project-name Documentation

Release 0.1.0

author

Jan 27, 2021

Contents:

This project defines the integration between various component repository on Kubernetes.

CHAPTER 1

Helm

Helm is a tool for managing Kubernetes charts. Charts are packages of pre-configured Kubernetes resources.

All the charts are included in the folder "charts". Every chart has the following structure:

```
Chart.yaml          # A YAML file containing information about the chart
values.yaml         # The default configuration values for this chart
chart/              # A directory containing any charts upon which this chart depends.
chart/templates/    # A directory of templates that, when combined with values,
                    # will generate valid Kubernetes manifest files.
```

```
# Chart.yaml
apiVersion: v1
appVersion: "1.0"
description: A Helm chart for deploying the Tango-Base on Kubernetes
name: tango-base
version: 0.1.0
```

```
# example of values
tmcprototype:
 enabled: true
 image:
    registry: nexus.engageska-portugal.pt/tango-example
    image: tmcprototype
    tag: latest
    pullPolicy: Always
```

1.1 Update chart settings.

In some cases you may want to alter the settings applied in the chart. E.g To set the Elastic index lifetime management policy to keep logs for 2 days, update *values.yaml* to the following:

```
elastic:
 enabled: true
 image:
  registry: docker.elastic.co
  image: elasticsearch/elasticsearch
  tag: 7.4.2
  pullPolicy: IfNotPresent
 ilm:
  rollover:
   max_size: "1gb"
   max_age: "2d" # Update here
   delete:
   min_age: "1d"
```

More information available here. Helm Glossare here <https://helm.sh/docs/glossary/>‘_.

CHAPTER 2

SKAMPI - SKA Mvp Prototype Integration

2.1 K8s Concepts

The following are key concepts to understand the project:

- Namespace: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/#when-to-use-multiple-namespaces
- Pod: https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/
- Service: https://kubernetes.io/docs/concepts/services-networking/service/
- StatefulSet: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/
- Ingress: https://kubernetes.io/docs/concepts/services-networking/ingress/
- IngressController: https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/
- Traefik IngressController: https://docs.traefik.io/user-guide/kubernetes/
- PersistentVolume: https://kubernetes.io/docs/concepts/storage/persistent-volumes/
- PersistentVolumeClaim: https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims

2.2 KubeCtl references

Overview: https://kubernetes.io/docs/reference/kubectl/overview/ Cheat Sheet: https://kubernetes.io/docs/reference/kubectl/cheatsheet/

2.3 K8s Templates

Template files follow the standard conventions for writing Go templates (see the documentation for details). For example, in the tango-base chart, the following files composes the templates for the generation of valid kubernetes manifest files:

- tangodb.yaml: define a k8s service for maria db and a statefulset attached to it
- tangodb-pv.yaml: define a PersistentVolume and a PersistentVolumeClaim for the database service (tangodb.yaml)
- databaseds.yaml: define a k8s service for the device server Databaseds and a statefulset attached to it
- itango.yaml: define a k8s pod for interacting with other containers (for local testing purpose)
- jive.yaml: define a k8s pod for interacting with the tango jive tool (for local development purpose)
- logviewer.yaml: define a k8s pod for interacting with the tango logviewer tool (for local development purpose)
- tangotest.yaml: define a k8s pod for the tangotest device server

2.4 K8s Tags

Below there are the main tags that constitute every object in the k8s integration.

2.4.1 Metadata tag

Every yaml file has a metadata tag which specify some important information like:

- name: a string that uniquely identifies this object within the current namespace (see the identifiers docs). This value is used in the path when retrieving an individual object.
- namespace: a namespace is a DNS compatible label that objects are subdivided into.
- **labels: a map of string keys and values that can be used to organize and categorize objects**
  - app: unique name (equals to name above)
  - chart: name of the chart
  - release and heritage: used by helm for install/upgrade

2.4.2 Spec

Every yaml file has a spec tag which is used to set all the parameters for a specific object. For instance, in databaseds.yaml the StatefulSet object specifies that the label 'app' should match with a specific value and that the related service is the one specified in the tag 'serviceName'.

```
selector:
  matchLabels:
    app: databaseds-{{ template "tango-base.name" . }}-{{ .Release.Name }}
serviceName: databaseds-{{ template "tango-base.name" . }}-{{ .Release.Name }}
```

2.4.3 initContainers

A Pod can have multiple Containers running apps within it, but it can also have one or more Init Containers, which are run before the app Containers are started. Check documentation for more information.

