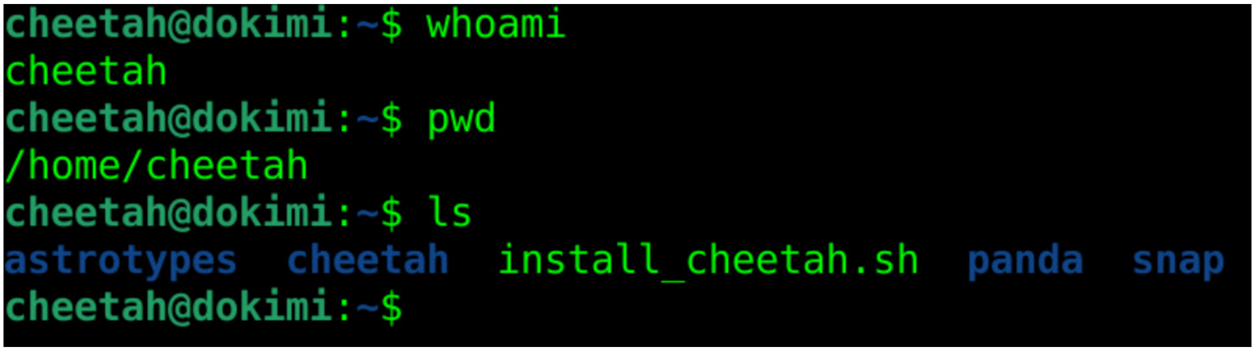
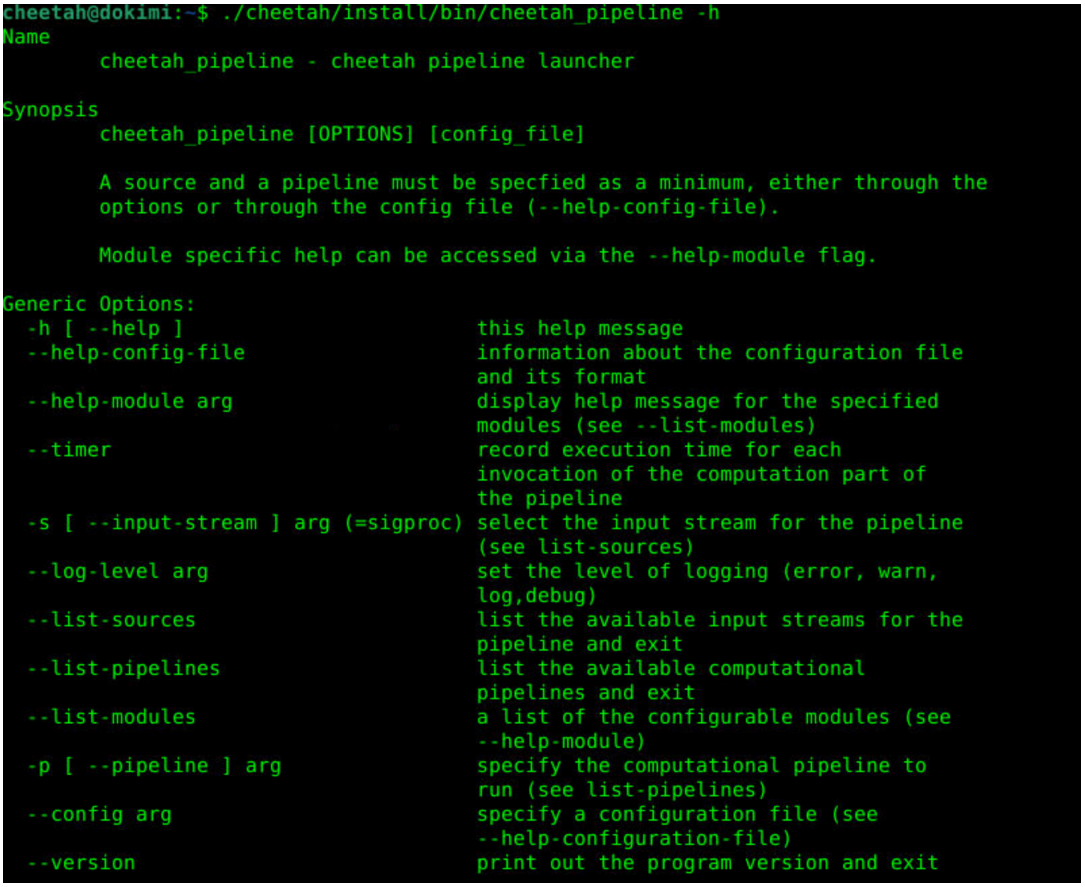
Below we detail the process of deploying the PSS.LMC and using it to control a cheetah pipeline on a physical server.

