Reducing observability cognitive load in KubeVirt

João Vilaça
Software Engineer at Red Hat

KubeVirt Red Hat

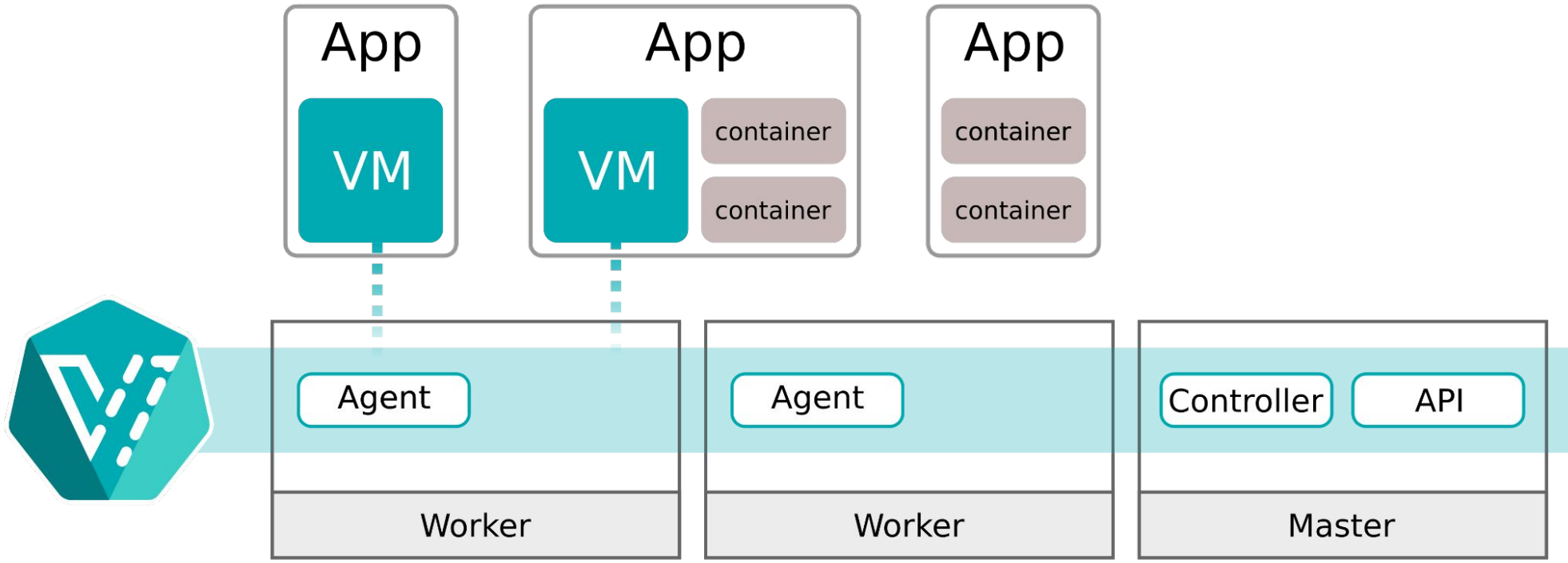
What we'll discuss today

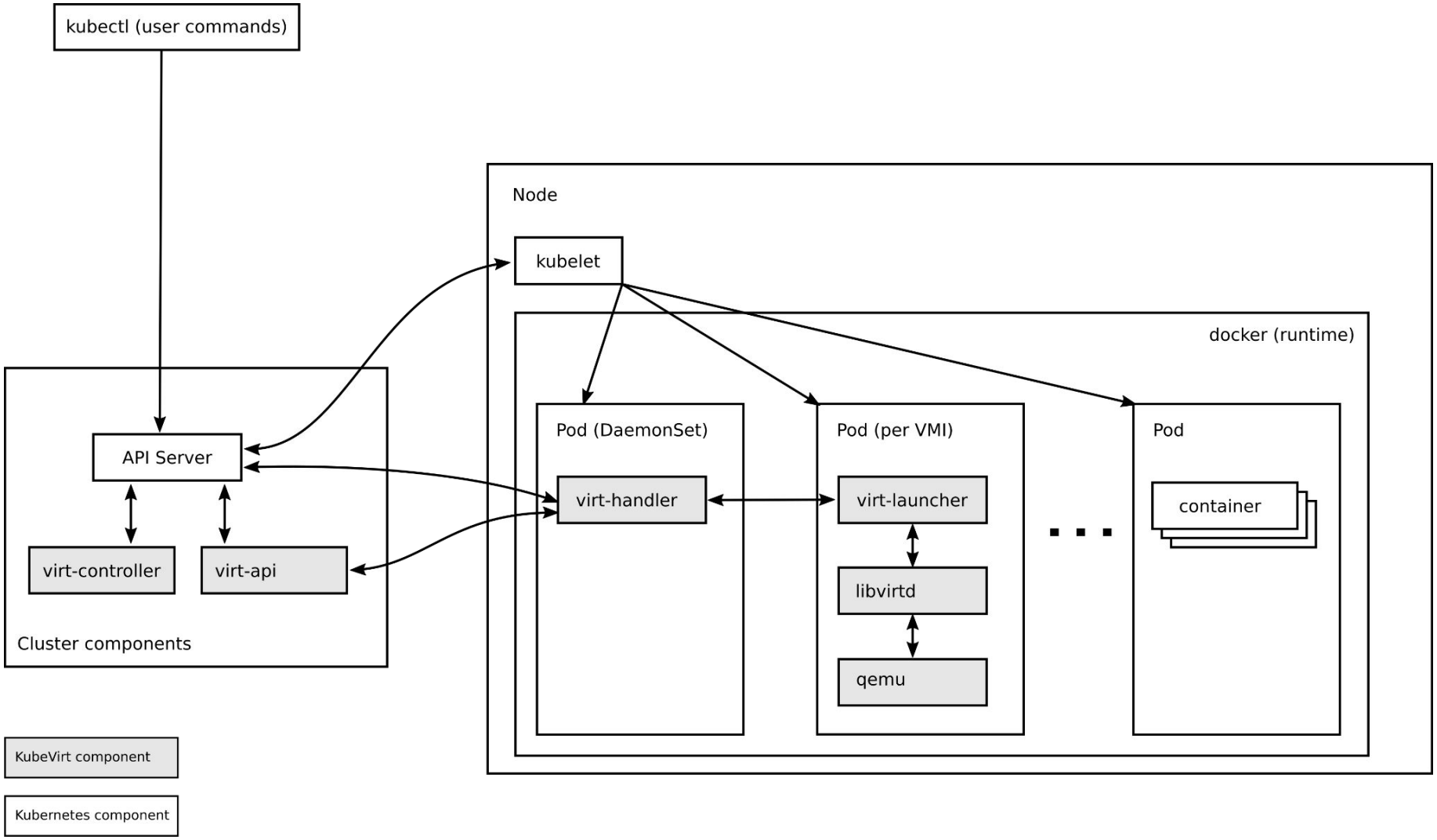
- Background on KubeVirt
- The Observability Challenge
- Our Approach: Modularizing Observability
- Demo
- Lessons Learned & Best Practices

Background on KubeVirt

Kubernetes Virtualization API
and runtime in order to define and manage virtual machines.

- Open source (Apache-2.0 license) CNCF project in incubation state
- 5.8k GitHub stars, 300+ contributors, 1k+ forks, 10k+ PRs
- Vendored and adopted by numerous organizations like Red Hat, Microsoft, Cloudflare, NVIDIA, arm, etc.


App
VM
App
VM
container
container
App
container
container
Agent
Agent
Controller
API
Worker
Worker
Master


kubectl (user commands)
Node
kubelet
docker (runtime)
Pod (DaemonSet)
Pod (per VMI)
Pod
API Server
virt-handler
virt-launcher
container
libvirtd
qemu
virt-controller
virt-api
Cluster components
KubeVirt component
Kubernetes component

The Observability Challenge

Need for a Dedicated Team

- Increased project maturity and complex business requirements, alongside growing number of end-users and vendors
- Requests for new observability features and signals
- Observability became a “first-class“ concern
- Formation of a specialized observability team

The “wild west” approach

- Each team or developer added Prometheus metrics in their own style across different codebases
- High cognitive load to maintain an internal mental model of metrics
- Business logic intertwined with observability logic
- Inconsistencies in naming conventions, labels, and best practices
- Hard to add new features since they are independently created in many different places

Our Approach: Modularizing Observability

Repositories Managed

- github.com/kubevirt/kubevirt
- github.com/kubevirt/hyperconverged-cluster-operator
- github.com/kubevirt/ssp-operator
- github.com/kubevirt/cluster-network-addons-operator
- github.com/kubevirt/containerized-data-importer
- github.com/kubevirt/hostpath-provisioner-operator
- github.com/kubevirt/hostpath-provisioner

proposal: Monitoring code refactor
https://github.com/kubevirt/community/pull/219