2.4.4 containers

The containers tag includes the containers that form the specific pod or object whithin k8s.

CHAPTER 3

Environments

Two environments has been defined for the SKAMPI repository, namely “test” and “staging”, both deployed into a k8s cluster linked in the Gitlab.

Those are visible in gitlab at this link and managed with schedule in the gitlab schedule tab.

3.1 Test environment

The test environment is deployed at every commit of the skampi repository. Every day it is scheduled a clean job (at 4 UTC am) ehich redeploy the entire project.

The primary ingress of this environment is the following link: http://integration.engageska-portugal.pt/

3.2 Staging environment

Staging environment is deployed only with scheduled job every 15 days.

CHAPTER 4

Deployment

SCAMPI deployment must be robust, repeatable, and idempotent. We have multiple flavours of deployment for different configurations.

4.1 Flavours

By running *make* in the command line, we can see all the **targets** and **arguments** (and their defaults) available.

```
[user@pc skampi]$ make
make targets:
Makefile:delete_all            delete ALL of the helm chart release
Makefile:delete                delete the helm chart release. @param: same as deploy_
↪all, plus HELM_CHART
Makefile:delete_etcd           Remove etcd-operator from namespace
Makefile:delete_gangway        delete install gangway authentication for gitlab.␣
↪@param: CLIENT_ID, CLIENT_SECRET, INGRESS_HOST, CLUSTER_NAME, API_SERVER_IP, API_
↪SERVER_PORT
Makefile:delete_traefik        delete the helm chart for traefik. @param: EXTERNAL_IP
...
make vars (+defaults):
dev-testing.mk:hostPort        2020
dev-testing.mk:k8_path         $(shell echo ~/.minikube)
dev-testing.mk:kube_path       $(shell echo ~/.kube)
Makefile:API_SERVER_IP         $(THIS_HOST)## Api server IP of k8s

Makefile:API_SERVER_PORT       6443         # Api server port of k8s
```

All the next deployments are deployed using using the same makefile.

4.1.1 Deploy

Deploy only one Helm Chart available at charts directory.

Basic arguments:

- **KUBE_NAMESPACE** - integration **default**
- **HELM_CHART** - tango-base **default**

```
make deploy KUBE_NAMESPACE=integration HELM_CHART=tmc-proto
```

4.1.2 Deploy All

Deploy every helm chart inside charts directory.

Basic parameters:

- **KUBE_NAMESPACE** - integration **default**

```
make deploy_all KUBE_NAMESPACE=integration
```

4.1.3 Deploy All with Order

Deploy every helm chart inside charts directory order by its dependencies.

Basic parameters:

- **KUBE_NAMESPACE** - integration **default**
- **DEPLOYMENT_ORDER** - tango-base cbf-proto csp-proto sdp-prototype tmc-proto oet webjive archiver dsh-lmc-prototype logging skuid **default**

```
make deploy_ordered KUBE_NAMESPACE=integration
```

4.2 Parameters

In SKAMPI, we separated the parameters into two levels. The first one can change the behaviour of the makefile, and the second level can only change the arguments in each chart.

4.2.1 Level 1

The first one is inside the Makefile of the repository and is the top priority meaning that it overrides all the parameters in any level below. We have three ways to customize these parameters and they are prioritize in this order (from most to last important):

1. Command-line arguments - make deploy_ord **KUBE_NAMESPACE=integration**;
2. PrivateRules.mak - Create this file and add arguments. Ex: **HELM_CHART = logging**;
3. *Makefile* defaults - All the defaults are available by running **make** in the command-line.

Please note that one of the parameter at this level is the *DEPLOYMENT_ORDER* which allow ability to select the charts needed for a particular configuration of the deployment (the charts will be deployed in the order or this parameter).

4.2.2 Level 2

The second level is specified with the Values Files.

