The Operator Framework

Automation, automation, automation

Lucas Caparelli

Ricardo Zanini

Have you ever been woken up in the middle of the night because of outages?

Imagine you run a Nexus cluster

Nexus is a repository manager. It allows you to proxy, collect, and manage your dependencies

Being the cluster's administrator requires specific knowledge about Nexus.

You must know how to:

- Join a new node to the cluster, which means:
  - Configuring each node to match requirements
  - Making connections to persistent storage
  - Making existing members aware of it
- Back up the repository data and configuration
- Upgrade the cluster to new versions

You have friend in the company

The application he’s in charge of is a REST API deployment that runs on Openshift

The morning after the outage, right after you *just* brought the cluster back up, he mentions his app also went through an outage:

- Several endpoints failed
- Openshift recovered from it automatically by re-deploying the necessary containers
- Their outage lasted a few minutes and required no human intervention
- Your friend had a great night of sleep
- You spent the night fixing your outage
- If it wasn’t for coffee *your brain* would be going through an outage

You would love not to be disturbed at night

You would also love to reduce total downtime of your cluster

Well, why not just host your cluster on Openshift as well? Unfortunately, it’s not so simple:

- Kubernetes core API needs to be generalist
- It’s not aware of specific requirements to re-create state, as this varies heavily from one application to another
- Stateless is easy
- Stateful is **hard**!

That’s where the Operator Framework comes in

An Operator is a piece of software which knows the specifics of how to manage an application

In few words, it automates the work of a human system operator/administrator by extending the Kubernetes API.

- Pattern first introduced by CoreOS
- Encodes SRE (Site Reliability Engineering) knowledge into software
- Allows Kubernetes to understand how to recreate state
- A method of packaging, deploying and managing a Kubernetes application

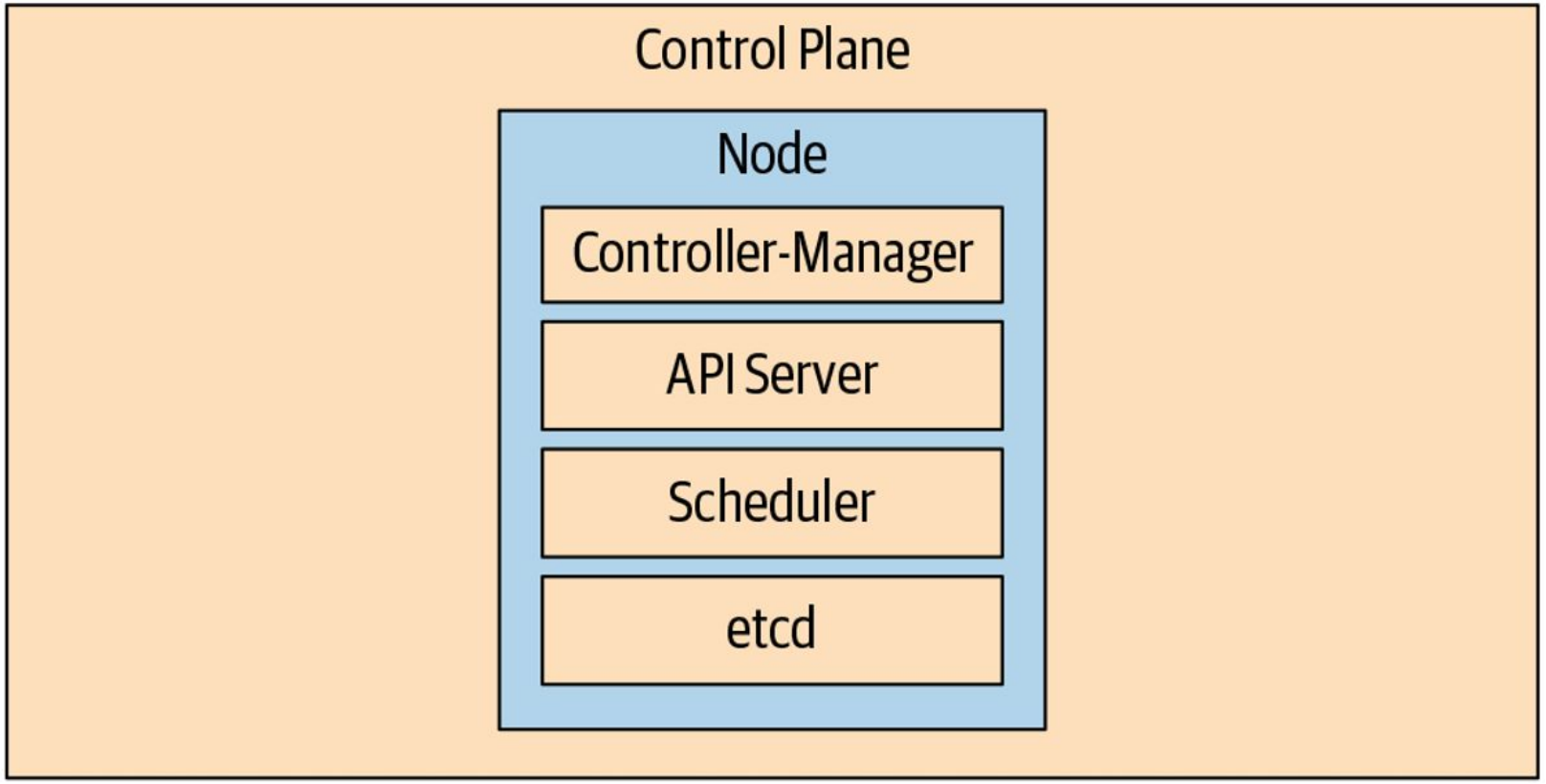
Understanding Kubernetes is key to understand Operators

