Operators Monitoring and Observability Best Practices in Operator SDK

Shirly Radco
Principal Software Engineer
Red Hat

João Vilaça
Associate Software Engineer
Red Hat

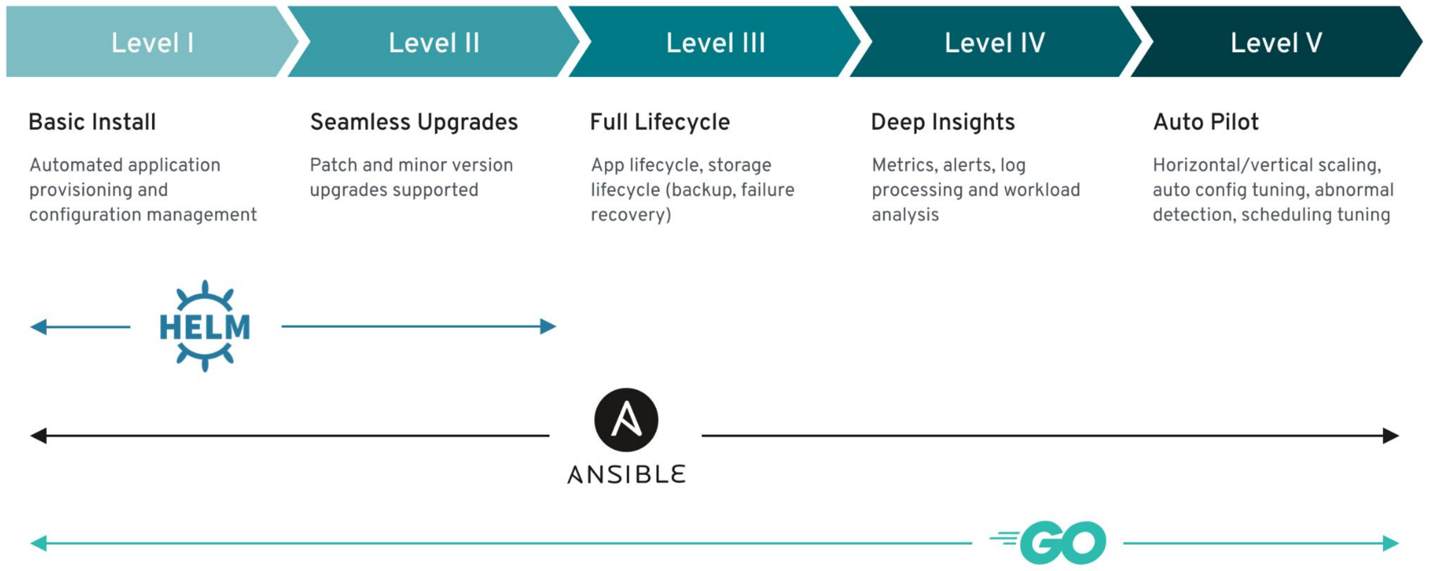
February 2023

What Will We Talk About Today

Operators Observability

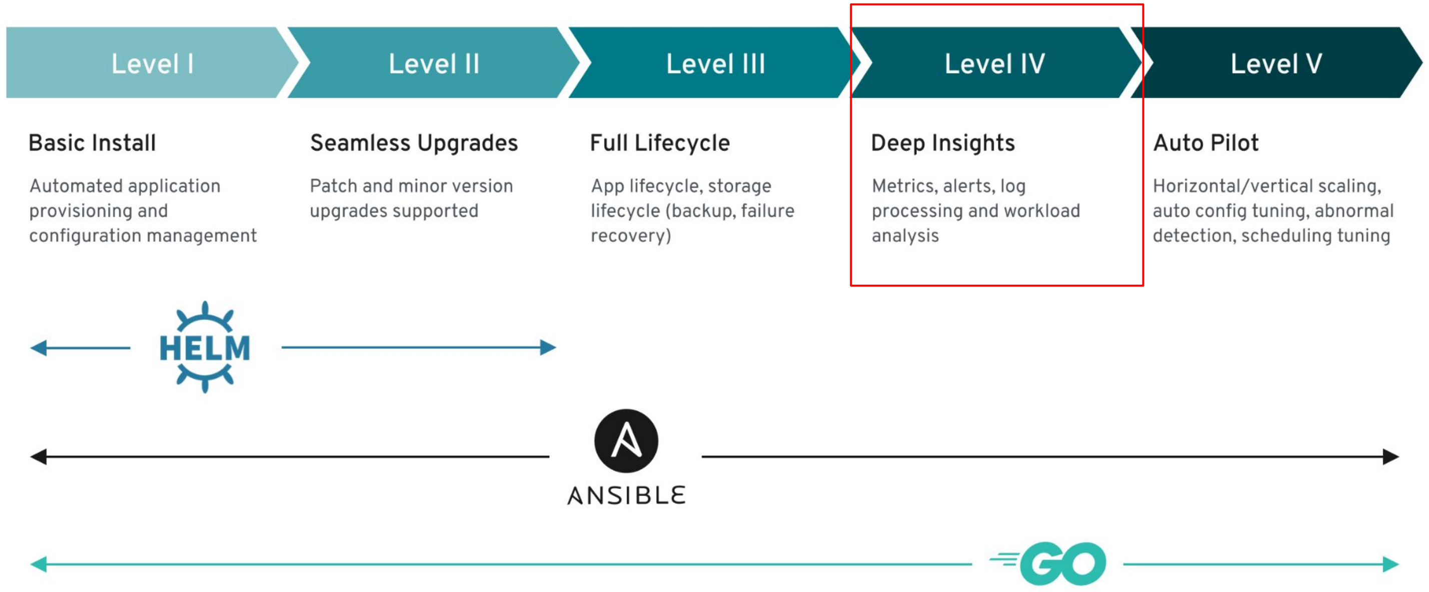
- When to start
- Metrics - Levels of Maturity
- Why monitor
- What to Monitor
- Best Practices in Operator SDK
- Code Examples in Memcached Operator

When should I start learning about monitoring


Level I
Level II
Level III
Level IV
Level V
Basic Install
Automated application provisioning and configuration management
Seamless Upgrades
Patch and minor version upgrades supported
Full Lifecycle
App lifecycle, storage lifecycle (backup, failure recovery)
Deep Insights
Metrics, alerts, log processing and workload analysis
Auto Pilot
Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning
HELM
ANSIBLE
GO

