ProTego

The ProTego Integration Toolkit – A Kubernetes Journey

Whitepaper

Noel Thomas and Arturo Arriaga, ICE

INTRODUCTION

Containers have quickly become the standard for application deployment. Containers are units of software that package up code and all dependencies so that an application is able to run quickly and reliably in various computing environments, from OnPremise to OnCloud deployments.

With containers comes the need for managing them, and to this end Kubernetes has emerged as the leading container orchestration software. Kubernetes is a portable, extensible, open-source platform for managing containerized workloads. It facilitates declarative configuration and automation, takes care of scaling and failing over applications, provides deployment patterns, and more.

In addition, current development trends rely strongly on microservice architectures and cloud deployments. Dev teams must use microservices - mainly using containers - to architect for portability, whilst Ops teams must manage large hybrid and multi-cloud deployments. Kubernetes plays a key role in supporting these microservice/cloud native architecture trends.

We have developed the ProTego Integration Toolkit in order to support the Kubernetes journey. This toolkit consists of a bundle of several open-source technologies based on the Kubernetes ecosystem to integrate and manage the ProTego components. In addition, a user-friendly installer has been developed, which automates the installation of this Integration Toolkit.

INTEGRATION TOOLKIT

The Integration Toolkit is based mainly on Kubernetes and Docker technologies and provides the key infrastructure for deployment and integration of applications. It also includes several other Kubernetes related tools, deployed along with the toolkit for ease of management, like Rancher, Helm, etc. The core components of the Integration Toolkit infrastructure are as follows.

- **Kubernetes**: leading technology in container orchestration, that manages the deployment and integration of containers. It is used as the base platform where applications are deployed.
- **Docker:** leading container technology. Containers allow for applications to be run independent of the operating system environment. It allows for separate environments for each application.
- **Rancher:** software that allows to deploy and manage Kubernetes clusters in a more user-friendly way both on premise and on cloud. Applications, i.e., ProTego Toolkit components, are deployed via Rancher/Helm charts to the Kubernetes clusters.
- **RKE (Rancher):** RKE is a CNCF-certified Kubernetes distribution that runs entirely within Docker containers. It solves the common frustration of installation complexity with Kubernetes by removing most host dependencies and presenting a stable path for deployment, upgrades, and rollbacks.
- **K3s (Rancher):** the lightweight Kubernetes, is a fully compliant Kubernetes distribution, easy to install, half the memory, all in a binary of less than 100 MB among other enhancements.
- **Helm:** the package manager for Kubernetes, applications are packaged in the form of Helm charts and deployed as a unit. By using Kubernetes yaml templates, allows multiple containers to be grouped together as well as describing the properties needed to deploy an application.
- **Istio:** service mesh technology that allows to add transparently a layer to provide the platform with enhanced connectivity, security, control and observability. It uses a sidecar container deployed along the application container to provide the service mesh features.
- **Ansible:** software for IT automation and provisioning, used to deploy the ProTego platform and its components.

© 2021 ICE and other members of the ProTego consortium

- **Longhorn:** highly available persistent storage for Kubernetes, which uses the local filesystem of the nodes and provides replication and backups. It is deployed on a Kubernetes cluster as a regular application using a Helm chart.
- **NFS:** Network File System server which can be deployed along with the Toolkit in order to provide shared storage for applications.
- **GitLab:** a platform for software development and version control based on Git, used as Container Registry and Helm Chart catalog.
- **Jenkins:** an automation software to build, test and deploy applications, it is the key CI/CD tool for the toolkit to automatically deploy applications and provide new versions.

All these provide the base platform for the deployment and integration of the ProTego Toolkit components in an automated and repeatable way.

INTEGRATION TOOLKIT ARCHITECTURE

The Integration Toolkit is composed of two different Kubernetes clusters.

- The K3s Cluster (Rancher cluster): a single node Kubernetes cluster where Rancher is deployed.
- The RKE Cluster (Application cluster): the cluster where the ProTego components are deployed. This cluster has a master and several worker nodes.

Figure 1 shows the basic Integration Toolkit Kubernetes Architecture, with a Rancher cluster in order to manage the Application cluster and the applications, and the Application cluster, with a master node where the Kubernetes control plane runs, and some worker nodes where applications are deployed.