Firstly, on the physical server (in this case, dokimi) we have a user called *cheetah* and in this user's home area we have a build of cheetah and its dependencies (panda and astrotypes).



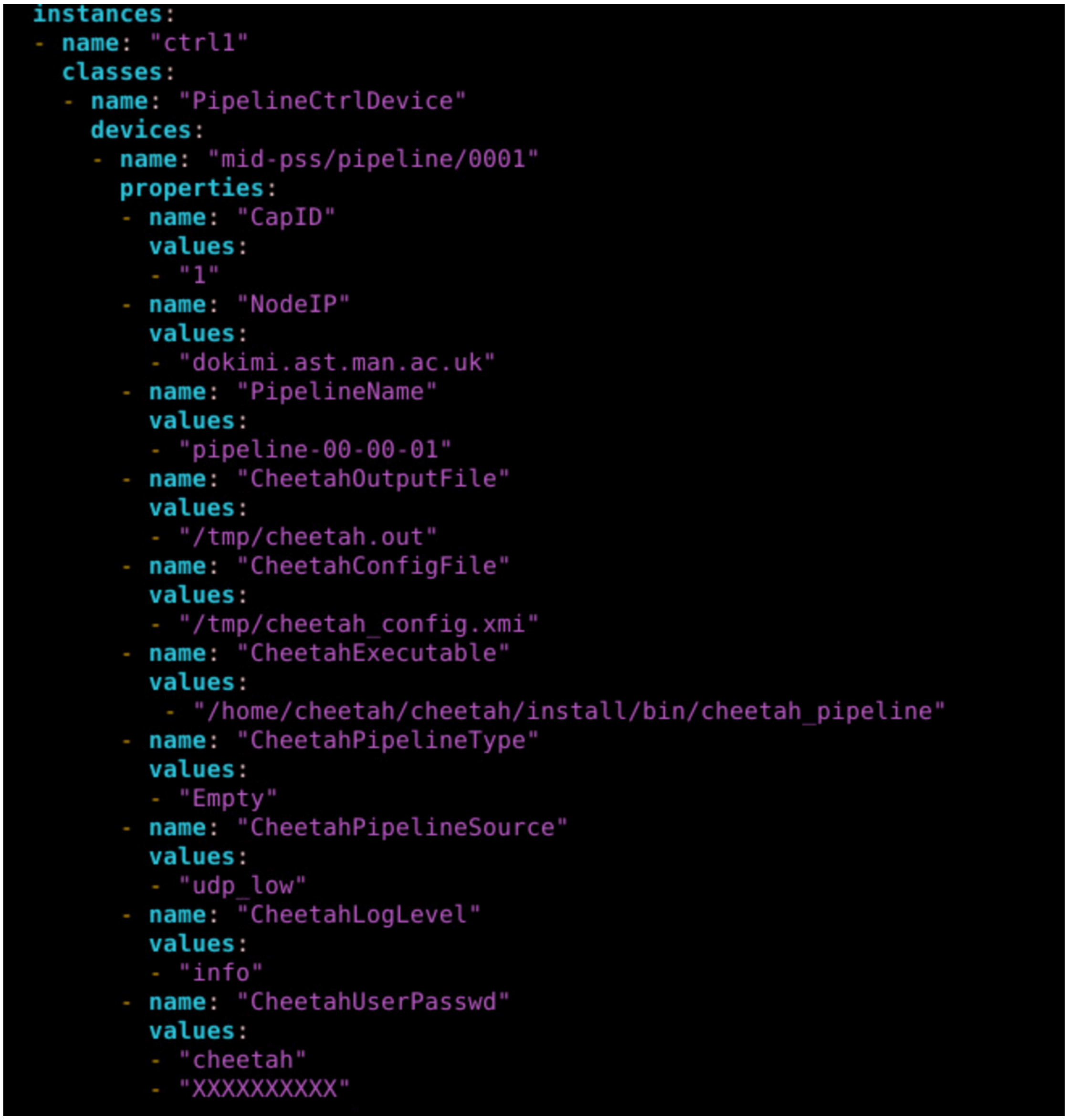
```
cheetah@dokimi:~$ whoami
cheetah
cheetah@dokimi:~$ pwd
/home/cheetah
cheetah@dokimi:~$ ls
astrotypes  cheetah  install_cheetah.sh  panda  snap
cheetah@dokimi:~$
```