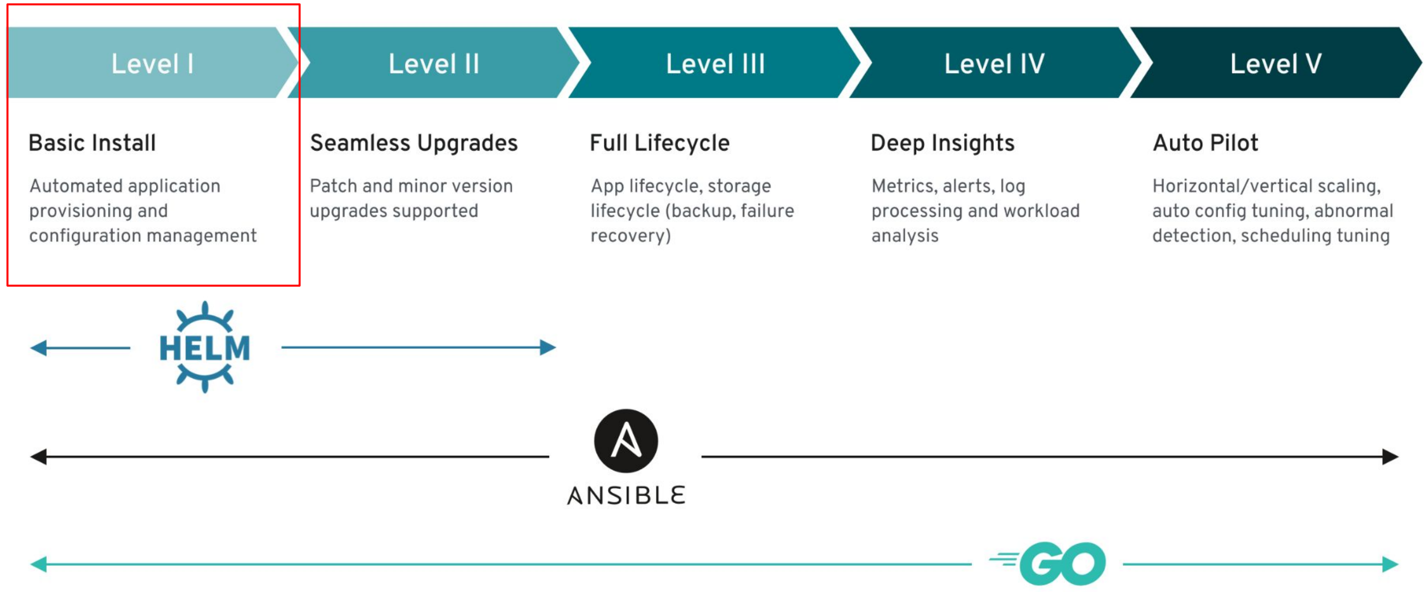
Source: https://sdk.operatorframework.io/docs/overview/operator-capabilities/

When should I start learning about monitoring


Level I
Level II
Level III
Level IV
Level V
Basic Install
Automated application provisioning and configuration management
Seamless Upgrades
Patch and minor version upgrades supported
Full Lifecycle
App lifecycle, storage lifecycle (backup, failure recovery)
Deep Insights
Metrics, alerts, log processing and workload analysis
Auto Pilot
Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning
HELM
ANSIBLE
GO

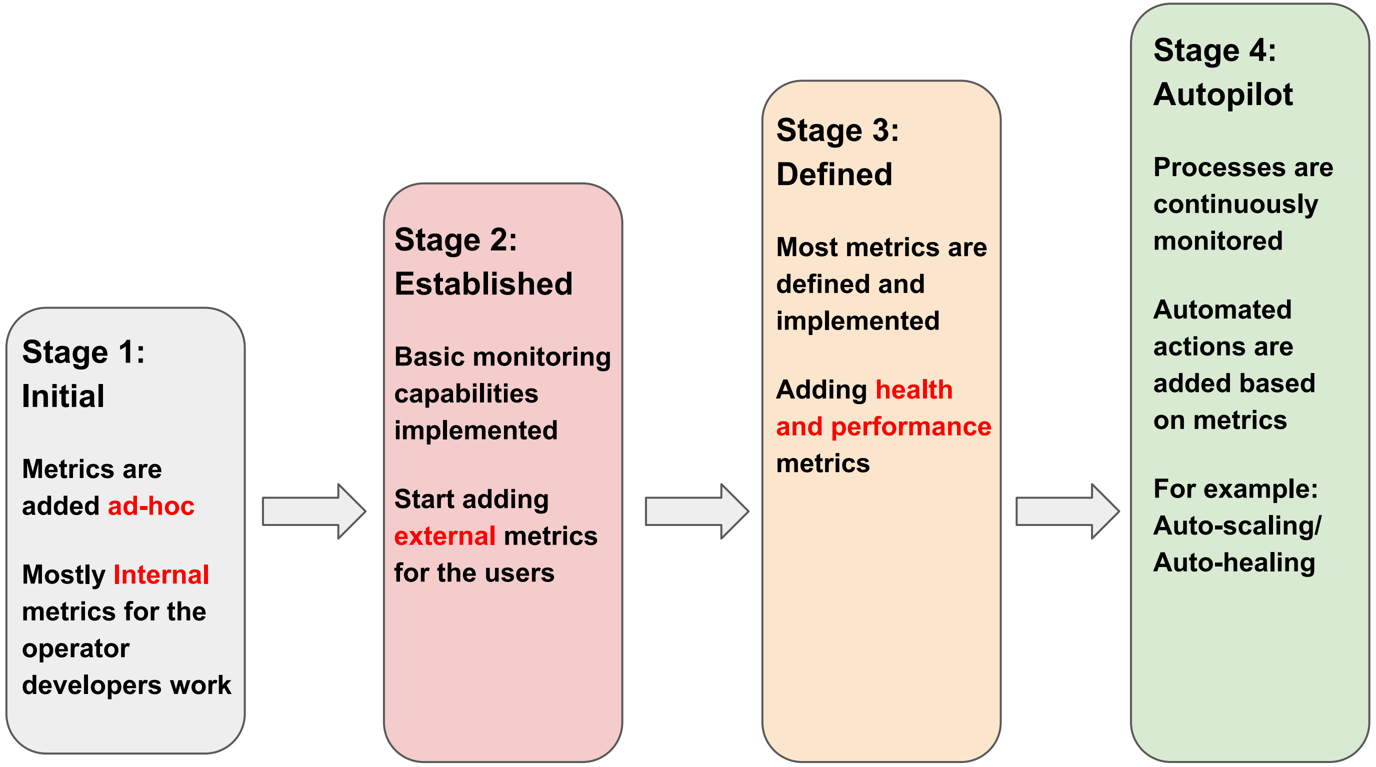
Source: https://sdk.operatorframework.io/docs/overview/operator-capabilities/

When should I start learning about monitoring


Level I
Basic Install
Automated application provisioning and configuration management
Level II
Seamless Upgrades
Patch and minor version upgrades supported
Level III
Full Lifecycle
App lifecycle, storage lifecycle (backup, failure recovery)
Level IV
Deep Insights
Metrics, alerts, log processing and workload analysis
Level V
Auto Pilot
Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning
HELM
ANSIBLE
GO

