANNED / FUTURE WORK

OLM feature evolution

- Past: OLMv0, flawed namespace-local CRD visibility assumptions, SQLite, FBC
- Current: OLMv1, ClusterExtension, pins, webhooks
- Ongoing: rollouts, config, install, migration



Planned: helm+v2 and v2 schemas. Future: OCI catalog storage.

Streamlined Installation

1 Starting Point | 2 Remove Install SA + RBAC | 3 Let OLM Manage Namespace

USER-AUTHORED INPUT

Setup manifests

Namespace

`kind:` Namespace

Install Account

`kind:` ServiceAccount

Permissions

`kind:` RBAC

ClusterExtension

```
kind: ClusterExtension
spec:
  namespace: my-operator
  serviceAccount:
    name: extension-installer
  source: ...
```

→

UNMANAGED RESOURCES

Namespace

`kind:` Namespace

Install Account

`kind:` ServiceAccount

Permissions

`kind:` RBAC

MANAGED RESOURCES

Extension manifests

Workloads, APIs, and other bundle resources

Same output across the first change

Starting Point: the user supplies an install namespace, account, permissions, and a ClusterExtension

Service account change: standard channel since v1.11.0

Managed namespace: experimental channel since v1.12.0

Streamlined Installation

1 Starting Point | **2 Remove Install SA + RBAC** | 3 Let OLM Manage Namespace

USER-AUTHORED INPUT

Setup manifests

Namespace

`kind:`
Namespace

Install Account

`kind:`
ServiceAccount

Permissions

`kind:`
RBAC

ClusterExtension

```
kind: ClusterExtension
spec:
  namespace: my-operator
  serviceAccount:
    name: extension-installer
  source: ...
```

→

UNMANAGED RESOURCES

Namespace

`kind:`
Namespace

Install Account

`kind:`
ServiceAccount

Permissions

`kind:`
RBAC

MANAGED RESOURCES

Extension manifests

Workloads, APIs, and other bundle resources

Same output across the first change

The controller uses its own service account. Unmanaged service account and RBAC for extension installation are no more!

Service account change: standard channel since v1.11.0

Managed namespace: experimental channel since v1.12.0

Streamlined Installation

1 Starting Point | 2 Remove Install SA + RBAC | **3 Let OLM Manage Namespace**

USER-AUTHORED INPUT

Setup manifests

Namespace

```
kind:
Namespace
```

Install Account

```
kind:
ServiceAccount
```

Permissions

```
kind:
RBAC
```

ClusterExtension

```
kind: ClusterExtension
spec:
  namespace: my-operator
  serviceAccount:
    name: extension-installer
  source: ...
```

UNMANAGED RESOURCES

Namespace

```
kind:
Namespace
```

MANAGED RESOURCES

Extension manifests

Workloads, APIs, and other bundle resources

Same output across the first change

OLM derives and manages the namespace by default. ClusterExtension deletion cascades to managed namespace deletion

Service account change: standard channel since v1.11.0

Managed namespace: experimental channel since v1.12.0

Streamlined Installation

1 Starting Point | 2 Remove Install SA + RBAC | **3 Let OLM Manage Namespace**

USER-AUTHORED INPUT

Setup manifests

Namespace	Install Account	Permissions
kind: Namespace	kind: ServiceAccount	kind: RBAC

ClusterExtension

```
kind: ClusterExtension
spec:
  namespace: my-operator
  serviceAccount:
    name: extension-installer
  source: ...
```

→

MANAGED RESOURCES

Namespace

kind: **Namespace**

Extension manifests

Workloads, APIs, and other bundle resources

Same output across the first change

OLM derives and manages the namespace by default. ClusterExtension deletion cascades to managed namespace deletion