From here, we can launch a cheetah pipeline. Here we are running it to see the help menu (note the path to the executable - we'll need this shortly).



```
cheetah@dokimi:~$ ./cheetah/install/bin/cheetah_pipeline -h
Name
        cheetah_pipeline - cheetah pipeline launcher

Synopsis
        cheetah_pipeline [OPTIONS] [config_file]

        A source and a pipeline must be specfied as a minimum, either through the
        options or through the config file (--help-config-file).

        Module specific help can be accessed via the --help-module flag.

Generic Options:
  -h [ --help ]                       this help message
  --help-config-file                  information about the configuration file
                                      and its format
  --help-module arg                   display help message for the specified
                                      modules (see --list-modules)
  --timer                             record execution time for each
                                      invocation of the computation part of
                                      the pipeline
  -s [ --input-stream ] arg (=sigproc) select the input stream for the pipeline
                                      (see list-sources)
  --log-level arg                     set the level of logging (error, warn,
                                      log,debug)
  --list-sources                      list the available input streams for the
                                      pipeline and exit
  --list-pipelines                    list the available computational
                                      pipelines and exit
  --list-modules                      a list of the configurable modules (see
                                      --help-module)
  -p [ --pipeline ] arg               specify the computational pipeline to
                                      run (see list-pipelines)
  --config arg                        specify a configuration file (see
                                      --help-configuration-file)
  --version                           print out the program version and exit
```

Now, assuming we've clone the ska-pss-lmc repo and have a running minikube environment somewhere (in this case, also dokimi), we have configure our PSS.LMC deployment such that it can control processes on the host. The first thing we need to do is edit the helm chart to tell PSS.LMC which machine we want to run cheetah on. We find this file in *charts/ska-pss-lmc/data/psspipelinectrl.yaml.* We can edit it to reflect our requirements as follows.



```
instances:
- name: "ctrl1"
  classes:
  - name: "PipelineCtrlDevice"
    devices:
    - name: "mid-pss/pipeline/0001"
      properties:
      - name: "CapID"
        values:
        - "1"
      - name: "NodeIP"
        values:
        - "dokimi.ast.man.ac.uk"
      - name: "PipelineName"
        values:
        - "pipeline-00-00-01"
      - name: "CheetahOutputFile"
        values:
        - "/tmp/cheetah.out"
      - name: "CheetahConfigFile"
        values:
        - "/tmp/cheetah_config.xmi"
      - name: "CheetahExecutable"
        values:
         - "/home/cheetah/cheetah/install/bin/cheetah_pipeline"
      - name: "CheetahPipelineType"
        values:
        - "Empty"
      - name: "CheetahPipelineSource"
        values:
        - "udp_low"
      - name: "CheetahLogLevel"
        values:
        - "info"
      - name: "CheetahUserPasswd"
        values:
        - "cheetah"
        - "XXXXXXXXXX"
```

Note that we specify a few things here. The hostname of the server on which cheetah will run, and the path to the cheetah executable that we showed above. We also set other command line arguments that this executable needs to run a pipeline. If one were to manually run cheetah from a CLI, we would need to specify the data source, the pipeline type (single-pulse search, acceleration search, etc), and the cheetah logging level. These parameters are also set in this helm chart. Note that we set the pipeline type to be "Empty", as for this demonstration we do not want to actually process any data. Finally we set the username and password for the user (cheetah) that will own the cheetah pipeline process. Password has been redacted.

