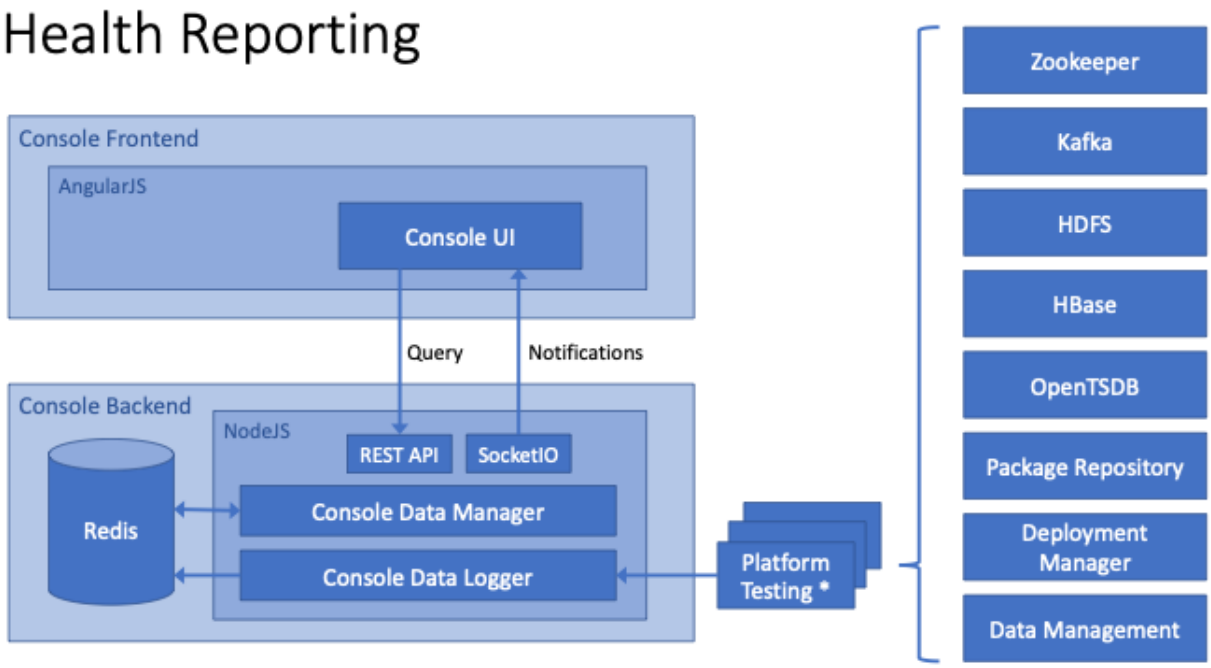
PNDA Platform Components

These are notes from a meeting to discuss the PNDA Platform Components and the development work required to migrate to Kubernetes.

PNDA Health Framework

The PNDA health framework is used to monitor component health and report it back to the PNDA Console. Each backend service has a platform-testing module that runs health tests on the service and reports it back to the PNDA console. The console itself has three main components:

- Console Backend Data Logger – provides an API for posting health messages to the console.
- Console Backend Data Manager – provides an API and a SocketIO interface to the console frontend for querying component metrics and health.
- Console Frontend – an AngularJS web UI served by Nginx.

