Kubernetes on bare metal

Improving high-availability

Ondrej Mular
Software engineer
omular@redhat.com

Brno, Feb 2020
Something about me

- PCS developer
- improving HA capabilities of Kubernetes/OpenShift bare metal clusters
  - working together with Andrew Beekhof
  - identifying differences between cloud and bare metal deployments
  - fencing
  - maintenance mode
Kubernetes
Disclaimer: This is not intended to be detailed user guide for Kubernetes, but more of brief introduction to the principles used.
What is Kubernetes

a.k.a. k8s

- Container-orchestration system
- Automation of container operations
  - deployment
  - scaling
  - management
Under the hood

Kubernetes Control Plane
- kube-controller manager
- cloud-controller manager
- kube-api-server
- kube-scheduler
- etcd

Cloud

Kubernetes Nodes
- kubelet
- kube-proxy
Kubernetes Objects

An object is a data structure (usually seen as YAML) stored inside `kube-api-server`.

Desired state of an object is described under the `spec` field.

Current status of the Object can be found under `status` field.

```yaml
apiVersion: v1
class: Pod
metadata:
  name: myapp-pod
labels:
  - app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/
Pod

A building block

A **Pod** is basic execution unit. It encapsulates one or more containers.

All containers in a pod share an IP address, IPC, hostname, and other resources.

A **Pod** represents processes running on a cluster.

Basic schedulable unit. **Kube-scheduler** assigns a **Node** to a **Pod** to run on.

You should almost never create a **Pod** by yourself, but instead use higher level resource types such as **Deployment**, **ReplicaSet**, **Service**, ...
Extending API

Custom resource definition

It is possible to extend Kubernetes API by creating object of type **CustomResourceDefinition** (CRD). This allows to store objects of a custom type on API server. This alone does nothing, except for storing the object in Kubernetes.
Controller

Interacts with cluster via API server.

Responsible for at least one resource type.

The controller for that resource is responsible for making the current state (status field) come closer to that desired state (spec field).

Note: Controller is not an Operator. See: https://www.openshift.com/learn/topics/operators
Kubernetes cluster has no build-in node management. Everything has to be managed externally.
Cluster API
The Cluster API is a Kubernetes project to bring declarative, Kubernetes-style APIs to cluster creation, configuration, and management. It provides optional, additive functionality on top of core Kubernetes.

Cluster API readme

https://github.com/kubernetes-sigs/cluster-api
Cluster API is not part of Kubernetes project. **Cluster Lifecycle SIG** is responsible for the project.

The Kubernetes cluster is itself an object that can be managed just like any other Kubernetes object. Similarly, machines that make up the cluster are also treated as a Kubernetes resource **Machine**.

The API has two pieces - the core API, and a (cloud) provider implementation such as:

- AWS
- Google cloud
- ...

What does it mean?
**Node vs Machine objects**

**Node**

- Kubernetes core resource
- created on `kubelet` startup
- Pod is scheduled to run on a Node

**Machine**

- Custom resource by Cluster API
- created by user/cluster
- declarative spec for a Node
- on creation
  - a provider-specific controller will handle provisioning and installation of a new host
  - the host is then registered as a new Node matching the Machine spec
This works nicely for the cloud. What about bare-metal? Bare-metal provider for cluster API should be enough, right?
Metal$^3$
The Metal³ project (pronounced: Metal Kubed) exists to provide components that allow you to do bare metal host management for Kubernetes. Metal³ works as a Kubernetes application, meaning it runs on Kubernetes and is managed through Kubernetes interfaces.

Metal³ project documentation

https://metal3.io/documentation.html
What it provides

▸ Bare-metal provider implementation for cluster API
  • build on top of Ironic
    • running as a cluster service
▸ Introduces new CRD BareMetalHost
  • inventory
  • configuration for remote management
Host, Machine and Node?

3 abstraction layers

- **BareMetalHost**: Metal³
- **Machine**: Cluster API
- **Node**: Kubernetes
Cloud vs. bare-metal

Cloud
- creating a new machine is cheap
- no reboot method
- immutable machine configuration
- storage is attached externally
- no need to deal with HW issues

Bare metal
- impossible to 'create' a new machine
- reboot is useful
- mutable configuration
- storage is part of a cluster
- need to deal with HW defects
High-availability on bare-metal
Health-checking

Monitoring health of Machines using Kubernetes node conditions.

Initiating appropriate remediation method

- reboot (power fencing)
- deprovisioning

https://github.com/openshift/machine-api-operator
High-availability on bare-metal

Power fencing

Challenge

Trying to avoid reprovisioning as it is slow and may cause even more faults during underlying storage solution resync.

No need to reprovision a machine every time. Reboot might solve the issue.

No reboot method in cluster API.

Solution

MachineRemediation CR and Bare Metal Remediator controller (which is using cluster API) will try to just reboot a node.

Simulating reboot with power cycling (power off, power on) via BHM object

Machine Disruption Budget - for limiting fencing floods

https://github.com/kubevirt/machine-remediation
### Maintenance mode

#### Challenges

- SW upgrades
- HW fix
- No workload running on a node

#### Solution

- Server-side drain and cordon implementation
- Node Maintenance Operator and `NodeMaintenance` CR. Possible to put a Node in to maintenance mode via Kubernetes API

---

[GitHub Repository](https://github.com/kubevirt/node-maintenance-operator)
High-availability on bare-metal

Future plans

▸ get implementation of
  · health-checking to upstream OpenShift
  · reboot remediation method into Metal

▸ find right place for server-side drain and cordon implementation
  · trying to get it accepted as a KEP
Questions?
Thank you