Source: https://sdk.operatorframework.io/docs/overview/operator-capabilities/

Operator Metrics - Levels of Maturity



Stage 1: Initial

Metrics are added ad-hoc

Mostly Internal metrics for the operator developers work

Stage 2: Established

Basic monitoring capabilities implemented

Start adding external metrics for the users

Stage 3: Defined

Most metrics are defined and implemented

Adding health and performance metrics

Stage 4: Autopilot

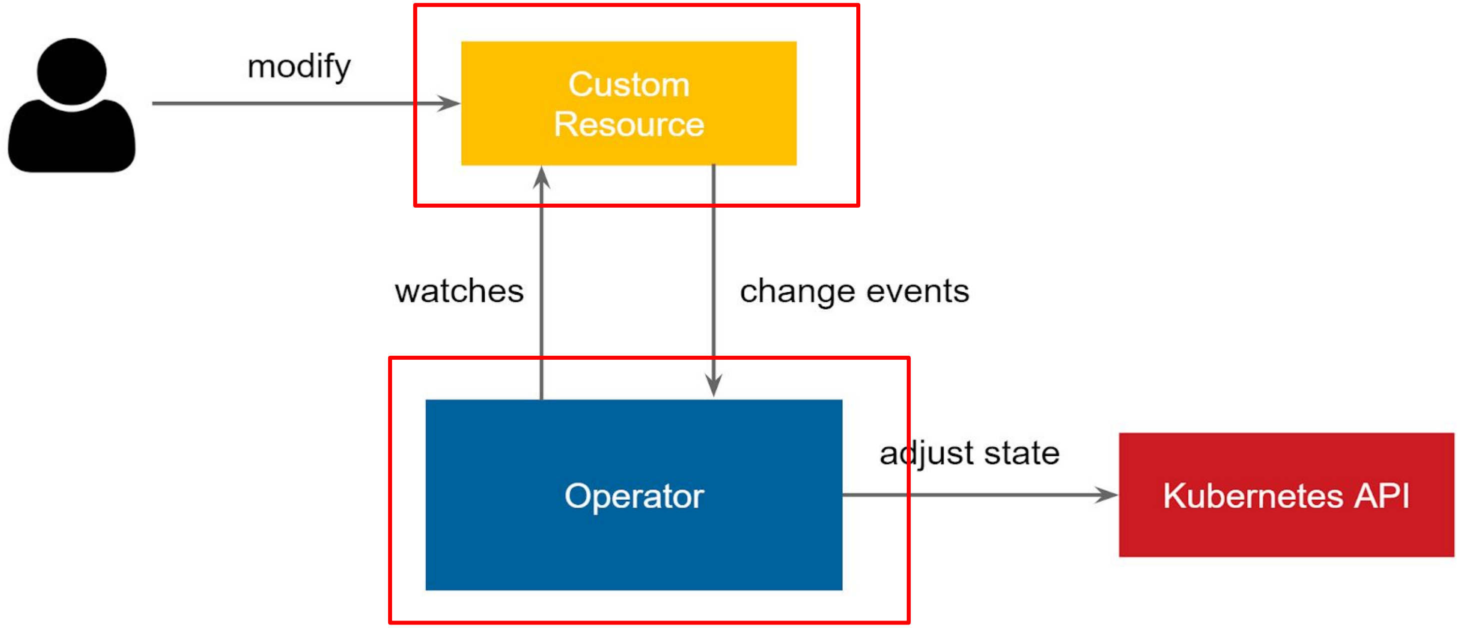
Processes are continuously monitored

Automated actions are added based on metrics

For example: Auto-scaling/ Auto-healing

What Should Operator Developers Monitor?

The Operator and Custom Resources **Performance and Health**



Source:https://cloud.redhat.com/blog/build-your-kubernetes-operator-with-the-right-tool

Why Add Operator Observability?

Performance and Health

Detect issues early, reduce downtime, detect regressions

Planning and Billing

Improve resources planning and Profitability

Alerts

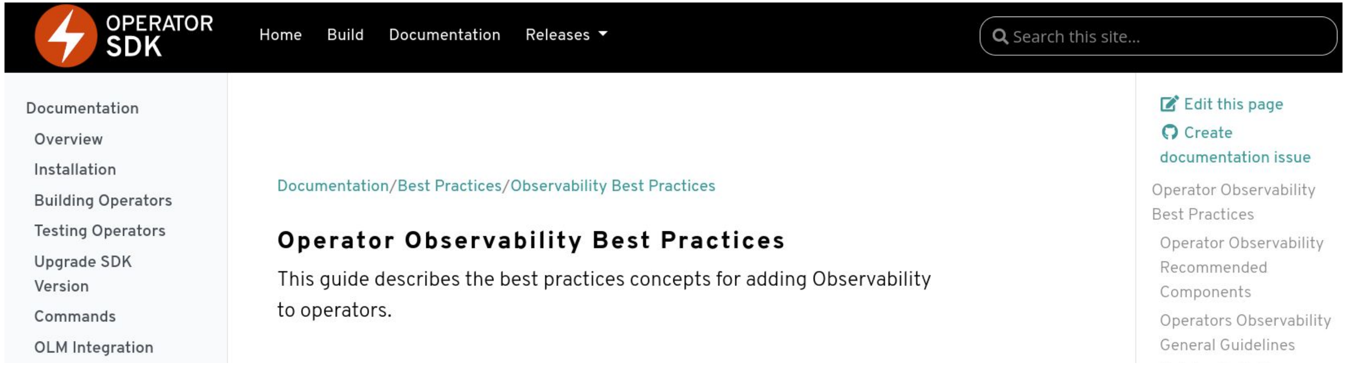
Enable **Proactive actions** and **Educate the user**

Autopilot

Auto-scaling, Auto-healing, Auto-tuning, Abnormality detection

Operator Observability in Operator SDK

- **Tools to build, test and package Operators**
- **Leading practices and code patterns to help prevent reinventing the wheel.**



Documentation/Best Practices/Observability Best Practices

Operator Observability Best Practices

This guide describes the best practices concepts for adding Observability to operators.

https://sdk.operatorframework.io/docs/best-practices/observability-best-practices/

Observability Best Practices - Objectives

The Power of Community

Learn from others mistakes and **avoid known pitfalls**

Shorten on-boarding and coding time

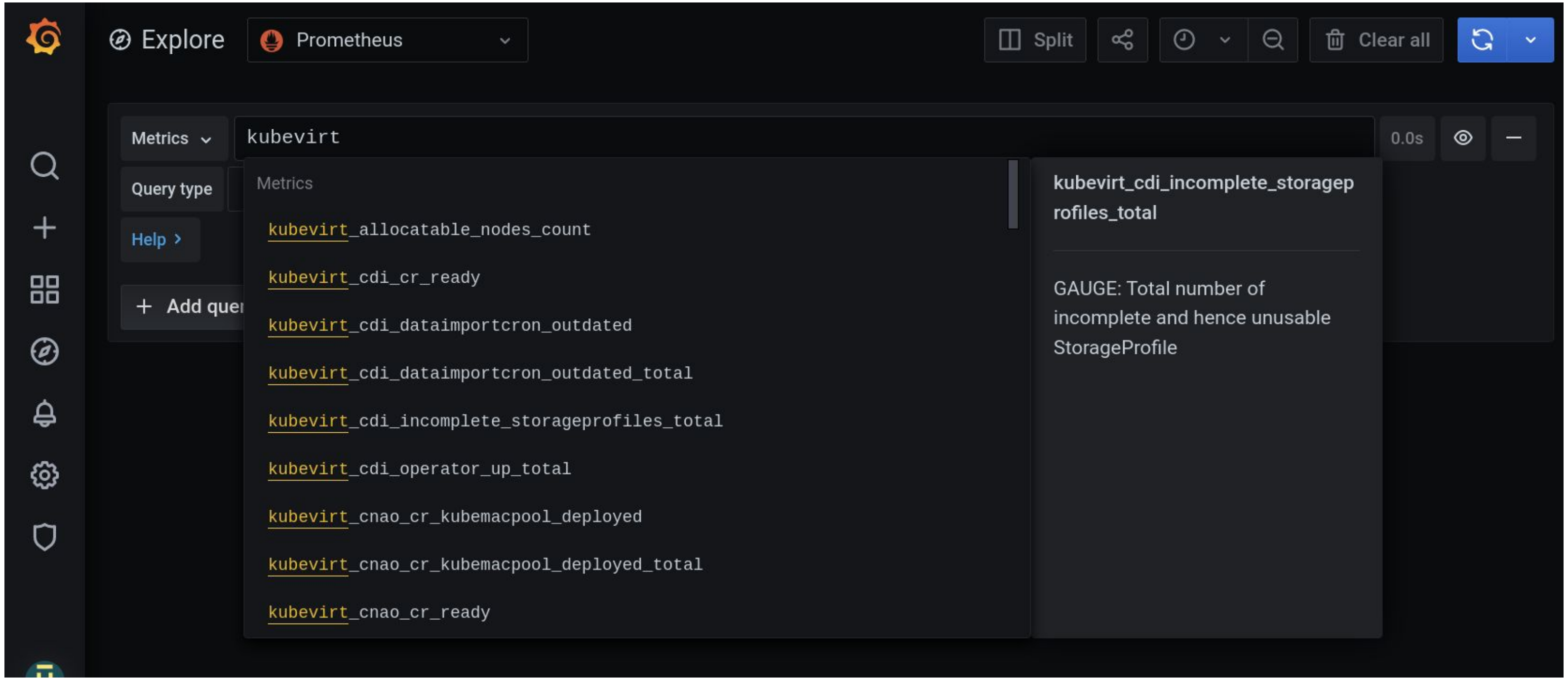
Code examples and documentation for Observability in operators