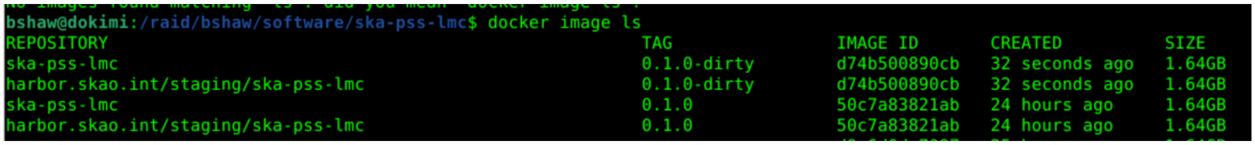
Once we have configured our LMC deployment we can make the LMC container image(s) [For further details, see the LMC documentation] by running:

```
$ make oci-build
```

Now if we run

```
$ docker image ls
```

We can see the LMC images that have been created and these will be deployed into K8s pods into a PSS.LMC namespace.



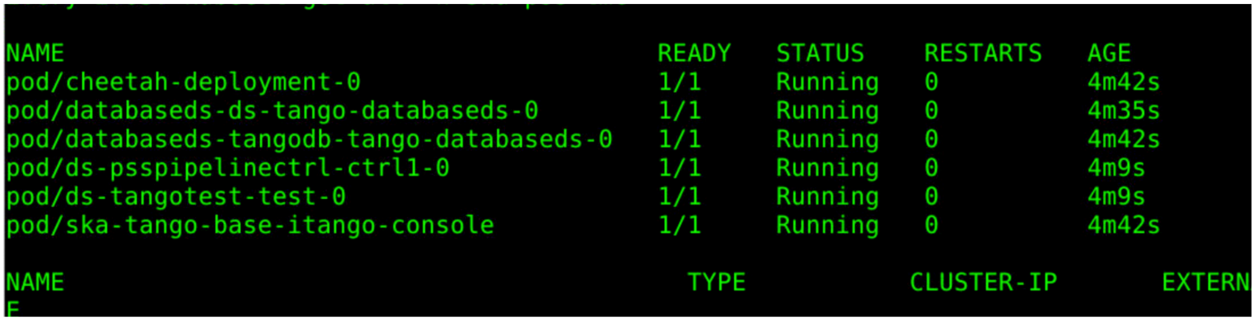
```
bshaw@dokimi:/raid/bshaw/software/ska-pss-lmc$ docker image ls
REPOSITORY                                                     TAG                 IMAGE ID       CREATED          SIZE
ska-pss-lmc                                                    0.1.0-dirty         d74b500890cb   32 seconds ago   1.64GB
harbor.skao.int/staging/ska-pss-lmc                            0.1.0-dirty         d74b500890cb   32 seconds ago   1.64GB
ska-pss-lmc                                                    0.1.0               50c7a83821ab   24 hours ago     1.64GB
harbor.skao.int/staging/ska-pss-lmc                            0.1.0               50c7a83821ab   24 hours ago     1.64GB
```

Now we can deploy the LMC.

```
$ make k8s-install-chart
```

and we can watch it springing to life if we want…

```
$ watch kubectl get all -n ska-pss-lmc
```

