developer.skatelescope.org Documentation

Release 0.0.1

Marco Bartolini

Oct 25, 2021

CONTENTS

The charts used by the Helm deployer are maintained in this repository.

CHAPTER

ONE

CREATING OR UPDATING A CHART

1.1 Creating a new chart

1.2 Updating an existing chart

- Create a new branch and update the chosen chart as needed
- Test the chart, make sure it is backwards compatible. If not, dependencies are also updated.
- Create a merge request and get it approved and merged
- Create a new branch and update the version number of your chart in *Chart.yaml*
- Packaging your new chart:

```
cd ska-sdp-helmdeploy-charts
helm package charts/<updated_chart> -d chart-repo
helm repo index chart-repo
```

  Make sure you replace <*updated_chart*> with the actual chart name.
- The last command will update the timestamp on every chart in the repository. Revert these for the charts that you did not update.
- Update the chart *README.md*, and commit and push the changes
- Create a merge request, get it approved and then merge it.
- Your chart is now available to be accessed from the *chart-repo* directory

CHAPTER

TWO

DASK

Dask allows distributed computation in Python.

- https://dask.org
- https://jupyter.org/

2.1 Chart Details

This chart will deploy the following:

- 1 x Dask scheduler with port 8786 (scheduler) and 8787 (Web UI) exposed on a ClusterIP (default)
- 2 x Dask workers that connect to the scheduler
- 1 x Jupyter lab notebook (optional, false by default) with port 8888 exposed on a ClusterIP (default)
- All using Kubernetes Deployments

Note: only version 0.2.0 of the chart contains the Jupyter lab notebook pod.

> **Tip**: See the Kubernetes Service Type Docs for the differences between ClusterIP, NodePort, and LoadBalancer.

2.2 Installing the Chart

First we need to add the helmdeployer-chars repo to our local helm config.

```
helm repo add ska-sdp-helm https://gitlab.com/ska-telescope/sdp/ska-sdp-helmdeploy-
↪charts/-/raw/master/chart-repo
helm repo update
```

To install the dask chart with the release name `test`:

```
helm install test ska-sdp-helm/dask
```

Depending on how your cluster was set up, you may also need to specify a namespace with the following flag: `--namespace my-namespace`.

2.3 Default Configuration

The following tables list the configurable parameters of the Dask chart and their default values. Note: the container images are not provided by default, you have to specify them in a custom values.yaml file or as a command line argument.

2.3.1 Dask scheduler

Parameter	Description	Default
`scheduler.name`	Dask scheduler name	`scheduler`
`scheduler.image`	Container image name	`""`
`scheduler.imageTag`	Container image tag	`""`
`scheduler.replicas`	k8s deployment replicas	`1`
`scheduler.tolerations`	Tolerations	`[]`
`scheduler.nodeSelector`	nodeSelector	`{}`
`scheduler.affinity`	Container affinity	`{}`

2.3.2 Dask webUI

Parameter	Description	Default
`webUI.name`	Dask webui name	`webui`
`webUI.servicePort`	k8s service port	`80`
`webUI.ingress.enabled`	Enable ingress controller resource	false
`webUI.ingress.hostname`	Ingress resource hostnames	dask-ui.example.com
`webUI.ingress.tls`	Ingress TLS configuration	false
`webUI.ingress.secretName`	Ingress TLS secret name	`dask-scheduler-tls`
`webUI.ingress.annotations`	Ingress annotations configuration	null

2.3.3 Dask worker

Parameter	Description	Default
`worker.name`	Dask worker name	`worker`
`worker.image`	Container image name	`""`
`worker.imageTag`	Container image tag	`""`
`worker.replicas`	k8s hpa and deployment replicas	`2`
`worker.resources`	Container resources	`{}`
`worker.tolerations`	Tolerations	`[]`
`worker.nodeSelector`	nodeSelector	`{}`
`worker.affinity`	Container affinity	`{}`
`worker.port`	Worker port (defaults to random)	`""`

2.3.4 Jupyter

Parameter	Description	Default
`jupyter.name`	Jupyter name	`jupyter`
`jupyter.enabled`	Include optional Jupyter server	`false`
`jupyter.image`	Container image name	`""`
`jupyter.imageTag`	Container image tag	`""`
`jupyter.replicas`	k8s deployment replicas	`1`
`jupyter.servicePort`	k8s service port	`80`
`jupyter.resources`	Container resources	`{}`

2.4 Custom Configuration

If you want to change the default parameters, you can do this in two ways.

2.4.1 YAML Config Files

You can update the default parameters in `values.yaml` by creating your own custom YAML config file with the updated parameters, and specify this file when installing your chart with the `-f` flag. Example:

```
helm install test ska-helm/dask -f values.yaml
```

2.4.2 Command-Line Arguments

If you want to change parameters for a specific install without changing `values.yaml`, you can use the `--set key=value[,key=value]` flag when running `helm install`, and it will override any default values. Example:

```
helm install test ska-helm/dask --set jupyter.enabled=false
```

2.5 Changelog

2.5.1 0.2.0

- Jupyter lab notebook deployment added to chart

CHAPTER

THREE

BUFFER

CHAPTER

FOUR

RECEIVE

The aim of this chart is to deploy all the receive workflows. This chart was developed to be as generic as possible by providing appropriate parameters to customise what it deploys. It is implemented as a StatefulSet to provide stable DNS-based IP addresses. This chart helps with reducing the number of helm charts that need to be maintained for SDP deployments.