Reusable Code

Saves Time, Lower Cost, Reduced Development Risks, Prevents Code Bloat

Operator Metrics - Naming Conventions

✓ **Operator Metrics Name Prefix** - Helps to search for all operator metrics



```
kubevirt
```

Metrics

```
kubevirt_allocatable_nodes_count
kubevirt_cdi_cr_ready
kubevirt_cdi_dataimportcron_outdated
kubevirt_cdi_dataimportcron_outdated_total
kubevirt_cdi_incomplete_storageprofiles_total
kubevirt_cdi_operator_up_total
kubevirt_cnao_cr_kubemacpool_deployed
kubevirt_cnao_cr_kubemacpool_deployed_total
kubevirt_cnao_cr_ready
```

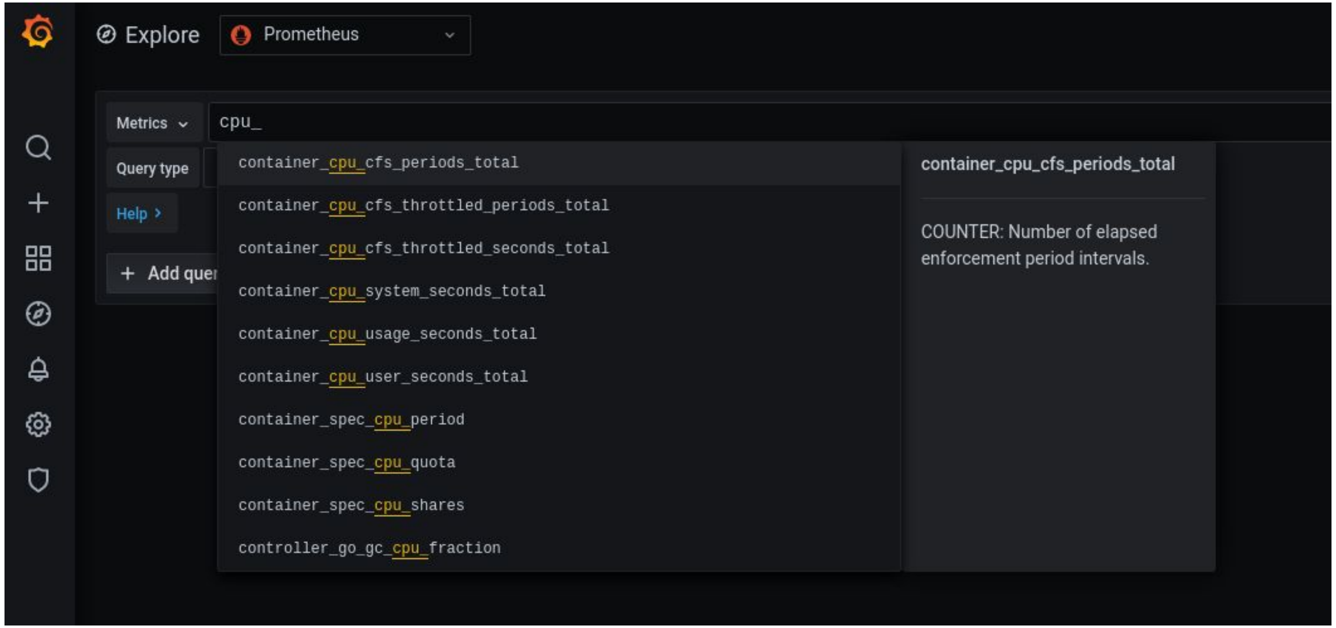
kubevirt_cdi_incomplete_storageprofiles_total

GAUGE: Total number of incomplete and hence unusable StorageProfile

Operator Metrics - Naming Conventions

Operator Metrics Name Prefix - No prefix.

- Hard to understand who generates this (cAdvisor)
- Can’t search all cAdvisor metrics



```
cpu_
container_cpu_cfs_periods_total
container_cpu_cfs_throttled_periods_total
container_cpu_cfs_throttled_seconds_total
container_cpu_system_seconds_total
container_cpu_usage_seconds_total
container_cpu_user_seconds_total
container_spec_cpu_period
container_spec_cpu_quota
container_spec_cpu_shares
controller_go_gc_cpu_fraction
```

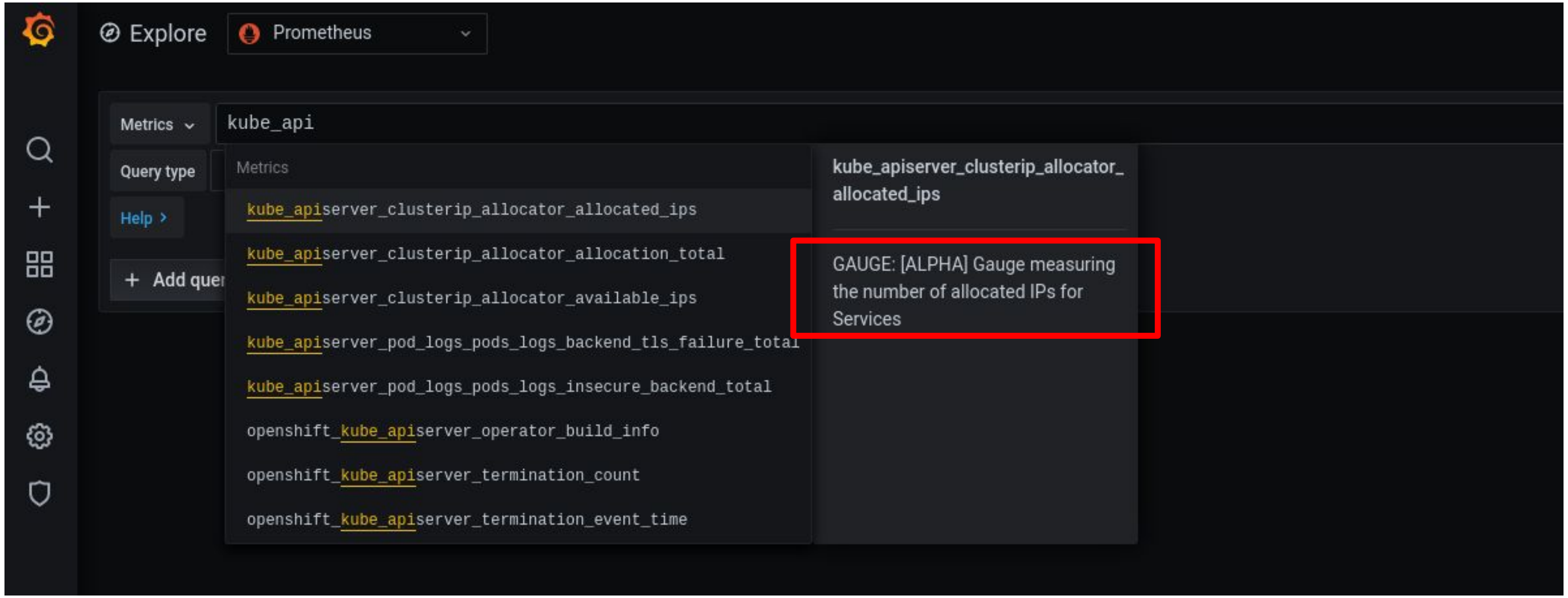
container_cpu_cfs_periods_total

COUNTER: Number of elapsed enforcement period intervals.