Operators build on top of the pre-existing Kubernetes API

Some concepts we need before moving on:

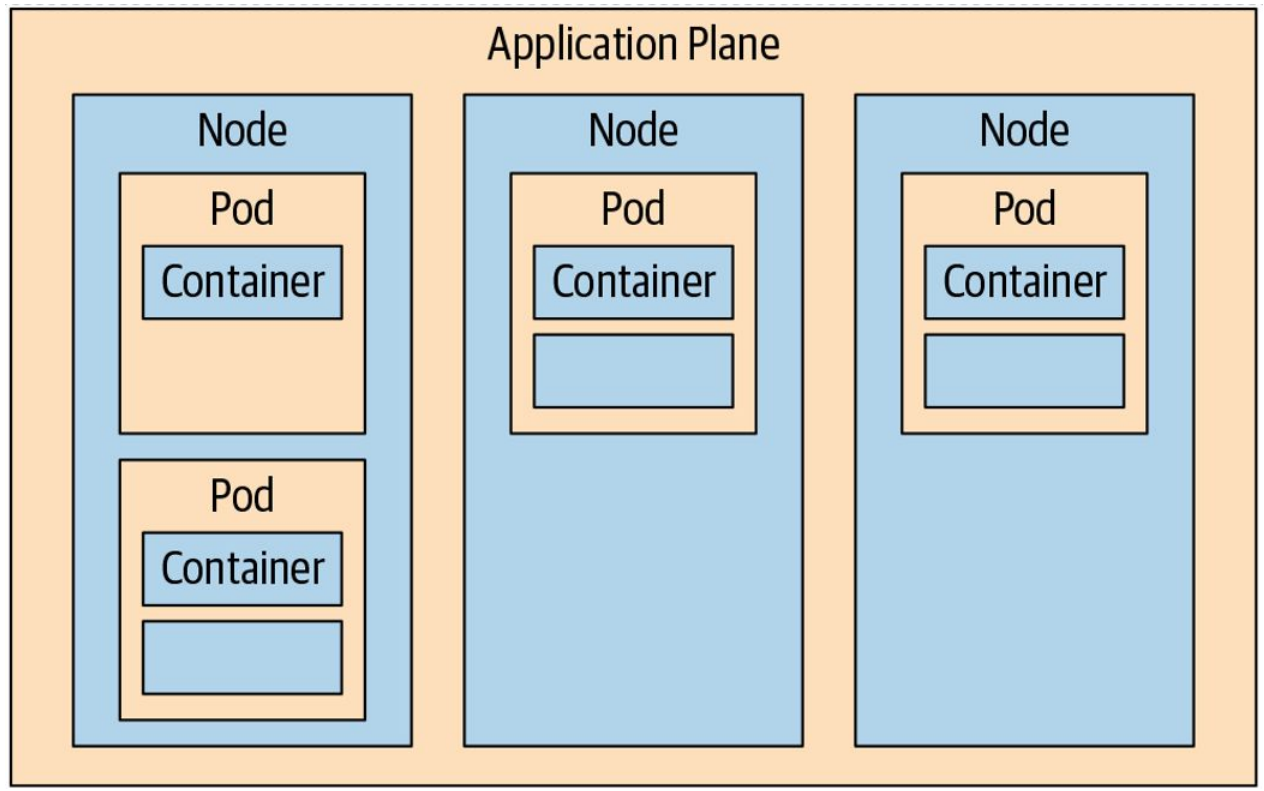
- Container: a lightweight and portable executable image that contains software and all of its dependencies
- Pod: a group of one or more Linux containers with common resources like networking, storage, and access to shared memory
- Node: a computer (physical or virtual) that is part of the Kubernetes cluster. Pods run on nodes
- Replica: a copy of an application running on the cluster

An image is worth a thousand words



Source:
Kubernetes Operators by
Jason Dobies and Joshua Wood (O'Reilly). Copyright 2020 Red Hat, Inc.,
978-1-492-04805-3.

Two images are worth two thousand words… I guess?