Service account change: standard channel since v1.11.0

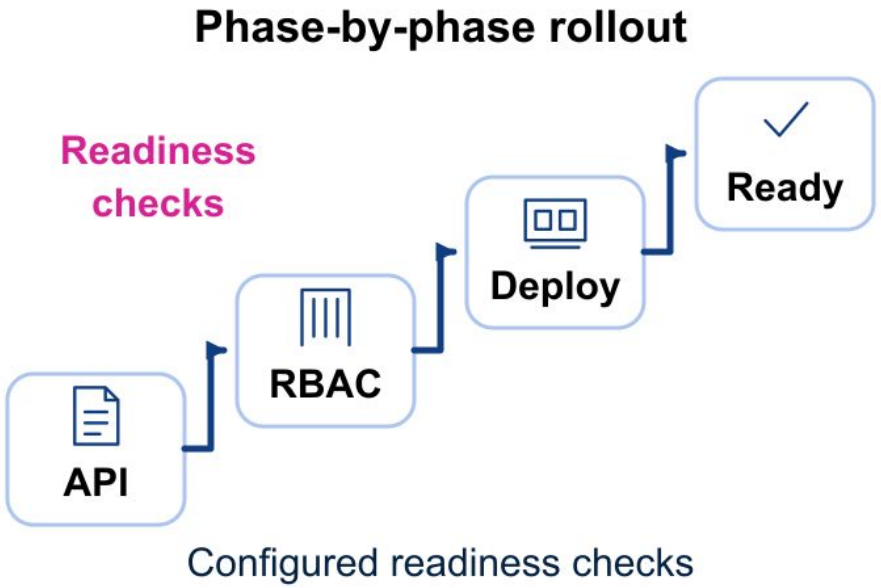
Managed namespace: experimental channel since v1.12.0

Inspect phased resource rollouts

IN ACTIVE DEVELOPMENT / CONTROLLED BY FEATURE-GATE

Ordered phases expose resource application and readiness progress.

- Apply resource groups in phase order
- Wait for applicable readiness checks
- Inspect ClusterObjectDeployment and revision status



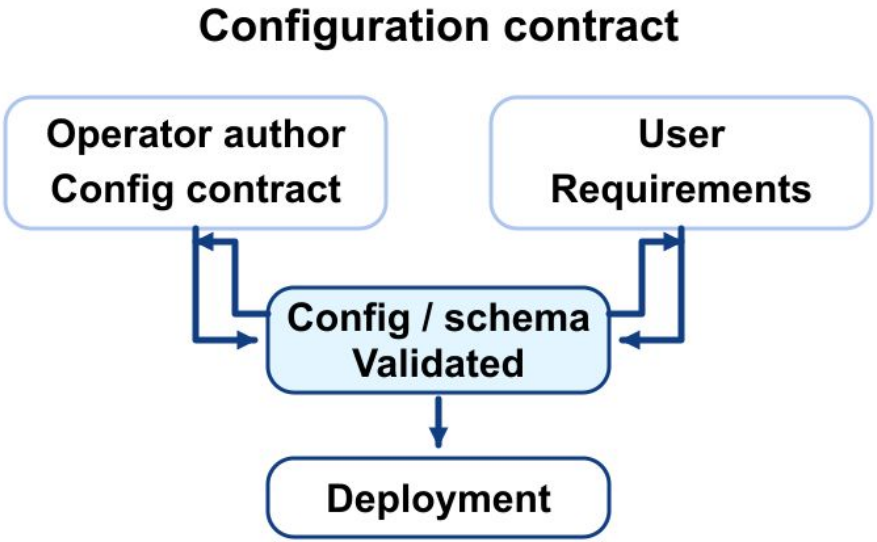
Readiness checks show which phase is waiting.

User control of extension deployments

IN ACTIVE DEVELOPMENT / CONTROLLED BY FEATURE-GATE

Flexible user-defined deployment configuration.