Operator Metrics - Help Text

Operator Metric Help Text - When adding a new metric, add a meaningful help text

- Helps users to **learn about the metric**
- Can be used for creating **auto generated documentation**



```
kube_api
```

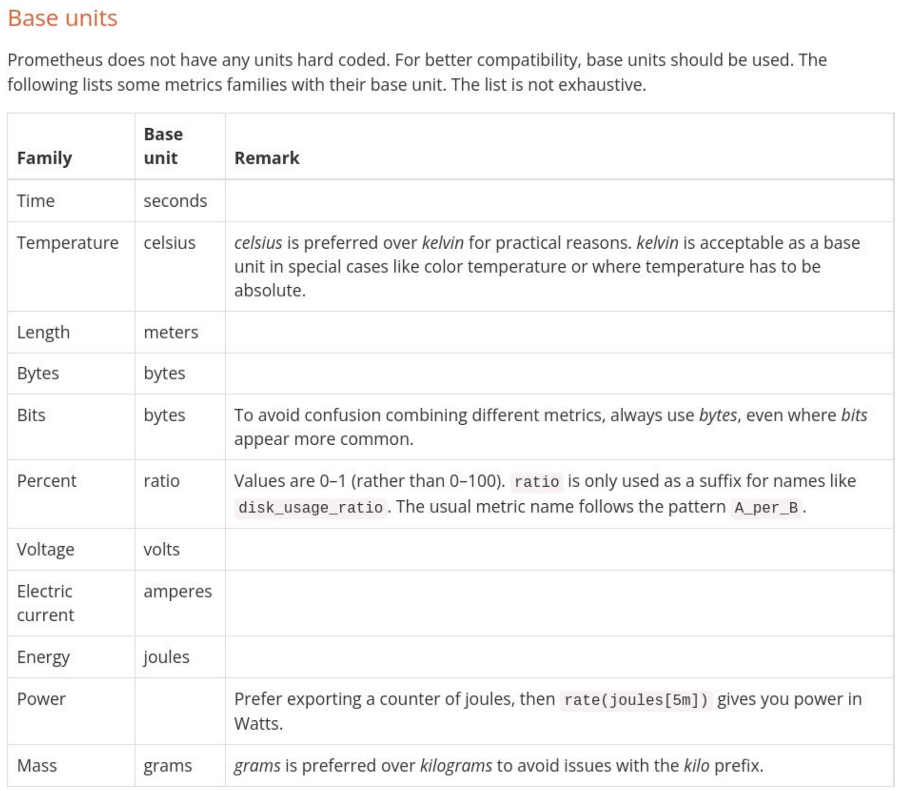
Metrics

```
kube_apiserver_clusterip_allocator_allocated_ips
kube_apiserver_clusterip_allocator_allocation_total
kube_apiserver_clusterip_allocator_available_ips
kube_apiserver_pod_logs_pods_logs_backend_tls_failure_total
kube_apiserver_pod_logs_pods_logs_insecure_backend_total
openshift_kube_apiserver_operator_build_info
openshift_kube_apiserver_termination_count
openshift_kube_apiserver_termination_event_time
```

kube_apiserver_clusterip_allocator_allocated_ips

GAUGE: [ALPHA] Gauge measuring the number of allocated IPs for Services

Operator Metrics - Base Units



Base units

Prometheus does not have any units hard coded. For better compatibility, base units should be used. The following lists some metrics families with their base unit. The list is not exhaustive.

Family	Base unit	Remark
Time	seconds	
Temperature	celsius	*celsius* is preferred over *kelvin* for practical reasons. *kelvin* is acceptable as a base unit in special cases like color temperature or where temperature has to be absolute.
Length	meters	
Bytes	bytes	
Bits	bytes	To avoid confusion combining different metrics, always use *bytes*, even where *bits* appear more common.
Percent	ratio	Values are 0–1 (rather than 0–100). `ratio` is only used as a suffix for names like `disk_usage_ratio`. The usual metric name follows the pattern `A_per_B`.
Voltage	volts	
Electric current	amperes	
Energy	joules	
Power		Prefer exporting a counter of joules, then `rate(joules[5m])` gives you power in Watts.
Mass	grams	*grams* is preferred over *kilograms* to avoid issues with the *kilo* prefix.

https://prometheus.io/docs/practices/naming/#base-units

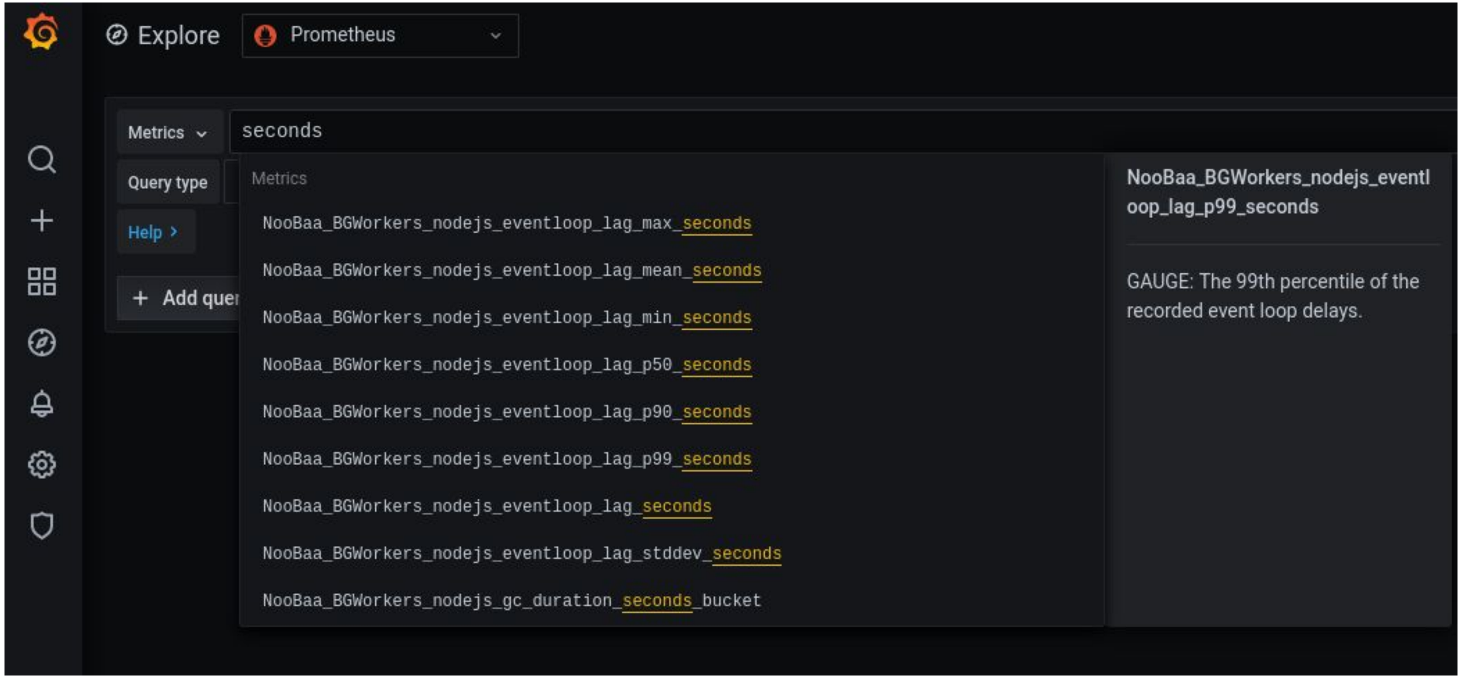
Operator Metrics - Base Units

- The Prometheus approach is to **use base units** such as seconds, and then **floating point numbers** to store the value.
- Floating point **removes the concern of the magnitude of the number**

https://www.robustperception.io/who-wants-seconds/

Operator Metrics - Base Units

✓ **Operator Metrics Units** - Uses Prometheus base units “seconds”