Currently, it works with the vis-receive workflow. We will be carrying out further testing and adding functionality if required to make it compatible with other receive workflows.

4.1 Deploying receive

Start minikube

```
minikube start --vm-driver=virtualbox
```

There are few setups we need to do before running the workflow.

As an example we will use cbf-sdp visibility. First need to download the `sim-vis.ms.tar.gz` from the CBF-SDP Emulator Repository.

Extract the file:

```
tar -xf sim-vis.ms.tar
```

Create a persistent volume, to do that create a file called `pvc.yaml` and add the following:

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-local
  labels:
    type: local
spec:
  storageClassName: local
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "<path to sim-vis.ms"
---
apiVersion: v1
```

(continues on next page)

(continued from previous page)

```
kind: PersistentVolumeClaim
metadata:
  name: local-pvc
spec:
  storageClassName: local
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      type: local
```

Note - Make sure to update the hostPath.

Create receive namespace

```
kubectl create namespace receive
```

Create persistent volume by executing the following command:

```
kubectl create -f pvc.yaml -n receive
```

After cloning the SDP Helm Deployer Charts repository,

```
cd ska-sdp-helmdeploy-charts/charts
```

To add all the required parameters to start the cbf receiver, need to create a temporary yaml file. Create `test.yaml` file and add the following

```
command:
- emu-recv
- -o
- payload.method=icd
- -o
- reader.num_chan=0
- -o
- reader.num_repeats=1
- -o
- reader.num_timestamps=0
- -o
- reader.start_chan=0
- -o
- reception.num_ports=1
- -o
- reception.outputfilename=/mnt/data/output.ms
- -o
- reception.receiver_port_start=41000
- -o
- transmission.channels_per_stream=4
- -o
- transmission.rate=147500
```

(continues on next page)

(continued from previous page)

```
- -o
- reception.datamodel=/mnt/data/sim-vis.ms
duration: 20
image: nexus.engageska-portugal.pt/ska-docker/cbf_sdp_emulator
length: 10
model:
  name: sim-vis.ms
payload:
  method: icd
pvc:
  name: local-pvc
  path: /mnt/data
reader:
  num_chan: 0
  num_repeats: 1
  num_timestamps: 0
  start_chan: 0
reception:
  num_ports: 1
  outputfilename: output.ms
  receiver_port_start: '41000'
recv_emu: emu-recv
replicas: 1
results:
  push: false
transmission:
  channels_per_stream: 4
  rate: '147500'
version: latest
```

Install the chart

```
helm install recv receive -n receive -f test.yaml
```

The deployment can be monitored using k9s or by running

```
kubectl get all -n receive

NAME                 READY   STATUS    RESTARTS   AGE
pod/recv-receive-0   1/1     Running   0          24s

NAME              TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/receive   ClusterIP   None         <none>        <none>    24s

NAME                             READY   AGE
statefulset.apps/recv-receive    1/1     24s
```

which shows the receive pod and the network service. Once the receive pod enters the running state, we can check if the container has been deployed correctly by running

```
kubectl logs <podname> -n receive
```

which, if the receive application has been successfully launched should show

```
1|2021-08-23T15:29:13.790Z|INFO|MainThread|__init__|utils.py#74||Attempting to build␣
↪model from /mnt/data/sim-vis.ms
Successful readonly open of default-locked table /mnt/data/sim-vis.ms: 22 columns, 1330␣
↪rows
...
1|2021-08-23T15:29:13.905Z|INFO|MainThread|_create_ms|msutils.py#471||Creating MS at␣
↪output.ms for 4 stations and 4 channels starting at 149.900 MHz
Successful read/write open of default-locked table /output.ms/OBSERVATION: 9 columns, 0␣
↪rows
...
1|2021-08-23T15:29:13.945Z|INFO|MainThread|__init__|spead2_receivers.py#85||Creating␣
↪stream with 1 UDP readers to receive data for 4 channels
...
1|2021-08-23T15:29:13.946Z|INFO|MainThread|_setup_streams|spead2_receivers.py
↪#99||Started udp_reader on port 41000
```

CHAPTER

FIVE

WORKFLOW

CHAPTER

SIX

PSS RECEIVE

PSS receive is an SDP component whose purpose is to receive data from the PSS pipeline. PSS searches beamformer data for compelling pulsar and single pulse candidates. It exports those candidates over a spead2 stream to pss-receive. Currently `pss-receive` stores single pulse candidate metadata. In the future, it will be updated to receive pulsar candidate data. These instructions describe the pss-receive helm chart and its deployment.

6.1 Setting up minikube

The following has been tested on Scientific Linux 7.9 using

- minikube v1.21.0
- Kubernetes v1.19.0
- Docker v20.10.6

Start minikube, using a virtualbox driver and the Calico container network interface.