Source:
Kubernetes Operators by
Jason Dobies and Joshua Wood (O'Reilly). Copyright 2020 Red Hat, Inc.,
978-1-492-04805-3.

Great, we know the lay of the land now! Well, but what *are* Operators, really?

Controllers

The stars of this show

Controllers are pieces of software that live in the Control Plane and run on pods (just like most things that comprise Kubernetes):

- Manage one or more resources
- Make sure the current state matches the desired state
- Are notified whenever there is a change to resources it controls
- The act of changing the current state to match the desired state is called “reconciling”

Resources

The building blocks of the Kubernetes API

Resources are the actual mechanisms exposed by the API to control all the moving parts. Examples:

- Pods
- Nodes
- Services
- Service Accounts

Custom Resources (CR)

The building blocks for extending the Kubernetes API

Custom Resources are resources which are not really part of the core Kubernetes API:

- They are extensions of it, created so that users can teach Kubernetes new tricks
- Defined by Custom Resource Definitions (CRD)
- The way to tell a controller the desired state of your cluster when the core API is not enough

Operators

Wrapping it all up

Our software SREs:

- Have one or more controllers, who do the actual heavy lifting for you
- Keep an eye out for certain CRs that dictate the desired state of the application
- Reconciles the current and desired states by creating, updating or deleting resources and CRs
- Perform administrative tasks such as:
  - Performing backups
  - Configuring the application
  - Updating the application
- In other words, make sure everything is perfect without a single human intervention

CRD example

A CRD is metadata that tells Kubernetes how to handle your CR

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: nexus.apps.m88i.io
spec:
  group: apps.m88i.io
  names:
    kind: Nexus
    listKind: NexusList
    plural: nexus
    singular: nexus
  scope: Namespaced
  subresources:
    status: {}
  version: v1alpha1
  versions:
  - name: v1alpha1
    served: true
    storage: true
```

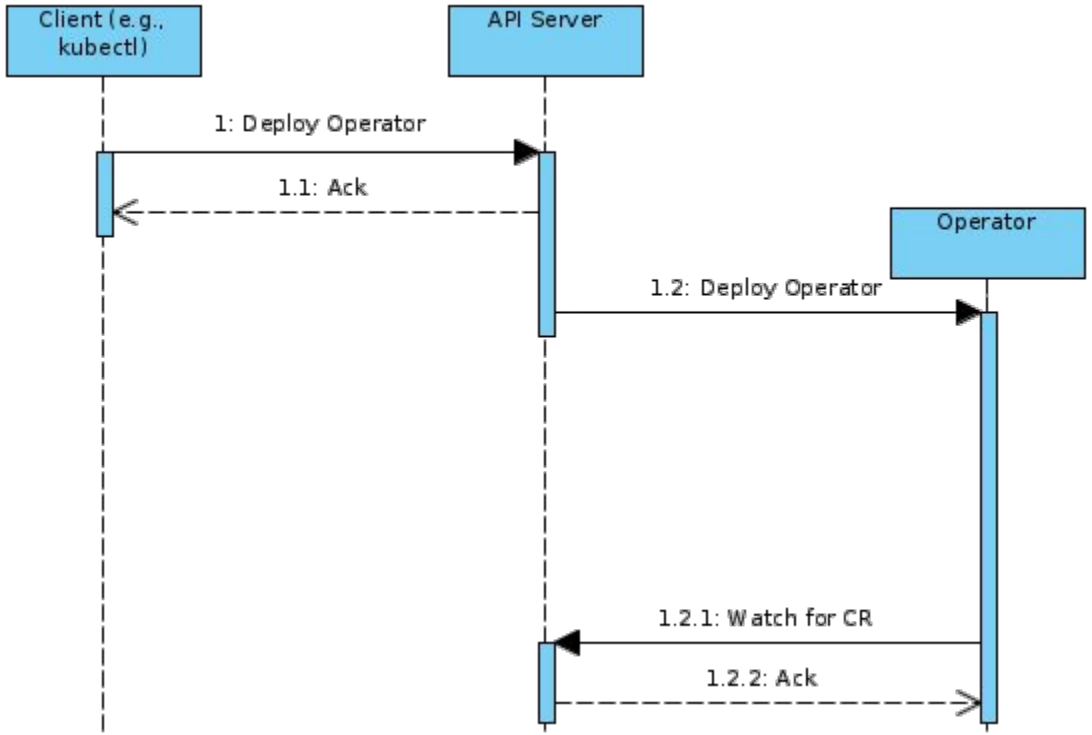
CR example

A CR defines the desired state in a way your Operator's controller(s) understands