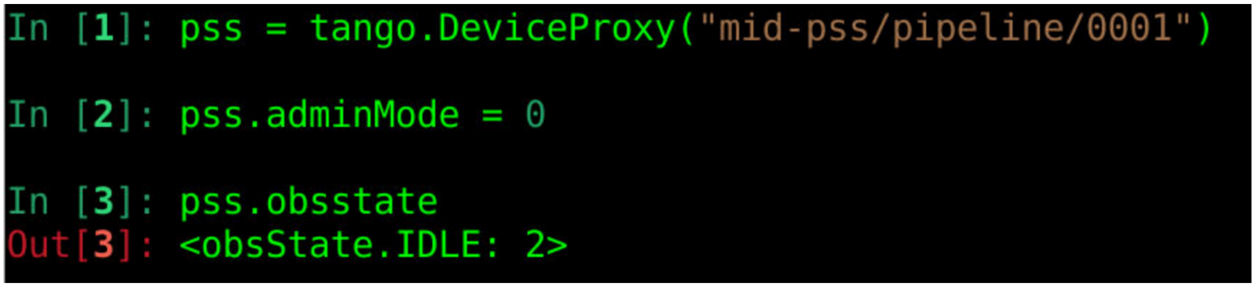


```
NAME                                           READY   STATUS    RESTARTS   AGE
pod/cheetah-deployment-0                       1/1     Running   0          4m42s
pod/databaseds-ds-tango-databaseds-0           1/1     Running   0          4m35s
pod/databaseds-tangodb-tango-databaseds-0      1/1     Running   0          4m42s
pod/ds-psspipelinectrl-ctrl1-0                 1/1     Running   0          4m9s
pod/ds-tangotest-test-0                        1/1     Running   0          4m9s
pod/ska-tango-base-itango-console              1/1     Running   0          4m42s

NAME                                             TYPE             CLUSTER-IP         EXTERN
```

Next we can connect to the itango-console pod and begin controlling the psspipelinectrl-ctrl1-0 device, which will in turn control the cheetah pipeline on dokimi.

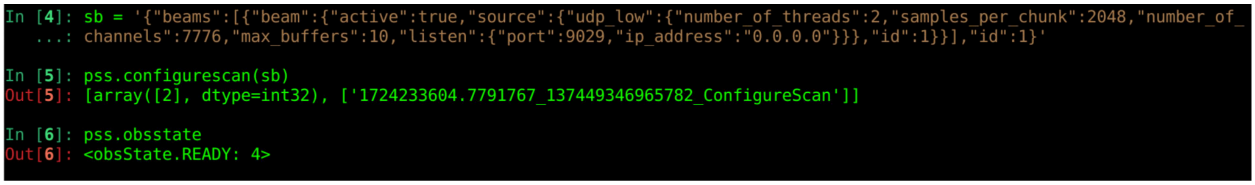
```
$ kubectl exec -it ska-tango-base-itango-console -n ska-pss-lmc -- itango3
```

This will give us an itango interface from which we can connect to CTRL.



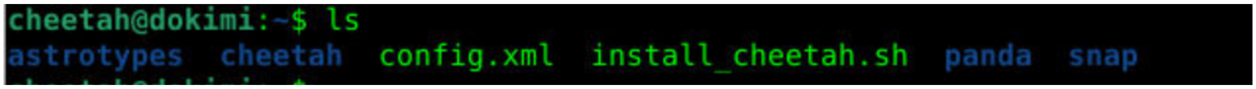
```
In [1]: pss = tango.DeviceProxy("mid-pss/pipeline/0001")

In [2]: pss.adminMode = 0

In [3]: pss.obsstate
Out[3]: <obsState.IDLE: 2>
```

Now we can create a scheduling block (json) which LMC will use to create an XML configuration for the cheetah pipeline. This set of parameters will instruct cheetah to start a single beam, and wait for data. We'll never pass it any, so it will wait indefinitely until it is killed. Here goes.