- Environment variables and resource settings
- Placement, volumes and deployment metadata
- Operator-provided configuration and schemas



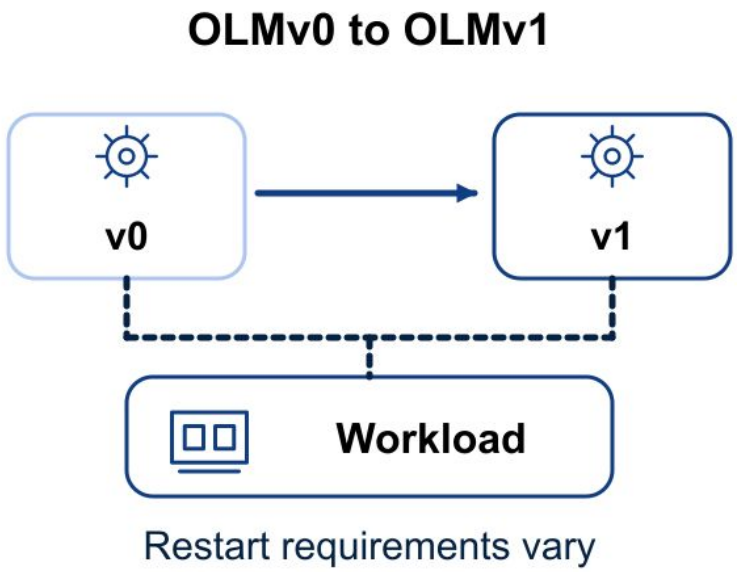
OLM applies user choices through operator configuration.

OLMv0 → OLMv1 migration

IN ACTIVE DEVELOPMENT

Move operator management from OLMv0 to OLMv1.

- Prepare v1 catalogs from v0 CatalogSources
- Check and preview operator migrations
- Migration library with example binaries



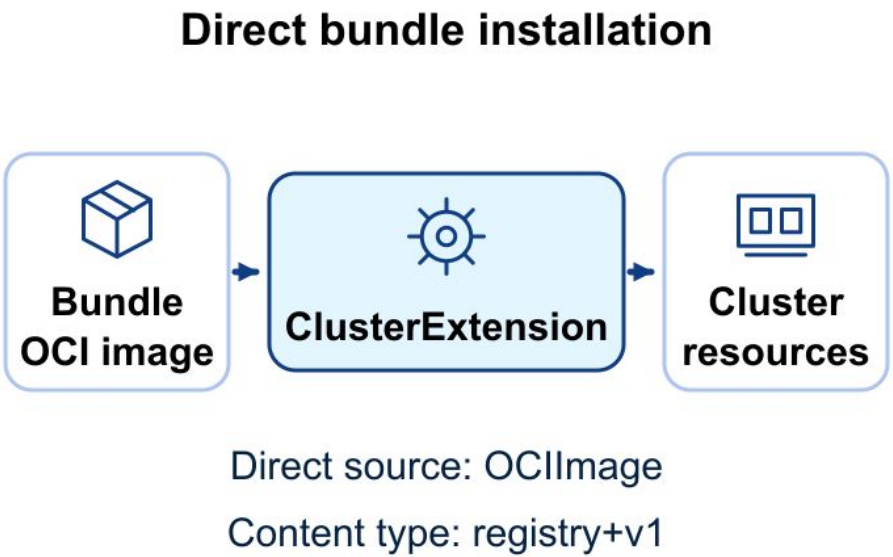
Structured output supports automation and models.

Install a bundle directly

IN ACTIVE DEVELOPMENT / CONTROLLED BY FEATURE-GATE

Select a registry+v1 bundle by its OCI image reference.

