Deploying DCAE service components using Helm (Work In Progress)

Background

DCAE *service components* are Docker containers that run data collection and data analysis software for monitoring the health and performance of xNFs deployed by ONAP. These components run in the ONAP Kubernetes environment. Currently (as of the Guilin release), these components are deployed using Cloudify. Each component has a Cloudify blueprint and a set of input parameters. The DCAE Cloudify Kubernetes plugin (dcaegen2/platform/plugins/k8s, also known as k8splugin) processes a blueprint and the associated inputs and uses the Kubernetes API to create the Kubernetes resources (Deployments, Services, etc.) that make up a running instance of a service component.

For a variety of reasons, we would like to move away from using Cloudify in ONAP. The standard mechanism for deploying components in ONAP is Helm, and we would like to use Helm to deploy DCAE service components. At a high level, this is straightforward: instead of using a blueprint, a set of inputs, and Cloudify to create the needed Kubernetes resources, we use a Helm chart with a `values.yaml` file containing input parameters to create a similar collection of Kubernetes resources. There are a few complications, however:

- DCAE service components get their configurations from a Consul key-value store. The DCAE Cloudify Kubernetes plugin knows how to extract configuration information from a Cloudify blueprint and store it into Consul. Over time, DCAE service components may move away from using Consul for configuration, but in the near and medium term, we will need to continue to support Consul-based configuration when we move to Helm deployment of service components. (Service components don't access Consul directly; instead they request their configurations through a DCAE platform component called the config binding service.)
- Some DCAE service components use DMaaP data router feeds or authenticated DMaaP message router topics to exchange data with other components. Provisioning DMaaP feeds and authenticated topics requires interaction with the DMaaP bus controller. DMaaP provisioning information needs to added to the configuration of the service component. Currently we use a DCAE Cloudify DMaaP plugin to access the DMaaP bus controller provisioning API, and the DMaaP plugin knows how to place the resulting DMaaP configuration information into Consul so that the service component can access it.
- Some DCAE service components use the ONAP policy subsystem to manage parts of their configurations. There is a DCAE Cloudify policy plugin as well as DCAE policy handler platform service that interact with the policy subsystem to retrieve policy-based configuration information, detect changes to configuration initiated by the policy subsystem, and notify service components of changes. Again, this works through the Consul-based configuration mechanism.
- Some DCAE service components use a shared DCAE postgres database server. A DCAE Cloudify postgres-as-a-service plugin allows creating databases on the server and setting up access to it.

As we move to Helm-based deployment of DCAE service components, we will need to preserve the functionality we have today. In some cases, we will be able to implement the functionality using Helm and using Kubernetes features such as init containers, so that no changes will be needed to the service components. In other cases, we may need to change the service components themselves.

Basic capabilities

All service components, whether or not they use DMaaP, policy, or postgres, need a Helm chart that creates a Kubernetes Deployment and (if the service exposes any ports) a Service, similar to what the k8splugin would create. This section describes how we will use Helm to create the necessary Kubernetes resources for a service component and how we will set up the component's configuration in Consul.

Creating a Kubernetes Deployment for a service component

We need to create a Kubernetes Deployment resource for the service component that looks like the Deployment currently created by the k8splugin. The k8splugin deployment includes the following resources:

- A container that runs the service component itself, using a Docker image pulled from the ONAP Nexus repository.
- If the component is using the filebeat centralized logging mechanism, a sidecar container running filebeat, plus the volume and volume mounts needed for logging.
- If the component is using TLS with certificates from AAF, an initContainer that retrieves the certificate information, plus the volume and volume mounts needed for certificates.
- If the component is using TLS with certificates from an external CA, an initContainer that retrieves the certificate information.
- If the component is using both AAF certificates and certificates from an external CA, an initContainer that merges the Java truststores.

In addition to the resources that the k8splugin creates, we need:

- An initContainer that waits for the other components that the service component needs, such as AAF and Consul In the current Cloudify implementation, we can guarantee that a service component won't be deployed before these other components are ready. We can't make that guarantee for Helm.
- An initContainer that puts the component configuration into Consul (see next section for details).

The configuration parameters for the component (the image name, log directory, TLS parameters) currently come from the Cloudify blueprint–either directly or through an inputs file. With Helm deployment, these parameters will move into the `values.yaml` file for the component.