```
apiVersion: apps.m88i.io/v1alpha1
kind: Nexus
metadata:
  name: nexus3
spec:
  replicas: 1
  persistence:
    persistent: true
  useRedHatImage: false
```

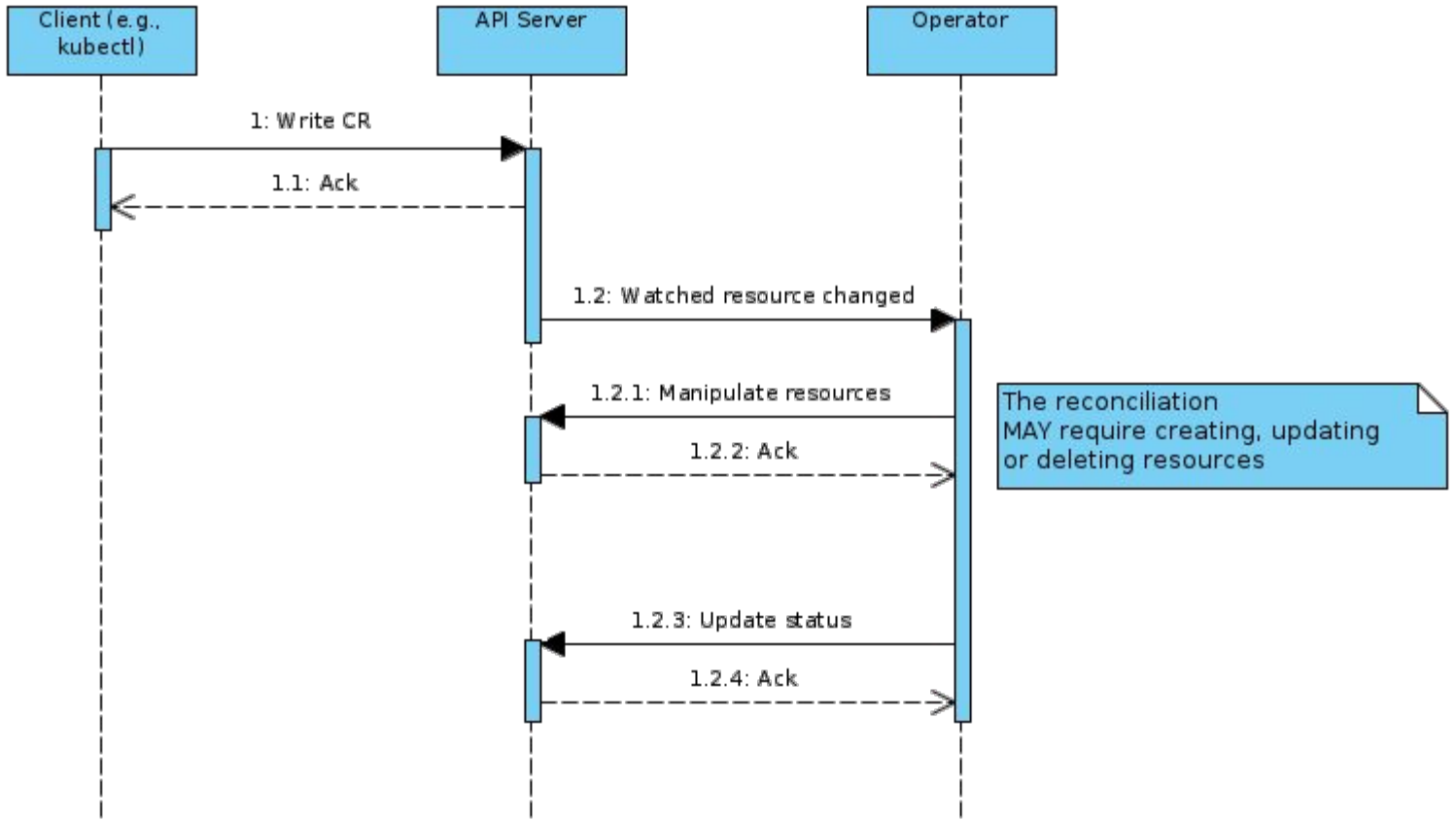
Communication flow

When you deploy an Operator



Communication flow

When you create/modify a CR managed by an Operator



Sounds good! But do I have to code one myself?

Operator Hub

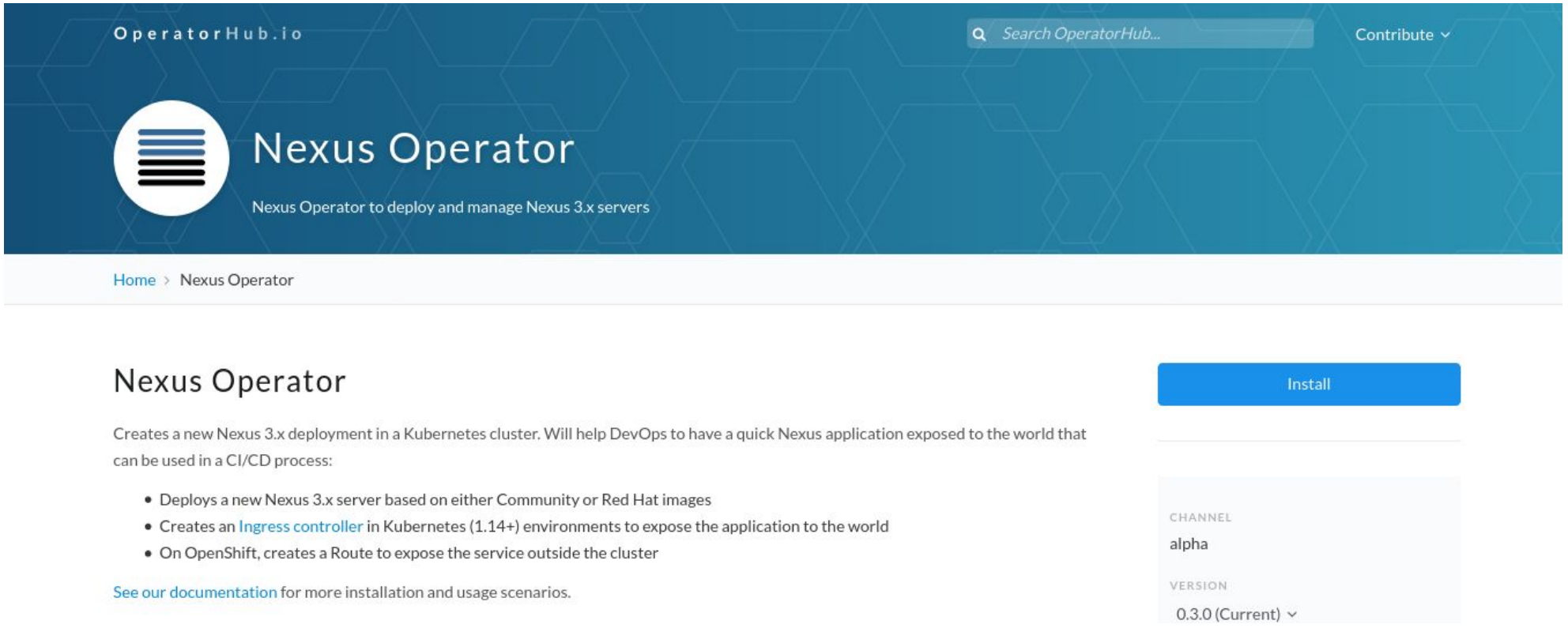
Home of Operators developed by communities and vendors

The Operator Hub offers an easy way to:

- Package
- Publicize
- Discover
- Pull

Operator Hub

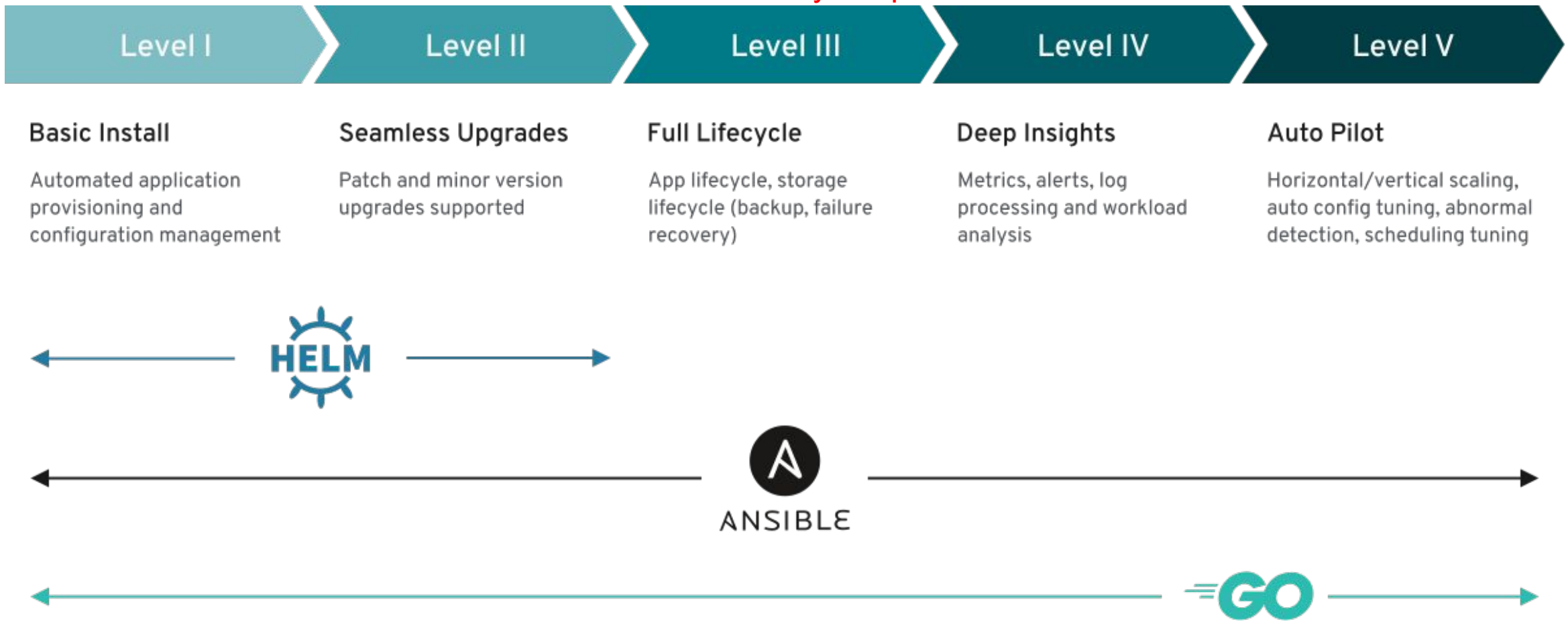
Home of Operators developed by communities and vendors



Source:
https://operatorhub.io/operator/nexus-operator-m88i

Capability Levels

How to measure maturity of operators?



Source:
https://operatorhub.io/static/images/capability-level-diagram.svg

Operator Lifecycle Manager

The OLM helps users install, update, and manage the lifecycle of all Operators

Part of the Operator Framework project, it provides:

- Over-the-Air Updates and Catalogs
- Dependency Model
- Discoverability
- Cluster Stability
- Declarative UI controls

Source:
https://github.com/operator-framework/operator-lifecycle-manager

Writing an Operator

Couldn’t find what you needed?

There are a few options for you:

- Go SDK
- Ansible SDK
- Helm SDK
- Anything that can act as a client for Kubernetes API (HTTP)
  - For example, Strimzi (Kafka Operator) is written entirely in Java

Read more at:
https://operatorhub.io/getting-started

What about my hypothetical cluster? Is there a Nexus Operator?

Sonatype NXRM 3 Certified Operator

https://github.com/sonatype/operator-nxrm3

- Maintained by Sonatype
- Uses the Helm SDK
- Requires self-hosting (not available at Operator Hub)
- Capability level unclear, but can only reach level II with the Helm SDK

M88i Nexus Operator

https://github.com/m88i/nexus-operator

- Maintained by 3 Red Hat employees*
- Uses the Go SDK
- Available at Operator Hub
- Capability Level II (seamless upgrades)
- Ongoing plans for Level III (full lifecycle)
  - General deployment configuration ✓
  - Automatic updates within same minor ✓
  - Automatic creation of repositories ✓
  - Management of backup activity
  - Management of HA clusters

*: The project is not maintained or endorsed by Red Hat in any way; the project is a fruit of individual contributions performed by its authors

Demo Time!

Installing OLM