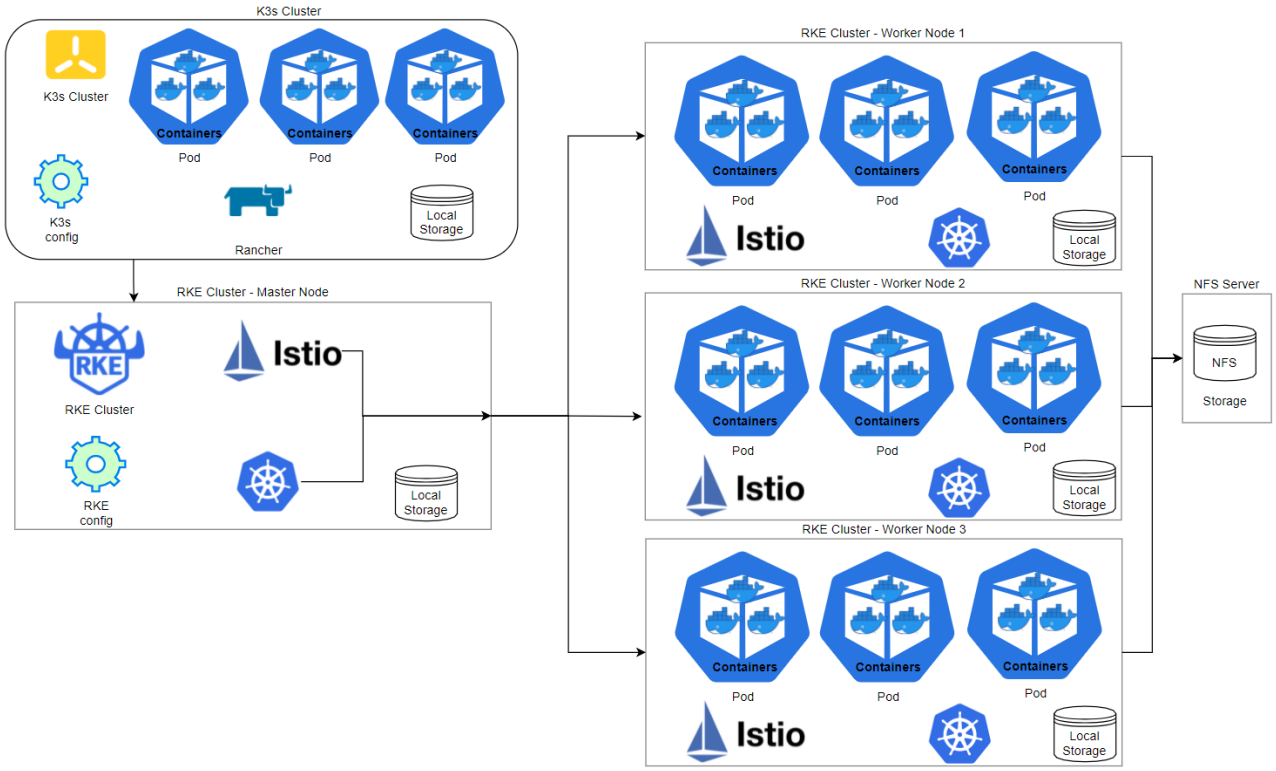

Figure 1 Integration Toolkit Architecture

INSTALLATION OVERVIEW

In order to install the Integration Toolkit, several Ansible playbooks are provided that allow for an automatic or semi-automatic deployment of the Toolkit.

Ansible is an IT automation engine that allows for automated provisioning and configuration management and other IT tasks. It uses no agents so it is easy to deploy, and uses playbooks expressed in yaml which allows the automation tasks to be described in a declarative way.

Ansible works by connecting to the nodes, where the Toolkit is going to be installed, and running small programs called “Ansible modules” as per Ansible playbook definitions. Ansible modules execute tasks. One or more tasks can be combined to make a play. Two or more plays can be combined to create a playbook. Ansible playbooks are lists of tasks that automatically execute against specific hosts. Groups of hosts form the Ansible inventory.

Ansible uses a single node, where it is installed. It uses ssh authorized keys to connect to the nodes where the playbooks will be run. The Ansible node can be any machine, even a user laptop, provided ssh keys are established.

The basic steps for Integration Toolkit deployment, using the Ansible playbooks, are as follows.

- Check/Install pre-requirements (Ansible).
- Install Kubernetes Application Cluster [RKE].
- Install Kubernetes Rancher Cluster [K3s].
- Deploy Rancher to the Rancher Cluster.
- Register the Application Cluster in Rancher.
- Deploy Istio.
- Install NFS for shared storage (optional).
- Deploy Longhorn for HA storage (optional).
- Deploy the CI/CD Pipeline (optional).

KUBERNETES 101

Kubernetes is the leading technology in container orchestration. There are several distributions that can be used to install a Kubernetes cluster. The RKE distribution from Rancher has been selected for the Integration Toolkit and for the Application Cluster. RKE is a production-grade Kubernetes distribution which reduces installation complexity by removing most host dependencies and presents a stable path for deployment, upgrades and rollbacks given that it runs entirely within Docker container.

A Kubernetes cluster consists of the components which represent the control plane and a set of machines called worker nodes which run containerized applications. Every cluster has at least one worker node. The worker node(s) host the applications (in the form of Pods) and the control plan manages the worker nodes and Pods in the cluster.

Kubernetes uses Kubernetes objects, a set of persistent entities in the Kubernetes system, to represent the state of the cluster. Specifically, they can describe:

- what containerized applications are running,
- the resources available to those applications, and
- the policies around how those applications behave, i.e. restart, upgrades …

A Kubernetes object is a “record of intent,” and once created Kubernetes will constantly work to ensure that object exists, i.e. this is your cluster’s desired state.