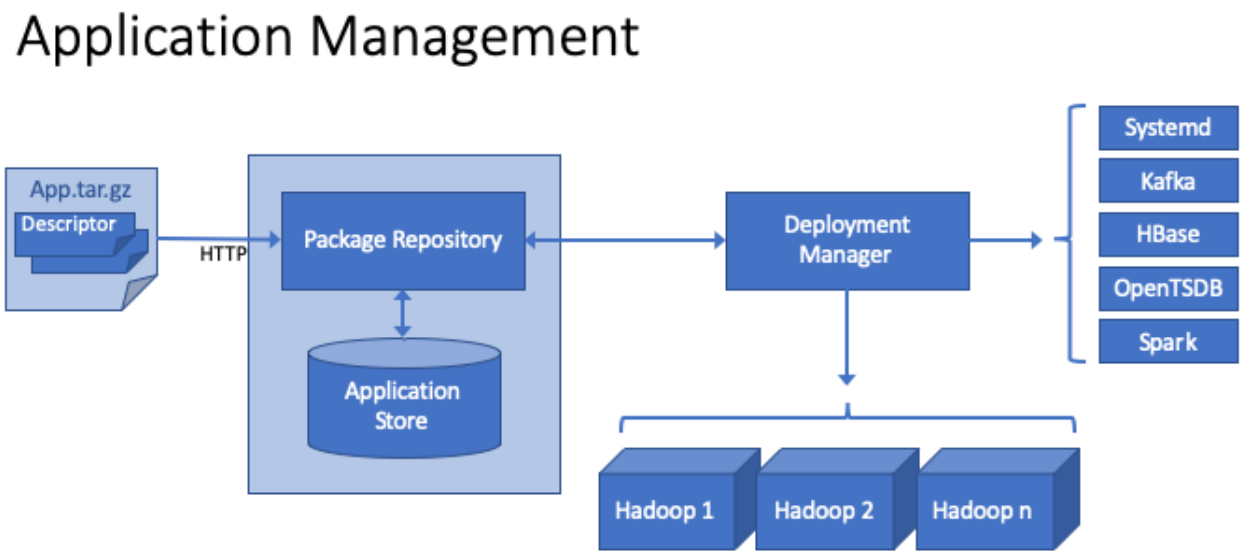


PNDA Applications

Applications are managed on PNDA by two main components:

- Package Repository – provides an API for uploading or deleting application packages from the system. The packages are stored in a Kubernetes persistent volume.
- Deployment Manager – provides an API for deploying applications from the package repository to the cluster.

Information about PNDA application management in PNDA 5.0 can be found here: http://pnda.io/pnda-guide/applications/



Accessing the Package Repository

It is possible to temporarily expose the package repository API with kubectl port forwarding:

```
kubectl -n pnda port-forward service/pnda-package-repository 8888
```

Then you can use curl to post a package to the package repository:

curl -XPUT "http://localhost:8888/packages/app-0.0.1.tar.gz?user.name=" --data-binary "@app-0.0.1.tar.gz"

Then you can verify the upload by fetching the list of packages:

```
curl "localhost:8888/packages?user.name="
[{"name": "app", "latest_versions": [{"version": "0.0.1", "file": "app-0.0.1.tar.gz"}]}]
```

Deployment Manager

The deployment manager is responsible for deploying apps into the PNDA cluster and configuring all components of the cluster, using descriptor files in the application .tar file. The deployment manager currently was quickly ported to Kubernetes, with sufficient changes to allow the PNDA console to work. There is not yet any functionality to deploy applications into Kubernetes.

A PNDA app is composed by one or more of the following component types. For each component, deployment-manager must be adapted to work with microservices in k8s instead of PNDA nodes (VMs/Baremetal) as before. Here are notes for transitioning each component type deployment:

oozie

An oozie application, consisting of a workflow and/or coordinator plus a set of supporting libraries and/or scripts (e.g. a Pig or Spark application). A coordinator runs periodically on a defined schedule, whereas a workflow must be manually run by using the start application API each time you want to run it. The job properties will automatically include all variables known to the deployment manager.

Oozie has not been integrated in cloud-native PNDA. Before integrating oozie, we should investigate more modern alternatives, mainly Apache Airflow https://airflow.apache.org/.

sparkStreaming

A spark streaming application. A text file named application.properties will be automatically appended with all variables known the the deployment manager and made available on the classpath.

PNDA deployment-manager creates a systemd service to manage the state of deployed sparkStreaming applications. To migrate to k8s, an option is to use a k8s Deployment to deploy sparkstreaming applicaiton in client-mode. spark-streaming application can be managed directly with the kubernetes API. For example can be stopped by scaling deployment to 0, started by scaling deployment to 1, and the status been the status of the POD.

flinkStreaming

A flink streaming application or a flink batch application. A text file named application.properties will be automatically appended with all variables known the the deployment manager and made available on the classpath.

Similar to sparkStreaming, flinkStreaming apps are managed with systemd.

jupyter

We have integrated jupyterhub in Cloud-Native PNDA. Jupyterhub allows multiple users to register and creates a single pod with a corresponding volume to store each user's notebooks. To make notebooks from an application available to users, deployment-manager copies (with scp) notebooks to the PNDA node running jupyter server. This process must be modified for jupyterhub on k8s. One option is to copy file to jupyter k8s volumes. However this can become meassy. zero-to-jupyterhub-k8s recommendation (https://github.com/jupyterhub/zero-to-jupyterhub-k8s/blob/master/doc/source/user-environment.rst) is to use nbgitpuller to synchronize a folder. nbgitpuller synchronize the master branch of a repository into a folder in the user volume. We can deploy a git server in k8s (maybe https://gogs.io/?) and make deployment-manager create a git repo and add PNDA application's notebooks to a folder in that repo. Then we configure jupyterhub to use nbgitpuller to synchronize the repo.

Development Tips For Helm and Kubernetes

When developing Helm charts, run your in-development chart separately from the main PNDA Helm chart. This will let you iterate through the install/delete cycle much faster.

Use https://www.telepresence.io/ to proxy into a Kubernetes cluster. This lets you access the cluster services as if your laptop is part of the cluster. It lets you run a component locally in a debugger or try out development changes without having to build a new image and push it to the registry.

Here is a cheat sheet for kubectl – https://kubernetes.io/docs/reference/kubectl/cheatsheet/