```
└─ curl -sL
https://github.com/operator-framework/operator-lifecycle-manager/releases/download/0.16.1/install.sh | bash -s 0.16.1

customresourcedefinition.apiextensions.k8s.io/catalogsources.operators.coreos.com created

customresourcedefinition.apiextensions.k8s.io/clusterserviceversions.operators.coreos.com created

# (output omitted)

Package server phase: Installing

Package server phase: Succeeded

deployment "packageserver" successfully rolled out
```

Installing OLM

```
└─ kubectl get namespace
NAME              STATUS   AGE
default           Active   12m
kube-node-lease   Active   12m
kube-public       Active   12m
kube-system       Active   12m
olm               Active   54s
operators         Active   54s
```

Installing the Operator via OLM

```
└─ kubectl create -f https://operatorhub.io/install/nexus-operator-m88i.yaml
subscription.operators.coreos.com/my-nexus-operator-m88i created
└─ kubectl get csv -n operators -w
NAME                    DISPLAY          VERSION   REPLACES              PHASE
nexus-operator.v0.3.0   Nexus Operator   0.3.0   nexus-operator.v0.2.1
nexus-operator.v0.3.0   Nexus Operator   0.3.0   nexus-operator.v0.2.1   Pending
nexus-operator.v0.3.0   Nexus Operator   0.3.0   nexus-operator.v0.2.1   InstallReady
nexus-operator.v0.3.0   Nexus Operator   0.3.0   nexus-operator.v0.2.1   Installing
nexus-operator.v0.3.0   Nexus Operator   0.3.0   nexus-operator.v0.2.1   Succeeded
^C%
```

Installing the Operator via OLM

```
└─ kubectl get all -n operators
NAME                                    READY   STATUS   RESTARTS   AGE
pod/nexus-operator-7ff5b8588c-7xpdx   1/1   Running   0          2m5s
NAME                                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)             AGE
service/nexus-operator-metrics   ClusterIP   10.99.157.28   <none>      8383/TCP,8686/TCP
95s
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nexus-operator   1/1    1            1           2m5s
NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/nexus-operator-7ff5b8588c   1       1       1       2m5s
```

Deploying a Nexus instance