seconds

Metrics

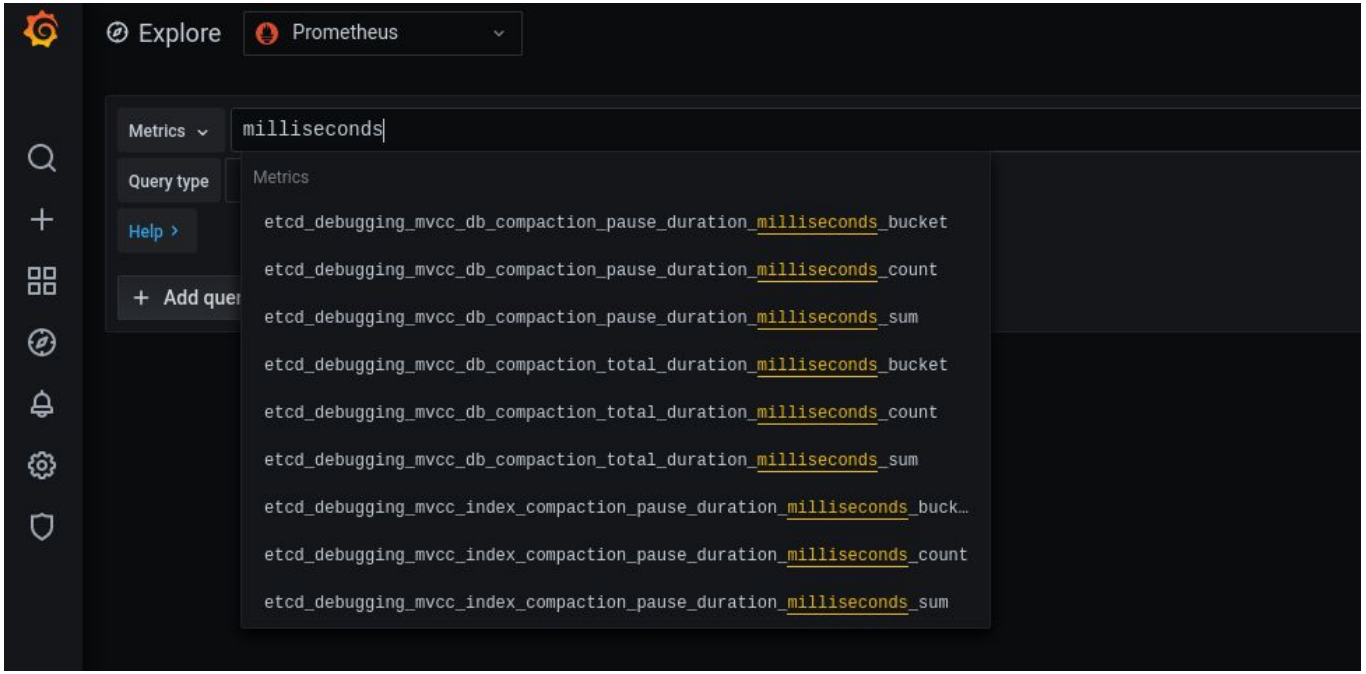
```
NooBaa_BGWorkers_nodejs_eventloop_lag_max_seconds
NooBaa_BGWorkers_nodejs_eventloop_lag_mean_seconds
NooBaa_BGWorkers_nodejs_eventloop_lag_min_seconds
NooBaa_BGWorkers_nodejs_eventloop_lag_p50_seconds
NooBaa_BGWorkers_nodejs_eventloop_lag_p90_seconds
NooBaa_BGWorkers_nodejs_eventloop_lag_p99_seconds
NooBaa_BGWorkers_nodejs_eventloop_lag_seconds
NooBaa_BGWorkers_nodejs_eventloop_lag_stddev_seconds
NooBaa_BGWorkers_nodejs_gc_duration_seconds_bucket
```

NooBaa_BGWorkers_nodejs_eventloop_lag_p99_seconds

GAUGE: The 99th percentile of the recorded event loop delays.

Operator Metrics - Base Units

❌ **Operator Metrics Units -** Uses “milliseconds” instead of “seconds”



```
milliseconds
```

```
etcd_debugging_mvcc_db_compaction_pause_duration_milliseconds_bucket
etcd_debugging_mvcc_db_compaction_pause_duration_milliseconds_count
etcd_debugging_mvcc_db_compaction_pause_duration_milliseconds_sum
etcd_debugging_mvcc_db_compaction_total_duration_milliseconds_bucket
etcd_debugging_mvcc_db_compaction_total_duration_milliseconds_count
etcd_debugging_mvcc_db_compaction_total_duration_milliseconds_sum
etcd_debugging_mvcc_index_compaction_pause_duration_milliseconds_buck…
etcd_debugging_mvcc_index_compaction_pause_duration_milliseconds_count
etcd_debugging_mvcc_index_compaction_pause_duration_milliseconds_sum
```

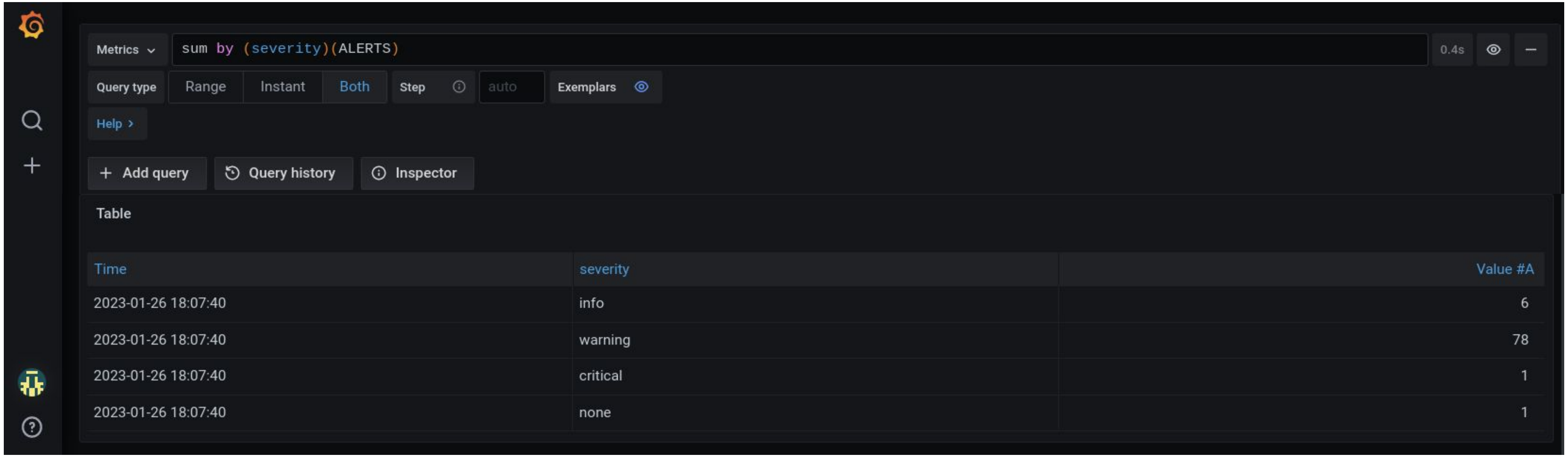
Operator Alerts - Example

```go
// createOperatorDownAlertRule creates MemcachedOperatorDown alert rule
func createOperatorDownAlertRule() monitoringv1.Rule {
        return monitoringv1.Rule{
                Alert: operatorDownAlert,
                Expr:  intstr.FromString("memcached_operator_up_total == 0"),
                Annotations: map[string]string{
                        "description": "No running memcached-operator pods were detected in the last 5 min.",
                },
                For: "5m",
                Labels: map[string]string{
                        "severity":    "critical",
                        "runbook_url": runbookURLBasePath + "MemcachedOperatorDown.md",
                },
        }
}
```