Goals

- Decouple monitoring logic from business logic
- Encapsulate the monitoring best-practices and the common patterns into a library and have it as a dependency for all KubeVirt components
- Keep monitoring code and utilities easy to maintain and evolve
- Have a structure and tools to accurately and easily generate monitoring documentation, lint metrics and alerts, define allow/deny/opt-in lists and other features without having to change the code in multiple places

Strict Interface and Dependency Model

```go
// kubevirt/pkg/monitoring/metrics/virt-controller/migration_metrics.go
var (
  migrationMetrics = []operatormetrics.Metric{
    vmiMigrationPhaseTransitionTimeFromCreation,
    ...
  }

  vmiMigrationPhaseTransitionTimeFromCreation = operatormetrics.NewHistogramVec(
    operatormetrics.MetricOpts{
      Name: "kubevirt_vmi_migration_phase_transition_time_from_creation_seconds",
      Help: "VM migration phase transitions duration from creation time in seconds.",
    },
    prometheus.HistogramOpts{
      Buckets: PhaseTransitionTimeBuckets(),
    },
    []string{"phase"},
  )
)

func CreateVMIMigrationHandler(informer cache.SharedIndexInformer) error {
  _, err := informer.AddEventHandler(cache.ResourceEventHandlerFuncs{
    UpdateFunc: func(oldVMIMigration, newVMIMigration interface{}) {
      updateVMIMigrationPhaseTransitionTime(oldVMIMigration, newVMIMigration)
    },
  })

  return err
}

func updateVMIMigrationPhaseTransitionTime(
  oldVMIMigration *v1.VirtualMachineInstanceMigration,
  newVMIMigration *v1.VirtualMachineInstanceMigration,
) {
  // calculation logic

  histogram.Observe(diffSeconds)
}
```


```go
// kubevirt/pkg/monitoring/metrics/virt-controller/migrationstats_collector.go
var (
  migrationStatsCollector = operatormetrics.Collector{
    Metrics: []operatormetrics.Metric{
      pendingMigrations,
      schedulingMigrations,
      runningMigrations,
      succeededMigration,
      failedMigration,
    },
    CollectCallback: migrationStatsCollectorCallback,
  }

  pendingMigrations = operatormetrics.NewGauge(
    operatormetrics.MetricOpts{
      Name: "kubevirt_vmi_migrations_in_pending_phase",
      Help: "Number of current pending migrations.",
    },
  )

  ...
)

func migrationStatsCollectorCallback() []operatormetrics.CollectorResult {
  cachedObjs := vmiMigrationInformer.GetIndexer().List()
  vmims := make([]*k6tv1.VirtualMachineInstanceMigration, len(cachedObjs))
  ...
  return reportMigrationStats(vmims)
}

func reportMigrationStats(vmims []*k6tv1.VMIM) []operatormetrics.CollectorResult {
  // calculation logic

  return append(cr,
    operatormetrics.CollectorResult{Metric: pendingMigrations, Value: pendingCount},
    operatormetrics.CollectorResult{Metric: schedulingMigrations, Value: schedulingCount},
    operatormetrics.CollectorResult{Metric: runningMigrations, Value: runningCount},
  )
}
```


```go
// kubevirt/pkg/monitoring/metrics/virt-controller/metrics.go
var (
  metrics = [][]operatormetrics.Metric{
    componentMetrics,
    migrationMetrics,
    perfscaleMetrics,
    vmiMetrics,
    vmSnapshotMetrics,
  }
)

func SetupMetrics(...) {
  ...
  if err := operatormetrics.RegisterMetrics(metrics...); err != nil {
    return err
  }

  return operatormetrics.RegisterCollector(
    migrationStatsCollector,
    vmiStatsCollector,
    vmStatsCollector,
  )
}
```