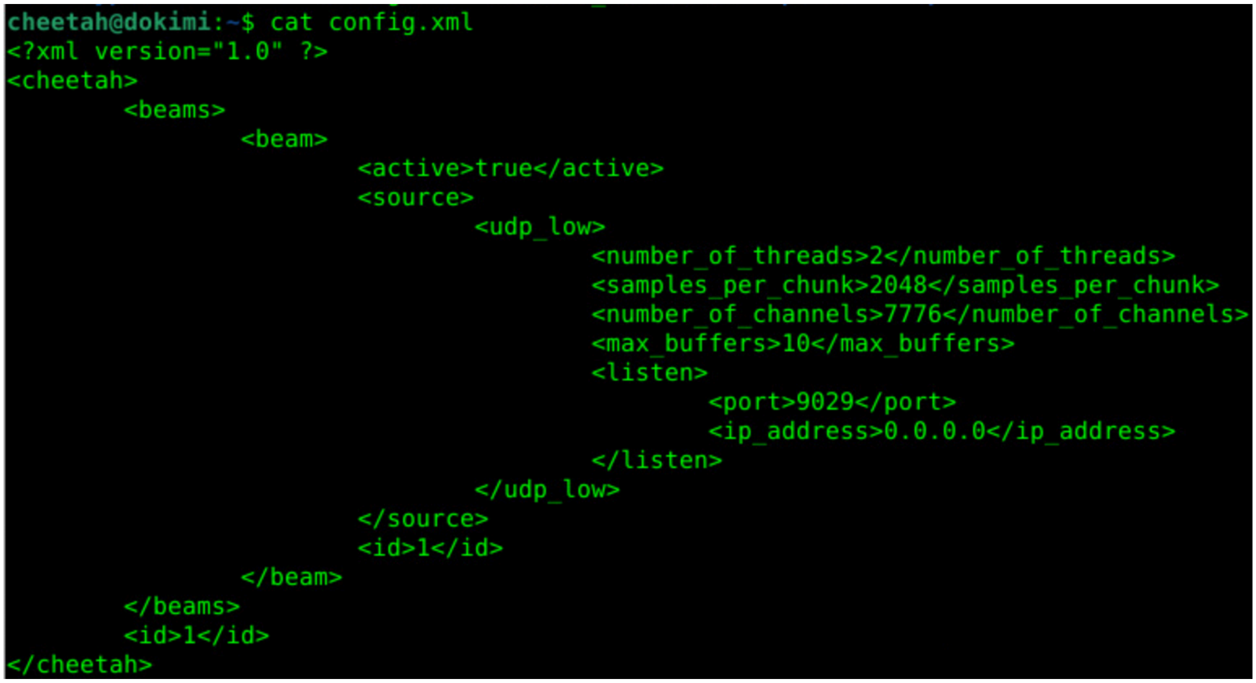
```
In [4]: sb = '{"beams":[{"beam":{"active":true,"source":{"udp_low":{"number_of_threads":2,"samples_per_chunk":2048,"number_of_
   ...: channels":7776,"max_buffers":10,"listen":{"port":9029,"ip_address":"0.0.0.0"}}},"id":1}}],"id":1}'

In [5]: pss.configurescan(sb)
Out[5]: [array([2], dtype=int32), ['1724233604.7791767_137449346965782_ConfigureScan']]

In [6]: pss.obsstate
Out[6]: <obsState.READY: 4>
```

Now if we look in cheetah's home area on dokimi, we'll find a cheetah config file.



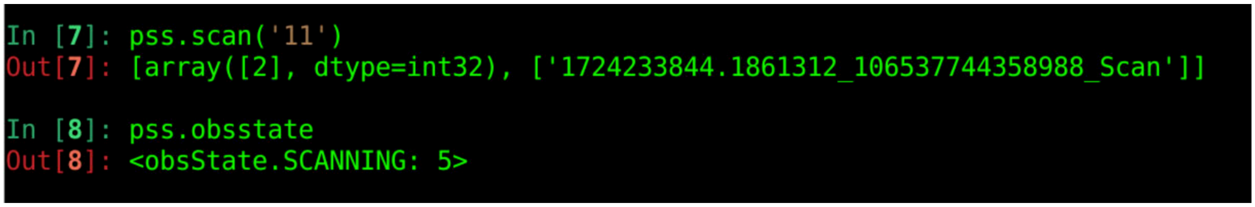
```
cheetah@dokimi:~$ ls
astrotypes  cheetah  config.xml  install_cheetah.sh  panda  snap
```