Operator Alerts - Severity Label

✓ **Operator Alerts Severity** - Only 3 valid Severity Label values : **Critical** / **Warning** / **Info**

Note: “none” severity - Only for the Watchdog

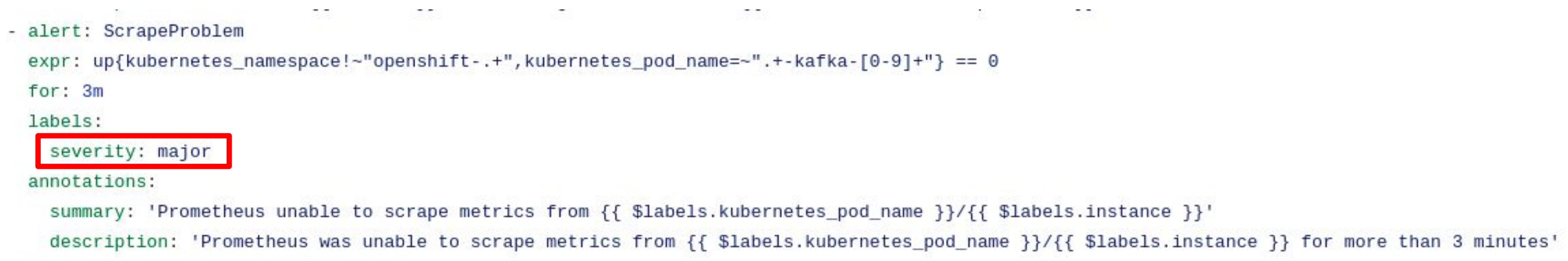


```
sum by (severity)(ALERTS)
```

Time	severity	Value #A
2023-01-26 18:07:40	info	6
2023-01-26 18:07:40	warning	78
2023-01-26 18:07:40	critical	1
2023-01-26 18:07:40	none	1

Operator Alerts - Severity Label

Operator Alerts Severity Label - “Major“ is not a valid value for Severity label



```
- alert: ScrapeProblem
  expr: up{kubernetes_namespace!~"openshift-.+",kubernetes_pod_name=~".+-kafka-[0-9]+"} == 0
  for: 3m
  labels:
    severity: major
  annotations:
    summary: 'Prometheus unable to scrape metrics from {{ $labels.kubernetes_pod_name }}/{{ $labels.instance }}'
    description: 'Prometheus was unable to scrape metrics from {{ $labels.kubernetes_pod_name }}/{{ $labels.instance }} for more than 3 minutes'
```

* Red Hat AMQ Streams

Operators Observability - Best Practices Status

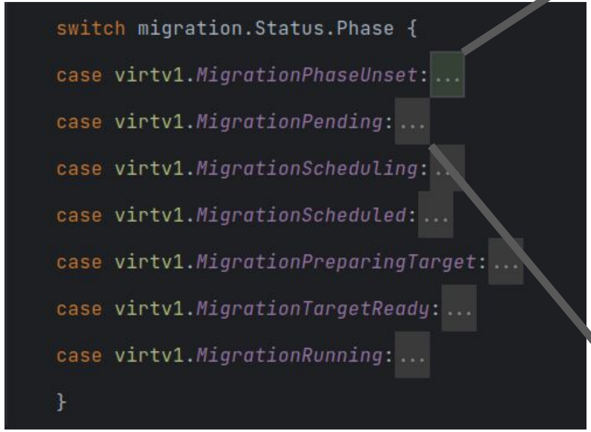
Today:

- **Metrics** naming convention
- **Metrics documentation**
- **Alerts** required labels
- **Alert runbooks**
- Metrics and Alerts **tests**

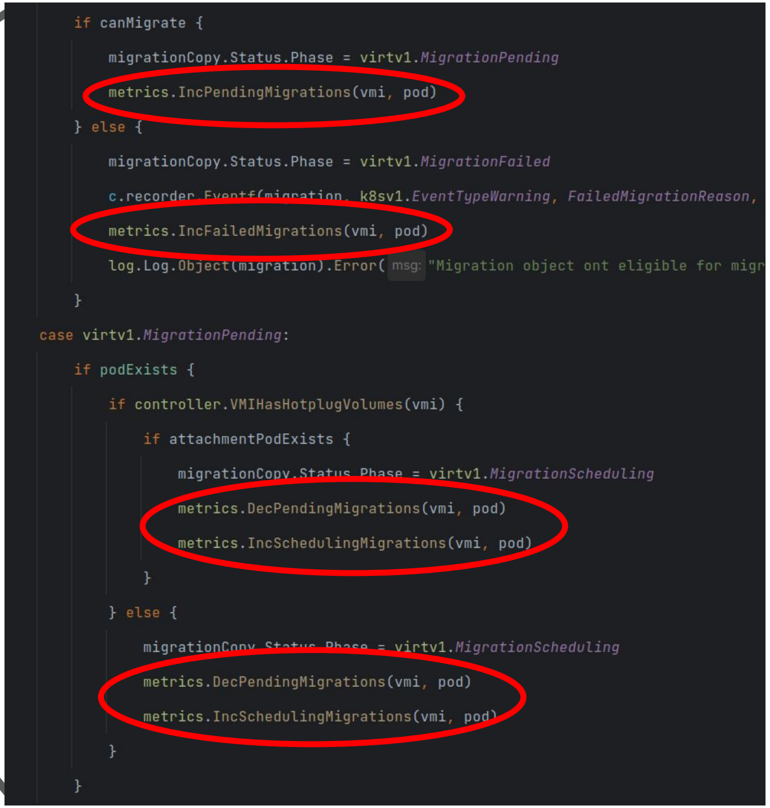
Future:

- **Dashboards**
- **Logging**
- **Tracing**
- **Events**

Managing Code Complexity



```
switch migration.Status.Phase {
case virtv1.MigrationPhaseUnset:...
case virtv1.MigrationPending:...
case virtv1.MigrationScheduling:...
case virtv1.MigrationScheduled:...
case virtv1.MigrationPreparingTarget:...
case virtv1.MigrationTargetReady:...
case virtv1.MigrationRunning:...
}
```