```
minikube start --cpus=16 --memory=32g --driver virtualbox --disk-size=40g --container-
↪runtime=docker --wait=apiserver --kubernetes-version=1.19.0 --cni=calico
```

Configure minikube host kernel parameters

```
minikube ssh <<EOT
sudo sysctl net.core.rmem_max=268435456
sudo sysctl net.core.wmem_max=268435456
sudo sysctl net.core.netdev_max_backlog=65536
sudo sysctl net.core.wmem_default=268435456
sudo sysctl net.core.rmem_default=268435456
exit
EOT
```

Update calico to use larger MTU (maximum transmission unit)

```
kubectl patch configmap/calico-config -n kube-system --type merge -p '{"data":{"veth_mtu
↪": "9000"}}'
kubectl rollout restart daemonset calico-node -n kube-system
```

Create pss namespace

```
kubectl create namespace pss
```

6.2 Deploying pss-receive

The pss-receive kubernetes manifest consists of the pss-receive job (which deploys the correct container image, and opens up the ports (defined in the helm chart), a network service which routes incoming data to the pss-receive pod, and a persistent volume (and persistent volume claim) for storage of the received candidate data.

After cloning the SDP Helm Deployer Charts repository,

```
cd ska-sdp-helmdeploy-charts/charts
```

Install the pss-receive helm chart (this will take ~40 minutes)

```
helm install test pss-receive -n pss --set helmdeploy.createClusterRole=true
```

The deployment can be monitored by running

```
kubectl get all -n pss


NAME                      READY   STATUS    RESTARTS   AGE
pod/pss-receive-v29tx     1/1     Running   0          80m

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/pss-receive     ClusterIP   10.105.199.245   <none>        9021/UDP   80m

NAME                      COMPLETIONS   DURATION   AGE
job.batch/pss-receive     0/1           80m        80m
```

which shows the pss-receive pod and the network service. Once the pss-receive pod enters the running state, we can check if the container has been deployed correctly by running

```
kubectl logs pss-receive-v29tx -n pss
```

which, if the pss-receive application has been successfully launched should show

```
[debug][tid=140659546208128][/opt/pss-pipeline/thirdparty/cheetah/cheetah/../cheetah/
↪exporters/detail/DataExport.cpp:61][1629726512]Creating sink of type sp_candidate_data␣
↪(id=candidate_files)
```

CHAPTER

SEVEN

PLASMA PIPELINE

The plasma-pipeline chart deploys its processes using separate containers in the same pod. That guarantees they will be on the same node and be able to share the memory volume. A StatefulSet with replicas > 1 would create multiple copies of the pod, so that would scale up to multiple nodes.

This chart:

- Creates a Job
- Pod contains four containers:
  - Plasma store process
  - sender
  - receiver (writes to plasma store)
  - MS writer (reads from plasma store)
- Commands are hard-coded
- Two volumes:
  - Memory emptyDir for the Plasma store. This is mounted by the Plasma store process, receiver and MS writer persistent volume claim (PVC) for data. This is not created by the template. This is mounted by the sender, receiver and MS writer

7.1 Deploying plasma pipeline

Start minikube

```
minikube start --vm-driver=virtualbox
```

Create namespace

```
kubectl create namespace plasma
```

After cloning the SDP Helm Deployer Charts repository,

```
cd ska-sdp-helmdeploy-charts/charts
```

Install the chart

```
helm install plasma-test plasma-pipeline -n plasma
```

The deployment can be monitored using k9s or by running

```
kubectl get all -n plasma

NAME                                                READY   STATUS    RESTARTS   AGE
pod/plasma-test-plasma-pipeline-plasma-thg85        0/4     Pending   0          92s

NAME                                                COMPLETIONS   DURATION   AGE
job.batch/plasma-test-plasma-pipeline-plasma        0/1           92s        92s
```

CHAPTER

EIGHT

CBF-SDP EMULATOR

This is an emulator for the Correlator Beamformer (CBF) and SDP recieve workflow interface. It is an extensible and configurable package that has been designed to support multiple communication protocols and technologies and provide a platform for testing consumers of CBF data payloads.

This chart:

- Creates a Job
- Pod contains:
  - Three initContainers to download the input data
  - Three containers: sender, receiver and rclone (to upload results)
- Receive command is templated
- emptyDir volume is mounted by all containers

More details about CBF-SDP Emulator can be found at https://developer.skatelescope.org/projects/cbf-sdp-emulator/en/latest/?badge=latest

CHAPTER

NINE

CBF-SDP EMULATOR MULTICAST EXAMPLE

This chart shows how to run the software contained in this package.

Both the sender and the receiver require an input Measurement Set (MS) file, which is mapped as a Volume in both containers. The source of this volume is fairly free, and is specified by the `input.mountType` and `inptut.mountTypeOptions` Helm values. Their default values are:

```
mountType: hostPath
mountTypeOptions:
   type: Directory
   path: /tmp/input.ms
```

Any other values should work correctly though, giving users the ability to specify any mount type for this volume.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search