- Use sourceType: OCIImage
- Set source.ociImage.ref
- Bypass catalog resolution

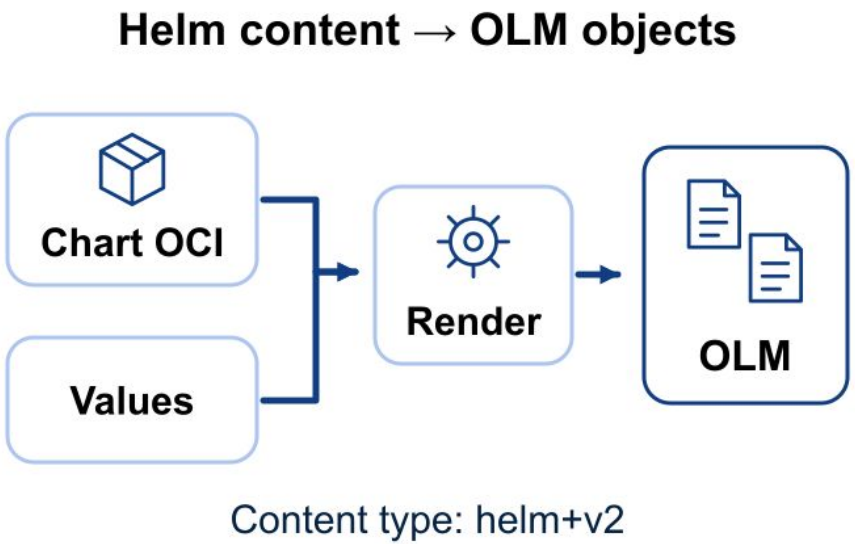


Planned Helm content support

PLANNED / helm+v2

The rendering contract targets deterministic chart behavior.

- Chart and values produce managed manifests
- Allow-listed features support reproducible rendering
- helm+v2 identifies the content representation



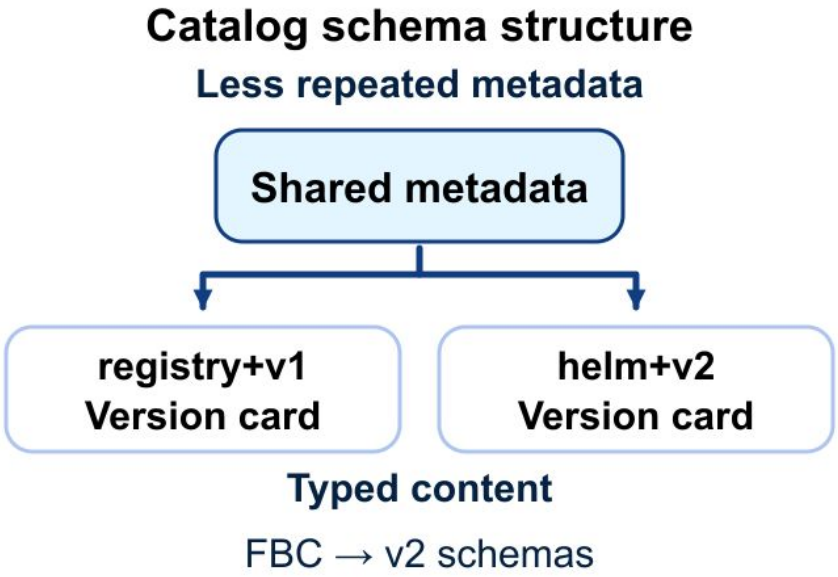
Standard chart artifacts use a defined rendering contract.

Catalog performance and content types

PLANNED / FUTURE WORK

Design for lower catalog-processing cost and additional content types.

- Lower duplicated CSV metadata
- Direct version fields reduce property traversal
- Typed registry+v1 and helm+v2 content