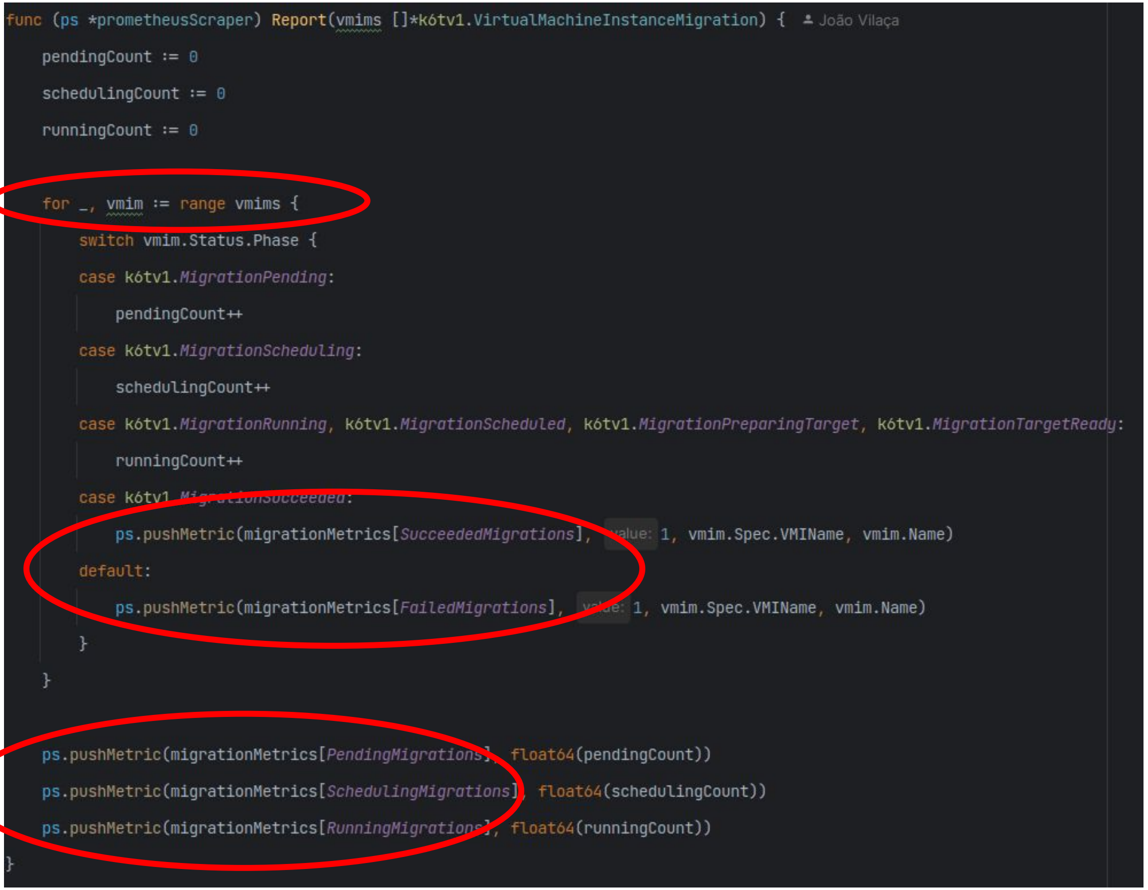


```
    if canMigrate {
        migrationCopy.Status.Phase = virtv1.MigrationPending
        metrics.IncPendingMigrations(vmi, pod)
    } else {
        migrationCopy.Status.Phase = virtv1.MigrationFailed
        c.recorder.Eventf(migration, k8sv1.EventTypeWarning, FailedMigrationReason,
        metrics.IncFailedMigrations(vmi, pod)
        log.Log.Object(migration).Error( msg: "Migration object ont eligible for migr
    }
case virtv1.MigrationPending:
    if podExists {
        if controller.VMIHasHotplugVolumes(vmi) {
            if attachmentPodExists {
                migrationCopy.Status.Phase = virtv1.MigrationScheduling
                metrics.DecPendingMigrations(vmi, pod)
                metrics.IncSchedulingMigrations(vmi, pod)
            }
        } else {
            migrationCopy.Status.Phase = virtv1.MigrationScheduling
            metrics.DecPendingMigrations(vmi, pod)
            metrics.IncSchedulingMigrations(vmi, pod)
        }
    }
```

https://github.com/kubevirt/kubevirt ->
pkg/virt-controller/watch/migration.go

16.4 %

monitoring code intertwined in migration logic code



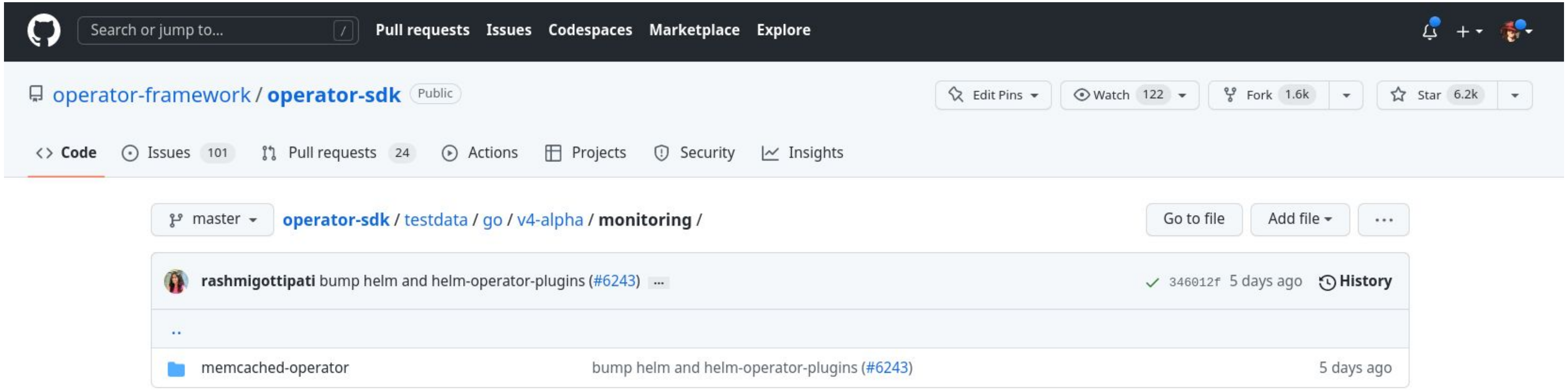
```go
func (ps *prometheusScraper) Report(vmims []*k6tv1.VirtualMachineInstanceMigration) {  João Vilaça
    pendingCount := 0
    schedulingCount := 0
    runningCount := 0

    for _, vmim := range vmims {
        switch vmim.Status.Phase {
        case k6tv1.MigrationPending:
            pendingCount++
        case k6tv1.MigrationScheduling:
            schedulingCount++
        case k6tv1.MigrationRunning, k6tv1.MigrationScheduled, k6tv1.MigrationPreparingTarget, k6tv1.MigrationTargetReady:
            runningCount++
        case k6tv1.MigrationSucceeded:
            ps.pushMetric(migrationMetrics[SucceededMigrations], value: 1, vmim.Spec.VMIName, vmim.Name)
        default:
            ps.pushMetric(migrationMetrics[FailedMigrations], value: 1, vmim.Spec.VMIName, vmim.Name)
        }
    }

    ps.pushMetric(migrationMetrics[PendingMigrations], float64(pendingCount))
    ps.pushMetric(migrationMetrics[SchedulingMigrations], float64(schedulingCount))
    ps.pushMetric(migrationMetrics[RunningMigrations], float64(runningCount))
}
```

Usage of collector approach for VM migration metrics

Memcached Example Operator

An example operator of building a simple operator using tools and libraries provided by the Operator SDK.

https://sdk.operatorframework.io/docs/building-operators/golang/quickstart/



operator-framework / operator-sdk

operator-sdk / testdata / go / v4-alpha / monitoring /

rashmigottipati bump helm and helm-operator-plugins (#6243) ... ✓ 346012f 5 days ago

..

memcached-operator   bump helm and helm-operator-plugins (#6243)   5 days ago

https://github.com/operator-framework/operator-sdk/tree/master/testdata/go/v4-alpha/monitoring/memcached-operator

Operator Observability Code

Clear

Modular

Reusable

Performant

Operator Observability Code

Keep Monitoring Code Separate from Logic Code

```
.
├── pkg/
│   ├── controllers
│   ├── migration
│   └── ...
└── monitoring/
    ├── alerts/
    └── metrics/
        ├── metrics.go
        ├── operator_metrics.go
        ├── migration_metrics.go
        └── ...
```

```go
var metrics = [][]util.Metric{
        OperatorMetrics,
        MigrationMetrics,
}

func RegisterMetrics() {
        util.RegisterMetrics(metrics)
}

func ListMetrics() []util.Metric {
        return util.ListMetrics(metrics)
}
```

Operator Observability in Memcached Operator

Automate metric and alert code generation

Metrics name linter

Automated metrics documentation

Easy structure for unit and e2e testing

Thank you!

Shirly Radco
Principal Software Engineer
Red Hat

João Vilaça
Associate Software Engineer
Red Hat

February 2023