The priority file is the root directory and goes along the deploy commands with *values.yaml* by default but that could change using the *VALUES* argument in the *makefile*.

```
elastic:
    enabled: false
fluentd:
    enabled: false
kibana:
    enabled: false
tests:
    enabled: false
hdbppdb:
    enabled: false
archiver:
    enabled: false

minikube: true
```

This root values file overrides the *values.yaml* file inside each chart. All chart values files can also be changed to customize your deployment needs.

In the skampi repository, there are 2 examples of values files, one that has everything enabled (pipeline.yaml) and another one with has come charts disabled (values.yaml). The latter disable the logging chart and the archiver chart and it has been thought for a minikube environment.

Please note that the two values file represent the minimum charts needed (values.yaml) for running all the fast tests (mark=fast) while the other (pipeline.yaml) is the complete deployment of SKAMPI.

4.3 Forward Oriented Deployment

With the help of the above parameter levels it is possible to customize the deployment of SKAMPI. It is very important to note that it is possible to deploy the charts incrementally (forward oriented).

CHAPTER 5

Testing SKAMPI

This page is outdated, refer to the System Team for support.

The SKA MPI codebase ultimately holds all the information required to deploy and configure the complete prototype. This information is encapsulated as a collection of Helm charts, Makefiles and any other scripts, components to support its test and deployment.

This page outlines the various categories of testing and approaches one can employ to test various aspects of SKA MPI prototype that can be implemented in this repository.

5.1 Minikube Testing Environment - EngageSKA Openstack

Minikube is a tool that makes it easy to run Kubernetes locally. Minikube runs a single-node Kubernetes cluster inside a Virtual Machine (VM) in Openstack.

5.1.1 Create a Virtual Machine

The first step is to create a Virtual Machine in EngageSKA Openstack: https://developerskatelescopeorg.readthedocs.io/en/latest/services/ait_performance_env.html. The recommended specifications are:

- Volume Size: 100 GB
- Image: Ubuntu 18.04 LTS Jan/2020
- Flavor: m2.small

Don't forget to associate your public key or generate a new key pair in the `Key Pair` section.

Next, go to the `Instances` section and create a new floating IP (dropdown on the right).

5.1.2 Create and test the environment

Install ansible inside the VM and run the ansible-playbooks for creating a development environment and the SHAMPI environment:

```
# Install Ansible
sudo apt-add-repository --yes --update ppa:ansible/ansible && sudo apt-get install␣
↪ansible
# Create Environment
git clone https://gitlab.com/ska-telescope/ansible-playbooks.git
cd ansible-playbooks
ansible-playbook -i hosts deploy_tangoenv.yml
ansible-playbook -i hosts deploy_skampi.yml
```

Verify if everything is running using `kubectl get services -n integration`:

```
NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)␣
↪                                AGE
archiverdb-archiver-test            NodePort    10.96.233.41     <none>        ␣
↪3306:31305/TCP                     5m27s
databaseds-tango-base-test          NodePort    10.105.145.228   <none>        ␣
↪10000:30897/TCP                    5m24s
elastic-logging-test                NodePort    10.103.79.41     <none>        ␣
↪9200:31976/TCP                     5m26s
etcd-restore-operator               ClusterIP   10.100.7.96      <none>        19999/
↪TCP                              5m28s
jupyter-oet-test                    NodePort    10.105.61.127    <none>        ␣
↪8888:32025/TCP                     5m26s
kibana-logging-integration-test     ClusterIP   10.102.79.54     <none>        5601/
↪TCP                               5m26s
mongodb-webjive-test                ClusterIP   None             <none>        27017/
↪TCP                              5m23s
rest-oet-test                       ClusterIP   None             <none>        5000/
↪TCP                               5m25s
ssh-oet-test                        NodePort    10.97.46.250     <none>        ␣
↪22:30520/TCP                       5m25s
tango-rest-tango-base-test          NodePort    10.99.6.220      <none>        ␣
↪8080:32490/TCP                     5m24s
tangodb-tango-base-test             NodePort    10.103.4.193     <none>        ␣
↪3306:31154/TCP                     5m24s
test-sdp-prototype-etcd             ClusterIP   None             <none>        2379/
↪TCP,2380/TCP                     3m18s
test-sdp-prototype-etcd-client      ClusterIP   10.107.155.120   <none>        2379/
↪TCP                              3m18s
test-sdp-prototype-etcd-nodeport    NodePort    10.107.127.158   <none>        ␣
↪2379:30456/TCP                     5m25s
vnc-tango-base-test                 NodePort    10.108.131.141   <none>        ␣
↪5920:30658/TCP,6081:30662/TCP      5m24s
vscode-tango-base-test              NodePort    10.107.133.184   <none>        ␣
↪22:31214/TCP                       5m24s
webjive-webjive-test                ClusterIP   10.111.102.81    <none>        80/TCP,
↪5004/TCP,3012/TCP,8080/TCP    5m23s
```

The next step is to reboot the system with `sudo reboot` and then ssh again into the VM.