Refactor monitoring metrics

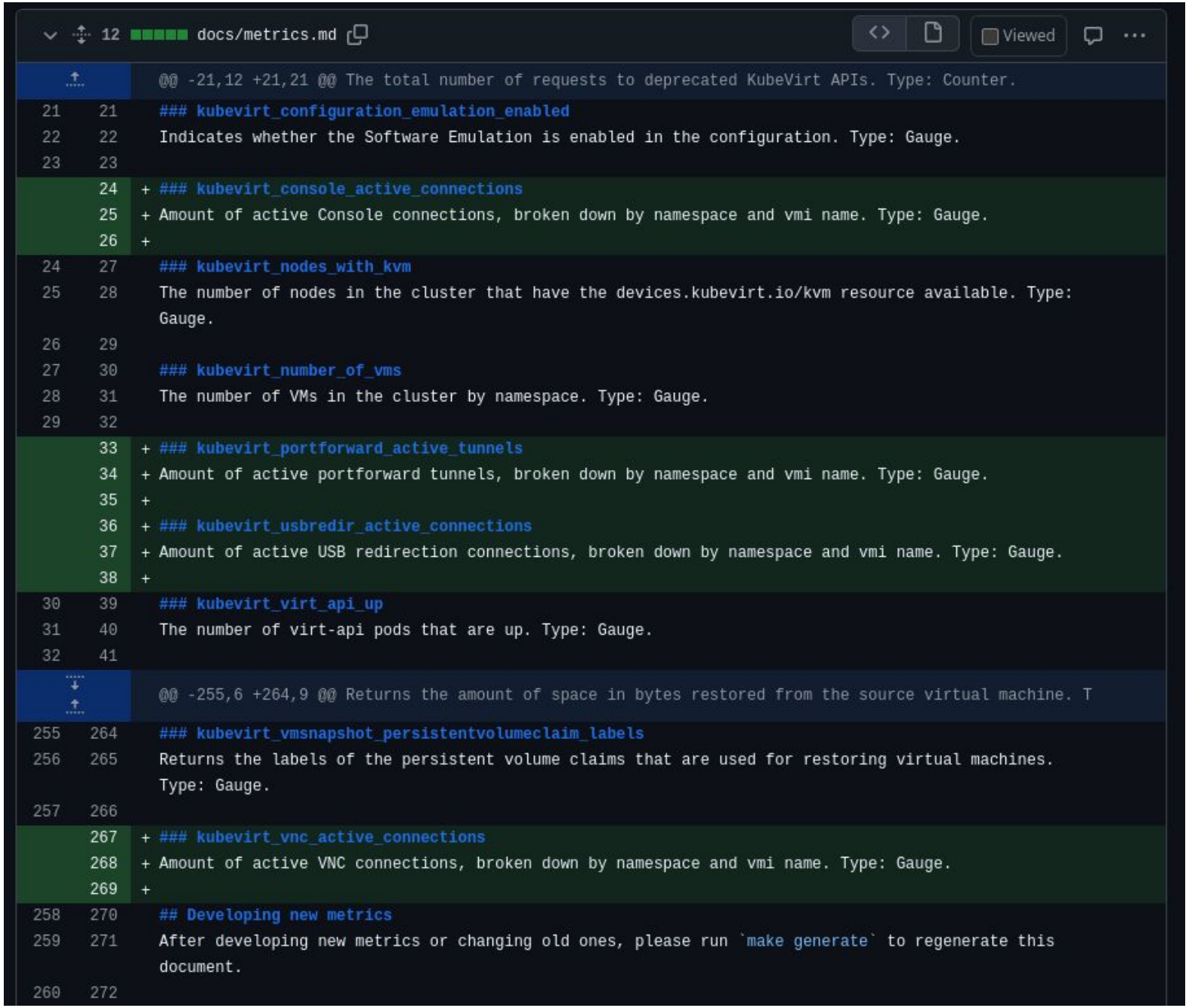
https://github.com/kubevirt/kubevirt/pull/10982

Our Approach: Modularizing Observability

40 pkg/virt-operator/application.go

Viewed

```
113  111
114      - var (
115      -         _ service.Service = &VirtOperatorApp{}
116      -
117      -         leaderGauge = prometheus.NewGauge(
118      -                 prometheus.GaugeOpts{
119      -                         Name: "kubevirt_virt_operator_leading_status",
120      -                         Help: "Indication for an operating virt-operator.",
121      -                 },
122      -         )
123      -
124      -         readyGauge = prometheus.NewGauge(
125      -                 prometheus.GaugeOpts{
126      -                         Name: "kubevirt_virt_operator_ready_status",
127      -                         Help: "Indication for a virt-operator that is ready to take the lead.",
128      -                 },
129      -         )
130      - )
131      -
132      - func init() {
133      -         prometheus.MustRegister(leaderGauge)
134      -         prometheus.MustRegister(readyGauge)
135      - }
136      -
137  112     func Execute() {
```