which looks like this…



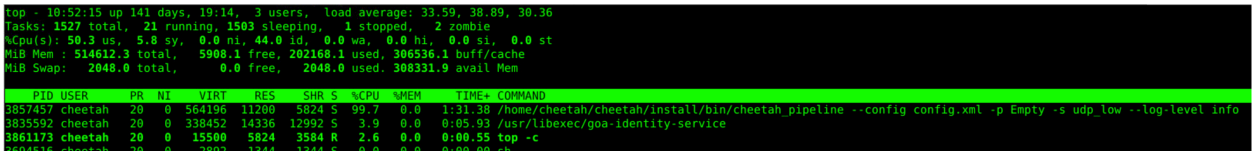
```
cheetah@dokimi:~$ cat config.xml
<?xml version="1.0" ?>
<cheetah>
        <beams>
                <beam>
                        <active>true</active>
                        <source>
                                <udp_low>
                                        <number_of_threads>2</number_of_threads>
                                        <samples_per_chunk>2048</samples_per_chunk>
                                        <number_of_channels>7776</number_of_channels>
                                        <max_buffers>10</max_buffers>
                                        <listen>
                                                <port>9029</port>
                                                <ip_address>0.0.0.0</ip_address>
                                        </listen>
                                </udp_low>
                        </source>
                        <id>1</id>
                </beam>
        </beams>
        <id>1</id>
</cheetah>
```

Next we can start the “scan”. This will execute cheetah, with the above configuration.



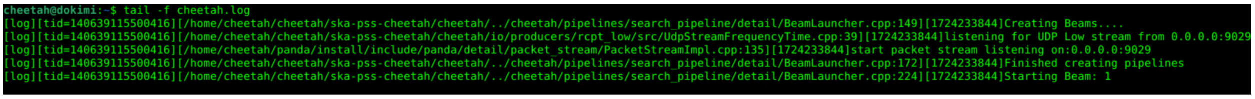
```
In [7]: pss.scan('11')
Out[7]: [array([2], dtype=int32), ['1724233844.1861312_106537744358988_Scan']]

In [8]: pss.obsstate
Out[8]: <obsState.SCANNING: 5>
```

…and if we look at the user cheetah’s processes on dokimi, we can see that cheetah is running and using our config.



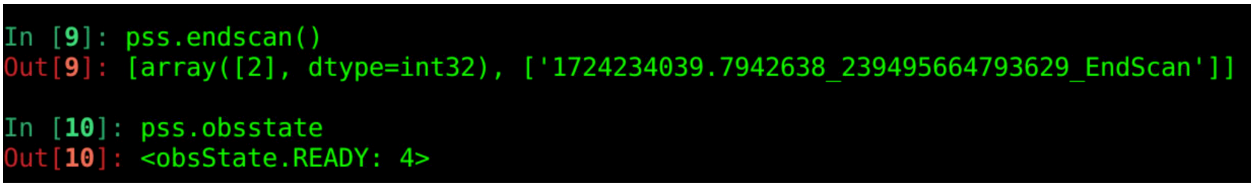
```
top - 10:52:15 up 141 days, 19:14,  3 users,  load average: 33.59, 38.89, 30.36
Tasks: 1527 total,  21 running, 1503 sleeping,   1 stopped,   2 zombie
%Cpu(s): 50.3 us,  5.8 sy,  0.0 ni, 44.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 514612.3 total,   5908.1 free, 202168.1 used, 306536.1 buff/cache
MiB Swap:   2048.0 total,      0.0 free,   2048.0 used. 308331.9 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
3857457 cheetah   20   0  564196  11200   5824 S  99.7   0.0   1:31.38 /home/cheetah/cheetah/install/bin/cheetah_pipeline --config config.xml -p Empty -s udp_low --log-level info
3835592 cheetah   20   0  338452  14336  12992 S   3.9   0.0   0:05.93 /usr/libexec/goa-identity-service
3861173 cheetah   20   0   15500   5824   3584 R   2.6   0.0   0:00.55 top -c
```

and if we want to look at the cheetah logs, we can do that too.



