







# Forklift Validation Service

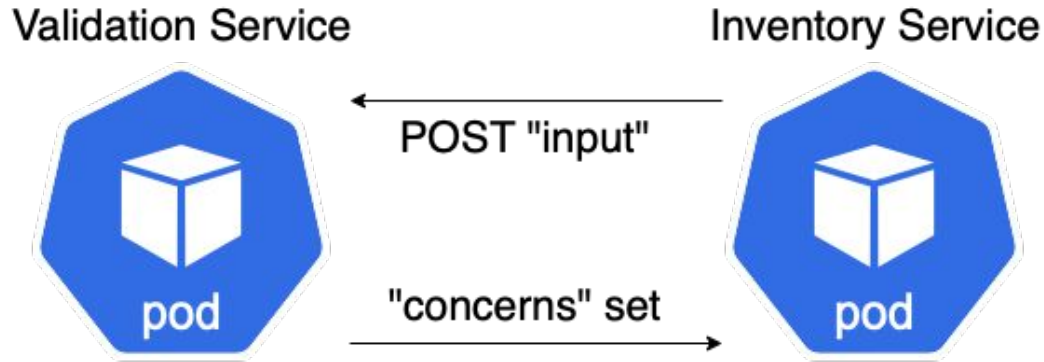
Forklift Hackathon 2021

# Provides “conditions” for the plan wizard to display...

>	<input type="checkbox"/>	 Warning	globex-esb-do...	V2V-DC	V2V_Cluster	esx12.v2v.bos.r...	
▼	<input type="checkbox"/>	 Warning	globex-oracled...	V2V-DC	V2V_Cluster	esx12.v2v.bos.r...	
<p>Conditions have been identified that make this VM a <b>moderate risk</b> to migrate.</p> <ul style="list-style-type: none"><li> <b>VM running in a DRS-enabled cluster:</b> Distributed resource scheduling is not currently supported by OpenShift Virtualization. The VM can be migrated but it will not have this feature in the target environment.</li><li> <b>Changed Block Tracking (CBT) not enabled:</b> Changed Block Tracking (CBT) has not been enabled on this VM. This feature is a prerequisite for VM warm migration.</li></ul> <p>See the <a href="#">product documentation</a> for more information.</p>							
>	<input type="checkbox"/>	 Advisory	ibragins-rhel8-...	V2V-DC	V2V_Cluster	esx12.v2v.bos.r...	V2V
>	<input type="checkbox"/>	 Critical	jortel-testing/...	V2V-DC	V2V_Cluster	esx12.v2v.bos.r...	Workloads/jortel/...

# Used by the Inventory Service

Validation service determines “concerns” that should be applied to VMs before migration.



The concerns set for each VM is generated by the validation service, but stored in the provider inventory as an attribute of each VM.

# Calling the Validation Service

After retrieving VM inventory from the source provider, the inventory service calls the validation service once for each VM.

Call is made using a RESTful POST to the `forklift-validation` service e.g.

```
POST  
https://forklift-validation/v1/data/io/konveyor/forklift/vmware/validate
```

# Calling the Validation Service - continued

The POST is made with a large JSON “input” body, e.g.

```
{
  "input": {
    "selfLink": "providers/vsphere/...0324e/workloads/vm-431",
    "id": "vm-431",
    "parent": {
      "kind": "Folder",
      "id": "group-v22"
    },
    "revision": 1,
    "name": "pemcg-iscsi-target",
    ...
  }
}
```

# input

*input* is a JSON structure containing attributes for VM, host & cluster...

```
...  
  "changeTrackingEnabled": false,  
  "cpuAffinity": [  
    0,  
    2  
  ],  
  "cpuHotAddEnabled": true,  
  "faultToleranceEnabled": false,  
  "cpuCount": 2,  
  "coresPerSocket": 1,  
  "memoryMB": 2048,  
  ...
```

# Output/Return From the Validation Service

The JSON body “result” from the validation service is the **concerns** set:

```
{
  "result": {
    "concerns": [
      {
        "assessment": "Distributed resource scheduling is not currently
supported by OpenShift Virtualization. The VM can be migrated but it will not have
this feature in the target environment.",
        "category": "Information",
        "label": "VM running in a DRS-enabled cluster"
      },
      ...
    ]
  }
}
```

The inventory service uses this to populate a **concerns** list associated with the VM's record in the inventory database.

# Open Policy Agent (OPA)

Validation Service uses OPA, and rules are written in the *Rego* policy language.

There is a separate Rego file for each validation requirement test.

The OPA rules can refer to any key in the JSON *input* structure.

Each Rego file defines a set rule called **concerns**. This rule in each file tests for a specific condition, and if true it adds a {"category", "label", "assessment"} hash to the global **concerns** set.