© 2021 ICE and other members of the ProTego consortium

To work with Kubernetes objects, the Kubernetes API is used, via kubectl from the command-line interface or directly in custom programs using a client library. When an object is created, the object spec and some basic information has to be provided. Most often, all this information is provided in a .yaml file describing the object.

The key Kubernetes object or resources related to containerized applications is a Pod. A Pod is the smallest deployable unit of computing that can be created in Kubernetes. A Pod is a group of one or more containers with shared storage and network resources, and a specification for how to run the containers.

Some other key resources grouped by functionality are as follows.

Workloads

A workload is an application running on Kubernetes. Whether this workload is a single component or several that work together, on Kubernetes they run inside a set of Pods. Then, instead of managing each Pod directly, workload resources can be used to manage a set of pods on the user's behalf. Workloads configure controllers that make sure that the right number of the right kind of pods are running, to match the state specified. Some of the main workload resources are as follows.

- ReplicaSet: maintain a stable set of replica Pods running at any given time.
- Deployment: provides declarative updates for Pods and ReplicaSets.
- StatefulSet: similar to a Deployment but for stateful applications keeping a persistent identifier for each Pod that is maintained across any rescheduling.
- DaemonSet: ensures that all (or some) Nodes run a copy of a Pod.
- Others like Job, CronJob …

Networking

Containers withing a Pod use networking to communicate via loopback. Cluster networking provides communication between different Pods.

Services are an abstract way to expose an application running on a set of Pods as a network service. Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods, and can load-balance across them. The main service types are as follows.

- ClusterIP: exposes the service on a cluster internal ip, being reachable from within the cluster.
- NodePort: exposes the service on each Node's IP at a static port being reachable from outside the cluster via NodeIP:NodePort.

Ingress manages external access to the services in a cluster, typically HTTP, providing load balancing, SSL termination and name-based virtual hosting. An ingress controller, such as ingress-nginx, must be deployed to satisfy an Ingress resource but there are a number of other ingress controllers with multiple features.

Storage

On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem is the loss of files when a container crashes. The kubelet restarts the container but with a clean state. A second problem occurs when sharing files between containers running together in a Pod.

- PersistentVolumes: a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.
- PersistentVolumeClaim: a request for storage by a user, PVCs consume PV resources.

© 2021 ICE and other members of the ProTego consortium

Configuration

ConfigMap is used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables or configuration files in a volume.

Secrets store and manage sensitive information, such as passwords, ssh keys, etc.

There are many other resources that can be used. For a detailed description of them, see the official Kubernetes documentation at https://kubernetes.io/.

HELM AND RANCHER CHARTS

Helm is the package manager for Kubernetes. Helm helps to manage Kubernetes applications using Helm charts. Helm charts help to define, install and upgrade Kubernetes application. Charts describe even the most complex apps, provide repeatability, and serve as single point of authority. In addition, charts are easy to version, share and host on public or private servers.

A Helm chart is a collection of files that describe a related set of Kubernetes resources. A single chart might be used to deploy a simple app such as a single nginx pod or a more complex app such as a full web app stack with HTTP servers, databases, frontend, backend, and so on.

Charts are created as files in a particular directory tree and can be packaged into versioned archives to be deployed. The directory name is the name of the chart, without the versioning information.

A sample Helm chart structure for an app:

```
app/
  Chart.yaml: a yaml file containing information about the chart
  README.md (optional): a human-readable readme file
  values.yaml: the default configuration values for this chart
  charts/ : directory containing any charts upon which this chart depends
  templates/ : directory of templates, that is the templates of the
               Kubernetes resource files that, combined with values,
               generate valid Kubernetes manifest files
```

A Helm chart uses the files in the templates/ directory as templates of Kubernetes resources for the application. Then the values file is used to render valid Kubernetes resources from the templates during deployment time.

If using Rancher, Rancher charts - which are a superset of Helm charts – can be used to deploy applications to the Kubernetes cluster using Rancher.

Rancher charts are very similar to Helm charts, differing only in the directory structure, which includes an app version directory charts/app/version/ … and two additional files, as follows.

- app-readme.md: a text-based file which provides a high-level overview display in the Rancher UI.
- questions.yaml: matches any values.yaml variable that needs or requires to be displayed in the Rancher UI during deployment, so that a user can set specific values for specific variables during deployment time from the Rancher UI.

Helm or Rancher charts in Rancher are deployed using Catalogs. A Catalog is a Git or Helm repository filled with Helm Charts ready to be deployed.

© 2021 ICE and other members of the ProTego consortium

CONCLUSIONS

Current development trends rely strongly on microservices and cloud native architecture, and Kubernetes plays a key role for development and deployment. The ProTego Integration Toolkit supports end users with this transition to Kubernetes.

In this document, a brief overview of the Integration Toolkit has been presented, along with a description of its main components and architecture, a Kubernetes 101 summary, a description of Helm charts and how to use them, and an overview of the installation steps.

© 2021 ICE and other members of the ProTego consortium