What's New in Kubernetes 1.12

Presenters

Stephen Augustus

Product Management Chair

Pengfei Ni

SIG-Azure

Juan Vallejo

SIG-CLI

Kaitlyn Barnard

1.12 Communications Lead

Agenda

1.12 Features Overview

Azure VMSS

Kubectl Plugins

Q&A

Overview of 1.12 Features

Stephen Augustus

Kubernetes Product Management Chair

TLS (Transport Layer Security)

TLS Bootstrapping moves to GA!

TLS Server Certificate Rotation is now beta.

Scheduling

Quota by priority - beta

Taint nodes by priority - beta

API Machinery

Resource Quota API - beta (defaulting quotas for high-cost resources)

API Server Dry-run - alpha

Network

NetworkPolicy

- Egress - GA
- ipBlock - GA

Node

RuntimeClass - alpha (cluster-scoped runtime properties)

Pod Process Namespace sharing - beta

Kubelet Device Plugin registration - beta

Storage

Topology-aware dynamic provisioning - beta

Dynamic maximum volume count - beta

Snapshot / restore via CRD (Custom Resource Definition) - alpha

Autoscaling

HPA (Horizontal Pod Autoscaler)

- Scaling via custom metrics (metrics-server) - beta
- Improving scaling algorithm to reach size faster - beta

VPA (Vertical Pod Autoscaler) - beta

Azure Virtual Machine Scale Sets (VMSS) and Cluster-Autoscaler

Pengfei Ni, Microsoft Azure

Azure Virtual Machine Scale Set (VMSS)

- Easy to create and manage identical VMs
- Provides high availability and application resiliency
- Allows your application to automatically scale as resource demand changes
- Provides large-scale VM instances (1000)

VMSS vs VM

Scenario	Manual group of VMs (VMAS)	Virtual machine scale set
Add additional VM instances	Manual process to create, configure, and ensure compliance	Automatically create from central configuration
Traffic balancing and distribution	Manual process to create and configure Azure load balancer or Application Gateway	Can automatically create and integrate with Azure load balancer or Application Gateway
High availability and redundancy	Manually create Availability Set or distribute and track VMs across Availability Zones	Automatic distribution of VM instances across Availability Zones or Availability Sets
Scaling of VMs	Manual monitoring and Azure Automation	Autoscale based on host metrics, in-guest metrics, Application Insights, or schedule

Current Status

- Currently the following is supported:
  - VMSS master nodes and worker nodes
  - VMSS on worker nodes and Availability set on master nodes combination.
  - Per vm disk attach
  - Per VM instance public IP
  - Azure Disk & Azure File support
  - Availability zones (Alpha)

- In future there will be support for the following:
  - AKS with VMSS support

Cluster Autoscaler on Azure

- Automatically adjusts the size of the Kubernetes cluster
- Four VM types are supported on Azure
  - Azure Kubernetes Service (AKS)
  - Virtual Machine Scale Set (VMSS)
  - Standard Virtual Machine (VMAS)
  - Azure Container Service (ACS)
- In the future, Cluster Autoscaler will be integrated within AKS product, so that users can enable it by one-click.

Kubectl Plugins

Juan Vallejo, Red Hat

Kubectl Plugins

- Plugins allow users to extend the core functionality of **kubectl**
- Enabled users to customize existing behavior to suit unique edge cases
- Allow the community to maintain and evolve extended functionality at its own pace
- Reworked in the **1.12 release** with ease-of-use and future improvement in mind

Kubectl Plugins

- Plugins were re-designed with three main criteria in mind:
  1. Plugins should be easy to write
  2. Plugins should be easy to install
  3. Plugins should be extendable

Kubectl Plugins

1. Plugins should be easy to write:
    - Can be written in any language
    - No framework or libraries required
    - All flags and arguments, specified by a user, are passed as-is to a plugin process
        - Plugins choose how to handle user input and parameters
    - We have made **kubernetes/cli-runtime** available in order to assist with plugin development
        - Same set of helpers, printers, and configuration-handling utilities in use by **kubectl**
    - Plugin template in Go available at **kubernetes/sample-cli-plugin**

Kubectl Plugins

2. Plugins should be easy to install
    a. No plugin-specific PATH env var or fixed location
        i. Plugins live anywhere on a user's PATH
    b. No configuration or pre-loading of plugin commands
        i. Plugins determine which command-path they implement based on their filename
    c. No external tool or program required in order to use a plugin with **kubectl**

Kubectl Plugins

3. Plugins should be extendable
    a. Easy to build additional functionality on top of the new plugin mechanism
        i. Plugin mechanism provides basic functionality in order to install + use plugins, as minimally as possible
        ii. Plugins themselves should be used to further extend the usability and functionality of plugin mechanism
            1. KREW: Plugin lifecycle and package manager, built on top of the plugin mechanism
    b. Plugins are able to "extend" other plugins
        i. Plugins can leverage naming scheme to "add" themselves as subcommands to other plugins (e.g. *kubectl-existing_plugin-foo -> kubectl existing-plugin foo*)

Naming a Plugin

- A plugin is an executable file, whose filename has the **kubectl-** prefix, and lives anywhere on a user's PATH
- Filename begins with: **kubectl-**
  - **/usr/local/bin/kubectl-whoami** provides the command **kubectl whoami**
  - Commands with dashes ("-") *use an underscore* ("_") in their plugin filename:
    - */usr/bin/kubectl-***get_pods** provides the command **kubectl get-pods**
    - Underscores act as both "-" and "_" when invoked: **kubectl get_pods** also works

Discovering Plugins

- **$ kubectl plugin list**
  - Discover all plugins available to **kubectl**
  - Plugins are listed and executed in the same order found in a user's PATH
  - Obtain warnings and errors preventing a plugin from being executed
    - Plugin **/a/b/kubectl-foo** overshadows **/c/d/kubectl-foo**
    - Plugin follows naming scheme, but is not executable
    - Plugin attempts to overwrite a core kubectl command
- Plugin executable preference:
  - **$ ls /usr/local/bin**
    - kubectl-foo-bar
    - kubectl-foo-bar-baz
  - **$ kubectl foo bar baz**
    - Longest available filename is always preferred (kubectl-foo-bar-baz)

Plugin Limitations

- Although plugins can "extend" other plugins, currently not possible to "extend" core **kubectl** commands (e.g. kubectl-version-foo)
- No additional information passed from **kubectl** process to **plugin** process
  - No plugin-specific environment variables (such as **KUBECTL_PLUGIN_CURRENT_NAMESPACE**)
    - Want to reduce complexity from plugins implementation
    - Avoid constant **kubectl** patches in order to offer more information to plugins to cover edge cases
- Plugins must parse and validate all flags and arguments passed to them
  - Although **kubectl** will not do this for you (see explanation above), we do provide **kubernetes/cli-runtime** to do this for plugins written in **Go**.
- Pre-1.12 plugins must be migrated in order to work with newer versions

Future of Plugins

- Although plugins cannot currently extend core **kubectl** commands:
  - Discussion underway to potentially add limited support in the future
    - Do users want this ability?
    - Which command paths should be "extendable"?
    - Should plugins be allowed to completely override **kubectl** commands, or just add sub-commands?
- Your feedback is welcome!

Questions?

CLOUD NATIVE
COMPUTING FOUNDATION

Thank You

CLOUD NATIVE
COMPUTING FOUNDATION