```
└─ kubectl create namespace nexus-demo
namespace/nexus-demo created

└─ echo "apiVersion: apps.m88i.io/v1alpha1
kind: Nexus
metadata:
  name: nexus3
spec:
  replicas: 1
  persistence:
    persistent: true
  useRedHatImage: false" | kubectl -n operators apply -f -
nexus.apps.m88i.io/nexus3 created
```

Installing the Operator via OLM

```
└─  kubectl -n operators get deployment
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
nexus-operator   1/1     1            1           7h38m
nexus3           0/1     1            0           12s
└─  kubectl -n operators get svc
NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)             AGE
nexus-operator-metrics   ClusterIP   10.99.157.28   <none>        8383/TCP,8686/TCP   7h38m
nexus3                   ClusterIP   10.98.65.215   <none>        8081/TCP            29s
```

Deploying a Nexus instance

```
└─  kubectl -n operators get pods -w
NAME                                  READY   STATUS RESTARTS
AGE
nexus-operator-7ff5b8588c-7xpdx   1/1   Running   2
7h38m
nexus3-5c7797464-z85s5        0/1     ContainerCreating   0
    5s
nexus3-5c7797464-z85s5        0/1     Running             0
    7s
nexus3-5c7797464-z85s5   1/1  Running   0       4m52s
^C%
```

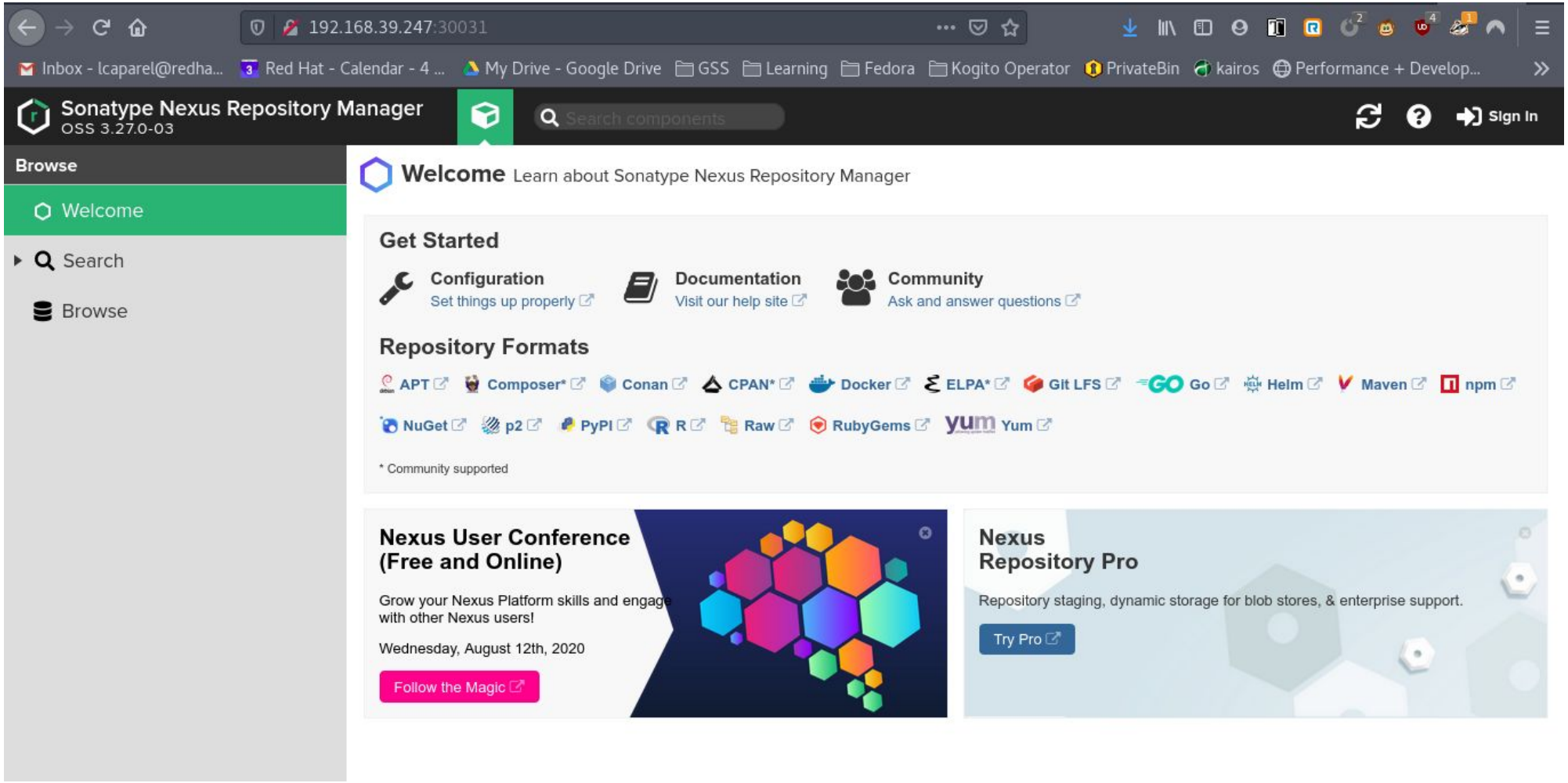
Editing an existing instance