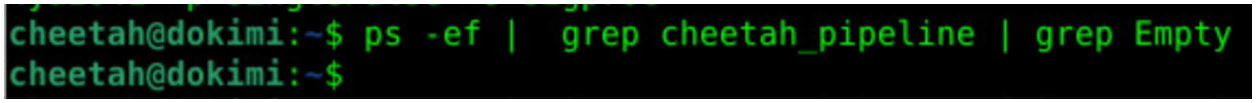
```
cheetah@dokimi:~$ tail -f cheetah.log
[log][tid=140639115500416][/home/cheetah/cheetah/ska-pss-cheetah/cheetah/../cheetah/pipelines/search_pipeline/detail/BeamLauncher.cpp:149][1724233844]Creating Beams....
[log][tid=140639115500416][/home/cheetah/cheetah/ska-pss-cheetah/cheetah/io/producers/rcpt_low/src/UdpStreamFrequencyTime.cpp:39][1724233844]listening for UDP Low stream from 0.0.0.0:9029
[log][tid=140639115500416][/home/cheetah/panda/install/include/panda/detail/packet_stream/PacketStreamImpl.cpp:135][1724233844]start packet stream listening on:0.0.0.0:9029
[log][tid=140639115500416][/home/cheetah/cheetah/ska-pss-cheetah/cheetah/../cheetah/pipelines/search_pipeline/detail/BeamLauncher.cpp:172][1724233844]Finished creating pipelines
[log][tid=140639115500416][/home/cheetah/cheetah/ska-pss-cheetah/cheetah/../cheetah/pipelines/search_pipeline/detail/BeamLauncher.cpp:224][1724233844]Starting Beam: 1
```

Now back to our tango console we can end the scan



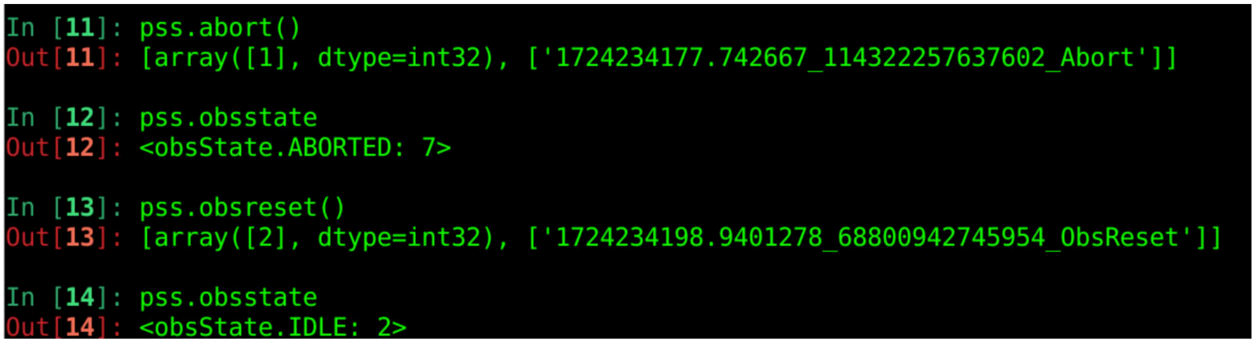
```
In [9]: pss.endscan()
Out[9]: [array([2], dtype=int32), ['1724234039.7942638_239495664793629_EndScan']]

In [10]: pss.obsstate
Out[10]: <obsState.READY: 4>
```

Now back to dokimi, we’ll see that cheetah is no longer running.



```
cheetah@dokimi:~$ ps -ef |  grep cheetah_pipeline | grep Empty
cheetah@dokimi:~$
```

…and back to tango again, and we can shut things down.



```
In [11]: pss.abort()
Out[11]: [array([1], dtype=int32), ['1724234177.742667_114322257637602_Abort']]

In [12]: pss.obsstate
Out[12]: <obsState.ABORTED: 7>

In [13]: pss.obsreset()
Out[13]: [array([2], dtype=int32), ['1724234198.9401278_68800942745954_ObsReset']]

In [14]: pss.obsstate
Out[14]: <obsState.IDLE: 2>
```