# Rule File Example

```
package io.konveyor.forklift.ovirt
```

← module namespace (mandatory)

```
default has_usb_enabled = false
```

← default value for rule (optional)

```
has_usb_enabled {  
    input.usbEnabled  
}
```

← rule

```
concerns[flag] {  
    has_usb_enabled  
    flag := {
```

← each rego file has a **concerns** set rule

← call the rule defined above

← define the category, label & assessment

```
        "category": "Warning",  
        "label": "USB support enabled",  
        "assessment": "The VM has USB support enabled, but USB devices are  
not currently supported by OpenShift Virtualization."  
    }  
}
```

# Policy Rule Testing

Each of the rego rules has a corresponding unit test that exercises the conditions that would trigger the rule.

There are tests for each rule, and the tests are run as a complete set, rather than individually.

```
$ opa test . --explain full
```

```
data.io.konveyor.forklift.vmware.test_with_changed_block_tracking_enabled: PASS (1.434945ms)
```

```
data.io.konveyor.forklift.vmware.test_with_changed_block_tracking_disabled: PASS (410.751µs)
```

```
data.io.konveyor.forklift.vmware.test_without_cpu_affinity#01: PASS (397.413µs)
```

```
data.io.konveyor.forklift.vmware.test_with_cpu_affinity#01: PASS (392.375µs)
```

```
...
```

# Rule Test Example

```
package io.konveyor.forklift.ovirt

test_without_usb_enabled {
    mock_vm := { "name": "test",
                  "usbEnabled": false
                }
    results = concerns with input as mock_vm
    count(results) == 0
}

test_with_usb_enabled {
    mock_vm := { "name": "test",
                  "usbEnabled": true
                }
    results = concerns with input as mock_vm
    count(results) == 1
}
```

# Validation Service Testing

Create personal *forklift-validation* image in quay.io

- Link forked forklift-validation repo to personal quay.io account
- Ensure new validation image is built on a git push

Create a secure route to the forklift-validation service in cluster

Scale down the forklift-operator deployment

Edit forklift-validation deployment YAML to add path to quay.io image

Use curl or Postman to test new image

# Rule Examples

## Flagging if a Boolean Attribute is True

This is the most simple type of test, and flags if an attribute is set to *true*

```
dpm_enabled {  
    input.host.cluster.dpmEnabled  
}
```

```
concerns[flag] {  
    dpm_enabled  
    ...  
}
```

## Flagging if a Boolean Attribute is False

This flags if an attribute is set to *false*

```
change_tracking_disabled {  
    not input.changeTrackingEnabled  
}
```

```
concerns[flag] {  
    change_tracking_disabled  
    ...  
}
```

## Flagging if a Boolean Attribute is True in a Repeating Structure

This flags if a value is *true* anywhere in a possibly repeating structure of objects, such as multiple disks or NICs

```
has_rdm_disk {  
    input.disks[_].rdm  
}
```

```
concerns[flag] {  
    has_rdm_disk  
    ...  
}
```



## Flagging if a Boolean Attribute is False in a Repeating Structure

This flags if a value is *false* anywhere in a possibly repeating structure of objects.

```
unplugged_nics {  
    input.nics[_].plugged == false  
    // Can't use: not input.nics[_].plugged  
}  
  
concerns[flag] {  
    unplugged_nics  
    ...  
}
```

## Flagging if a Boolean Attribute is False in a Repeating Structure - 2

Alternatively (using a counter):

```
unplugged_nics [i] {  
    some i  
    input.nics[i].plugged == false  
}
```

```
concerns[flag] {  
    count(unplugged_nics) > 0  
    ...  
}
```

## Flagging if a Text Attribute is Set to a Particular Value

Use a **default** to prevent the rule returning `undefined`.

```
default warn_placement_policy = false
```

```
warn_placement_policy {  
    regex.match(`\bmigratable\b`, input.placementPolicyAffinity)  
}
```

```
concerns[flag] {  
    warn_placement_policy  
    ...
```

## Flagging if a List Attribute is not Empty

```
default has_cpu_affinity = false

has_cpu_affinity {
    count(input.cpuAffinity) != 0
}

concerns[flag] {
    has_cpu_affinity
    ...
}
```

## Flagging if Any of Several Attributes is True

The same rule name defined multiple times results in the rules being OR'd together:

```
default has_hotplug_enabled = false
```

```
has_hotplug_enabled {  
    input.cpuHotAddEnabled  
}
```

```
has_hotplug_enabled {  
    input.memoryHotAddEnabled  
}
```

```
concerns[flag] {  
    has_hotplug_enabled  
    ...  
}
```

# Summary Steps for Testing/Contributing

1. Read the *Konveyor Forklift Validation Service Implementation Doc*
2. Fork repo: <https://github.com/konveyor/forklift-validation>
3. Link repo to personal quay.io account so that new validation image is built on git push.
4. Deploy OCP cluster with OCPv & MTV, & add source provider
5. Add secure route to forklift-validation service.
6. Scale down forklift operator to zero pods (so that it doesn't try to overwrite changes).
7. Add or edit validation rule and test rego files, & test locally using `opa test`
8. When happy, commit & push. Check that new image is built in quay.io
9. Edit yaml for validation service deployment to install new image from personal quay.io account.
10. Scale down/up validation service deployment & ensure new pod with updated image is loaded.
11. Check pod logs for errors, & test using POST to forklift-validation route.

Questions?