```
└─ kubectl -n operators edit nexus/nexus3
# (output omitted)
spec:
  # (output omitted)
  networking:
    expose: true
    exposeAs: NodePort
    nodePort: 30031
  # (output omitted)
```

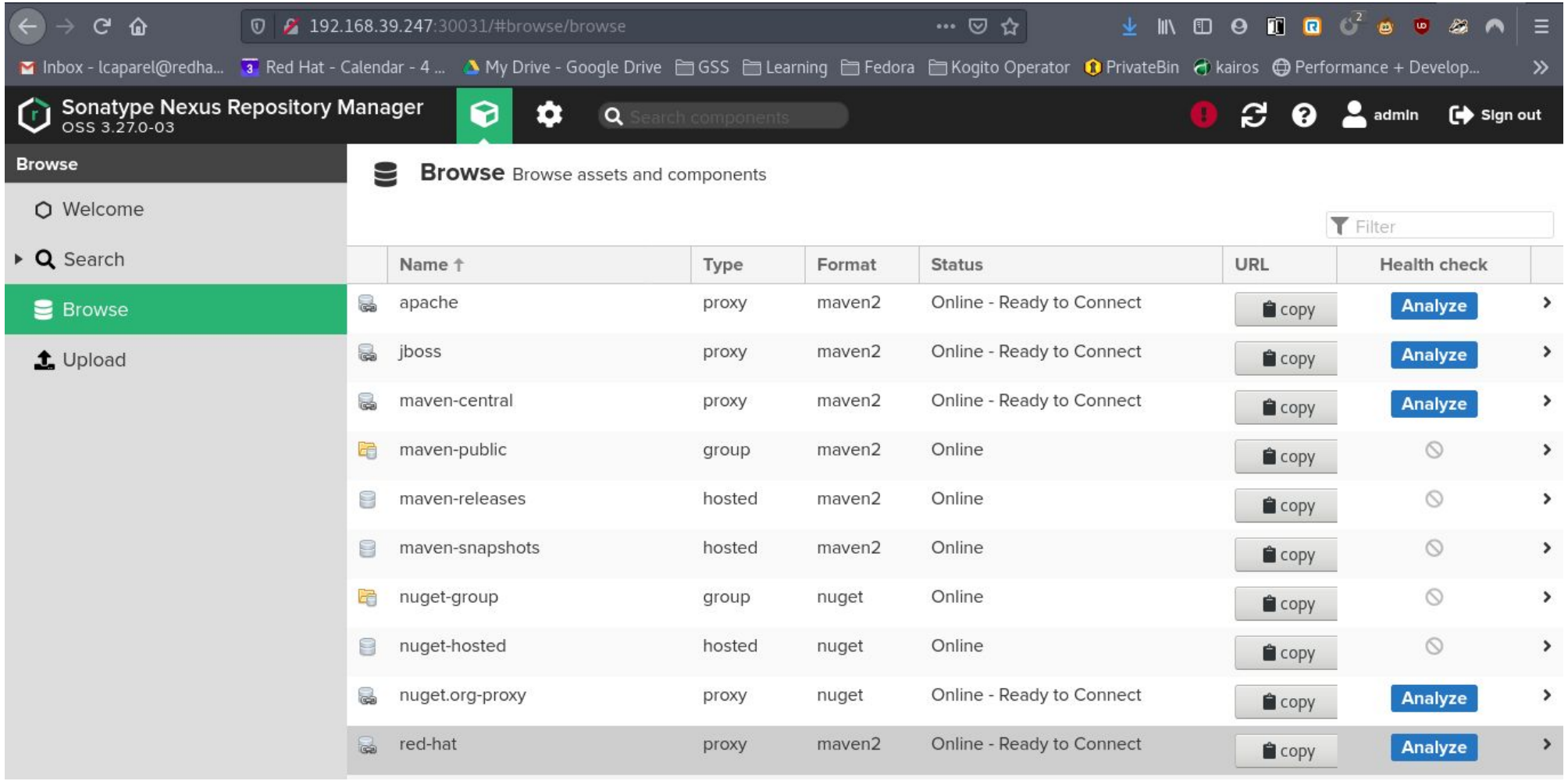
Checking out the effect of our changes

```
└─ kubectl -n operators describe service/nexus3 | grep
NodePort

Type:                     NodePort

NodePort:                 http  30031/TCP

└─ curl -i $(minikube ip):30031/service/rest/v1/status

HTTP/1.1 200 OK

Date: Wed, 16 Sep 2020 19:04:41 GMT

Server: Nexus/3.27.0-03 (OSS)

X-Content-Type-Options: nosniff

Content-Length: 0
```

Checking out the effect of our changes



Automatically created repos



What’s next?

Thank you