12 docs/metrics.md
Viewed
@@ -21,12 +21,21 @@ The total number of requests to deprecated KubeVirt APIs. Type: Counter.
kubevirt_configuration_emulation_enabled
Indicates whether the Software Emulation is enabled in the configuration. Type: Gauge.
+### kubevirt_console_active_connections
+Amount of active Console connections, broken down by namespace and vmi name. Type: Gauge.
+
kubevirt_nodes_with_kvm
The number of nodes in the cluster that have the devices.kubevirt.io/kvm resource available. Type: Gauge.
kubevirt_number_of_vms
The number of VMs in the cluster by namespace. Type: Gauge.
+### kubevirt_portforward_active_tunnels
+Amount of active portforward tunnels, broken down by namespace and vmi name. Type: Gauge.
+
+### kubevirt_usbredir_active_connections
+Amount of active USB redirection connections, broken down by namespace and vmi name. Type: Gauge.
+
kubevirt_virt_api_up
The number of virt-api pods that are up. Type: Gauge.
@@ -255,12 +264,9 @@ Returns the labels of the persistent volume claims that are used for restoring v
Type: Gauge.
+### kubevirt_vnc_active_connections
+Amount of active VNC connections, broken down by namespace and vmi name. Type: Gauge.
+
Developing new metrics
After developing new metrics or changing old ones, please run `make generate` to regenerate this document.

Enforced Validations

```
// operator-observability/pkg/testutil/alert_validation.go
var defaultAlertValidations = []AlertValidation{
        validateAlertName,
        validateAlertHasExpression,
        validateAlertHasSeverityLabel,
        validateAlertHasSummaryAnnotation,
}

// operator-observability/pkg/testutil/alert_custom_validations.go
func ValidateAlertNameLength(alert *promv1.Rule) []Problem
func ValidateAlertHasDescriptionAnnotation(alert *promv1.Rule) []Problem
func ValidateAlertRunbookURLAnnotation(alert *promv1.Rule) []Problem
func ValidateAlertHealthImpactLabel(alert *promv1.Rule) []Problem
func ValidateAlertPartOfAndComponentLabels(alert *promv1.Rule) []Problem

// kubevirt/pkg/monitoring/rules/rules_test.go
It("Should validate alerts", func() {
  linter.AddCustomAlertValidations(
    testutil.ValidateAlertNameLength,
    testutil.ValidateAlertRunbookURLAnnotation,
    testutil.ValidateAlertHealthImpactLabel,
    testutil.ValidateAlertPartOfAndComponentLabels,
  )

  problems := linter.LintAlerts(rules.ListAlerts())
  Expect(problems).To(BeEmpty())
})
```


Easier and more complete unit-tests

```
// kubevirt/pkg/monitoring/metrics/virt-controller/vmistats_collector_test.go
DescribeTable(
  "should show instance type value correctly",
  func(
    instanceTypeKind string,
    instanceTypeName string,
    expected string,
  ) {
  ...

  vms := []*k6tv1.VirtualMachine{{
    Spec: k6tv1.VirtualMachineSpec{Instancetype: instanceType}
  }}

  crs := CollectVMsInfo(vms)
  Expect(crs).To(HaveLen(1), "Expected 1 metric")
  ...
  Expect(cr.GetLabelValue("instance_type")).To(Equal(expected))
},

  Entry("no instance type", "Instancetype", "", "<none>"),
  Entry("managed instance type", "Instancetype", "i-managed", "i-managed"),
  Entry("custom instance type", "Instancetype", "i-unmanaged", "<other>"),
  Entry("cluster instance type", "ClusterInstancetype", "", "<none>"),
  Entry("managed cluster instance type", "ClusterInstancetype", "ci-managed", "ci-managed"),
  Entry("custom cluster instance type", "ClusterInstancetype", "ci-unmanaged", "<other>"),
)
```


Clearly bounded ownership

```
#
# SIG Observability
# Owns the responsibility to keep kubevirt observable by i.e.
# having mertics, alters, and runbooks.
#
sig-observability-approvers:
  - sradco
  - machadovilaca
sig-observability-reviewers:
  - machadovilaca
  - avlitman
  - nunnatsa
  - orenc1
```


Challenges in Transition

- Overhead creating the package
- Huge and complex refactor work across all the repositories
- Educating developers
- Golang linter designed to ensure that in Kubernetes operator projects, monitoring-related practices are implemented within the *pkg/monitoring* directory using *operator-observability*

Demo

Lessons Learned & Best Practices

- Reduce Technical Debt Early
- Encourage a Dedicated Observability Mindset
- Library-Based Approach
- Continuous Iteration

Thank you

linkedin.com/in/machadovilaca

x.com/machadovilaca

KubeVirt

Red Hat