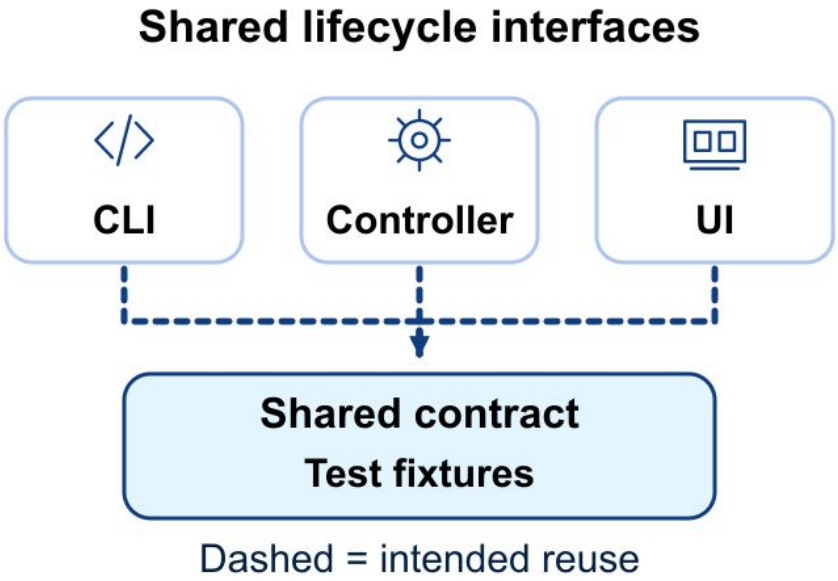
File-based catalogs remain the established foundation.

Shared lifecycle logic in library-olm

IN ACTIVE DEVELOPMENT

Reuse lifecycle logic through a library with explicit contracts.

- Share migration checks and workflow logic
- Exercise current development examples
- Record expected results in reusable fixtures



Examples and fixtures make supported behavior concrete.

Demos

Runnable companions for the topics in this talk:

Direct bundle install and managed namespaces	demos/01-direct-bundle-install
Phased resource rollouts and revision inspection	demos/02-phased-rollout
Deployment configuration	demos/03-deployment-config
OLMv0 → OLMv1 migration with structured output	demos/04-migration
Helm chart direct install	demos/05-helm-direct-install

github.com/grokspawn/kubecon-2026-demos

Contribute to Operator Framework

PROJECT DEVELOPMENT

Contribute to Operator Framework development.

- Read the repository contribution guide
- Discuss an issue and propose a change
- Submit code, documentation or test fixtures

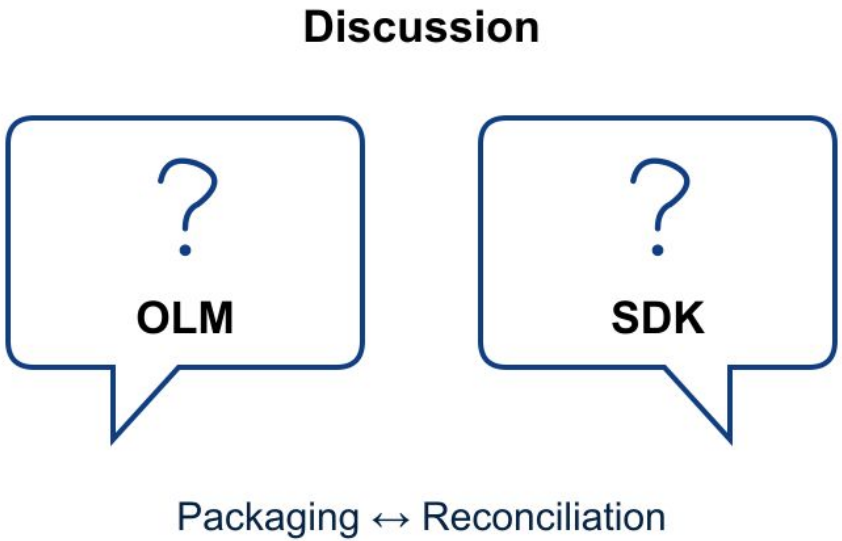


operator-controller: CONTRIBUTING.md

Questions?

Questions about lifecycle APIs, packaging or controller behavior?

- OLM configuration and migration
- Helm content and catalog schemas
- Java Operator SDK reconciliation



Discussion of capabilities and proposed work.