Because there is a common structure to the Kubernetes Deployments for all service components, we should create a Helm template for a DCAE service component Deployment. The Deployment will be configured for a specific service component through the parameters in the `values.yaml` file. The OOM common Helm templates do not include a template for a Deployment, but they do include templates for various parts of a deployment (for instance, a logging sidecar). The DCAE service component Deployment template should use OOM common templates wherever possible.

Setting up the component configuration in Consul

The k8splugin gets the application's initial configuration from the `application_config` property in the blueprint and uses the Consul API to push this into Consul, using the service component's name as the key. To accomplish the same thing with Helm, we add another init container to the Deployment. This init container runs a Docker image that can take a key name and a path to a file as an argument and push the content of the file to Consul. A container that does already exists (org.onap.dcaegen2.deployments.consul-loader-container). It's being used for some DCAE *platform* components that also get their configurations from Consul. (For an example, see the OOM chart for the DCAE deployment handler, in the OOM tree at oom/kubernetes /dcaegen2/components/dcae-deployment-handler.)

To support configuration via Consul for DCAE service components:

- We put the application configuration into the component's values.yaml file as a property.
- The Helm chart creates a Kubernetes configMap using the application configuration from `values.yaml`, converted to JSON, as the content of the configMap.
- The Helm chart adds an initContainer to the component's Deployment, with the consul-loader-container as the image to run with the configMap containing the application configuration mounted onto the container's file system.

When the Helm chart for the service component is deployed, the initContainer will run and will store the application configuration into Consul. When the service component starts, the configuration information will be available
in Consul.

Kubernetes Service

The k8splugin currently creates a ClusterIP service if a component exposes IP ports within the cluster and a NodePort service plus a ClusterIP service if the component exposes a port outside the cluster. In the latter case, the ClusterIP service is redundant–a NodePort service will also expose a component within the cluster. The OOM common templates include a template for a Service. We can use that template to create a Service for the component, with all of the details configured in the `values.yaml` file. The k8splugin does not currently allow for exposing a component via an Ingress. Supporting Ingresses should be investigated as part of the move to Helm.

Extended capabilities

Using DMaaP

Using policies

Using a Postgres database

ONAP already has a Helm-based mechanism for deploying instances of Postgres and setting up credentials for client access. We use this mechanism to create the DCAE postgres database already. The drawback to this approach is that it creates a separate instance of Postgres each time it is used. There has been work on setting up shared instances of databases in ONAP, but so far Postgres has not been included in those efforts.

As we move DCAE service component deployment to Helm, we should make use of the ONAP project-wide Helm mechanisms. If DCAE has certain special requirements, we should work to enhance the project-wide mechanisms.

Implementation Phases

Service components deployed at DCAE installation time

As of the Guilin release, we are deploying 6 service components at DCAE installation time, via a script that runs on the DCAE k8s-bootstrap-container:

1. tcagen2 (org.onap.dcaegen2.analytics.tca-gen2.dcae-analytics-tca-web)
2. ves-tls (org.onap.dcaegen2.collectors.ves.vescollector)
3. prh (org.onap.dcaegen2.services.prh.prh-app-server)
4. hv-ves (org.onap.dcaegen2.collectors.hv-ves.hv-collector-main)
5. holmes_rules (holmes/rule-management)
6. holmes_engine (holmes/engine-management)

The first four components do not use any extended capabilities (DMaaP, policies, Postgres database), so they can be moved to a Helm-based deployment without implementing solutions for the extended capabilities. The last two components use the DCAE postgres database. Moving these components will require some alternative solution for the database. The Holmes components are managed by the Holmes project, which is separate from the DCAE project. The Holmes project will need to address the issue of handling the Postgres database differently.

For ONAP R8 ("Honolulu"), we propose creating the Helm templates and charts needed to deploy the first four components on the list above.

Service components deployed on demand

Helm Chart Management

Currently nearly all of the Helm charts for ONAP, including the charts for the DCAE platform components, are stored in a single source tree under the OOM repository. Typically the charts are built and stored into a locally-running Helm repository server instance.   This works reasonably well for deploying the base ONAP platform in a single installation step.   Most DCAE service components are deployed and undeployed after the initial ONAP installation. The charts for DCAE service components should be managed separately from the OOM tree, with charts for released versions of service components being pushed into a public ONAP Helm repository.