Finally, download the SKAMPI repository and run the test in minikube:

```
#Remove the existing skampi directory
sudo rm -rd skampi/
# Download and run test
git clone https://gitlab.com/ska-telescope/skampi.git
cd ansible-playbooks
ansible-playbook deploy_minikube.yml
cd ..
cd skampi/
make deploy_all KUBE_NAMESPACE=integration
```

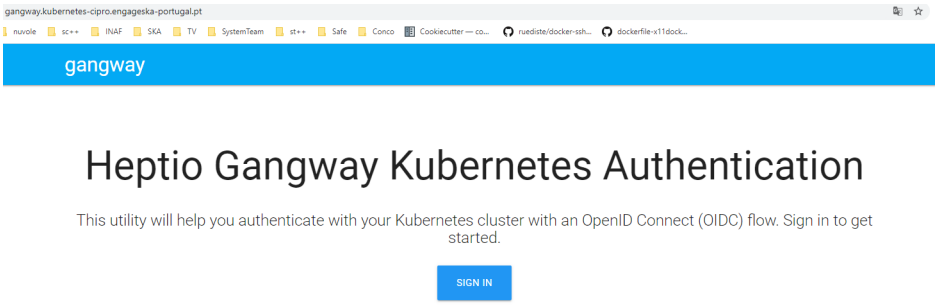
5.2 Kubernetes Testing Environments

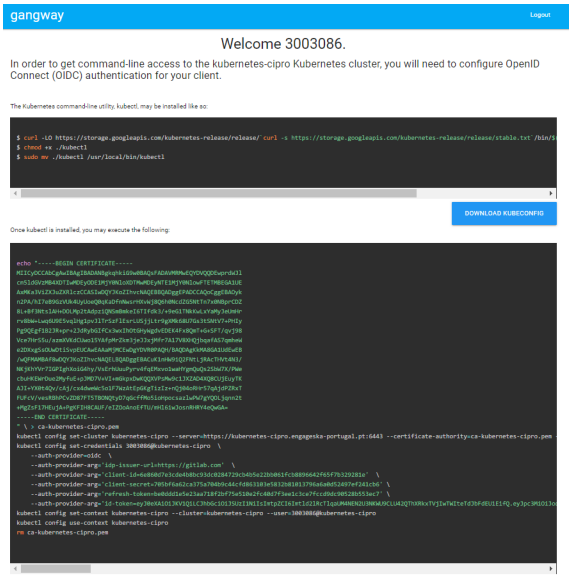
At the moment 3 k8s multi-node clusters are available for testing purpose:

Cluster name	Information
engageska-k8s-master	• 1 master, 4 worker nodes • working in the skampi pipeline • A&A not available
engageska-k8s-v2	• 1 master, 2 worker nodes • working in the skampi pipeline • A&A available. To work with it the file /etc/hosts has to be modified with the following lines: `192.168.93.46 gangway.kubernetes-v2.engageska-portugal.pt`
kubernetes-cipro	• 1 master, 2 worker nodes • not working in the skampi pipeline • A&A available. To work with it the file /etc/hosts has to be modified with the following lines: `192.168.93.46 gangway.kubernetes-cipro.engageska-portugal.pt`

5.2.1 Kubectl setup

If a cluster has the A&A module enabled it is possible to generate the instructions to let the local kubectl work with it. In order To do that, once modified the file /etc/hosts as explained above, open the [gangway](https://github.com/heptiolabs/gangway) url for engageska-k8s-v2 or **'kubernetes-cipro <http://gangway.kubernetes-cipro.engageska-portugal.pt >'_**. The *Sign In* button will redirect to gitlab.com for authentication. Once authenticated it will appear the set of commands to setup the local kubectl as shown below.



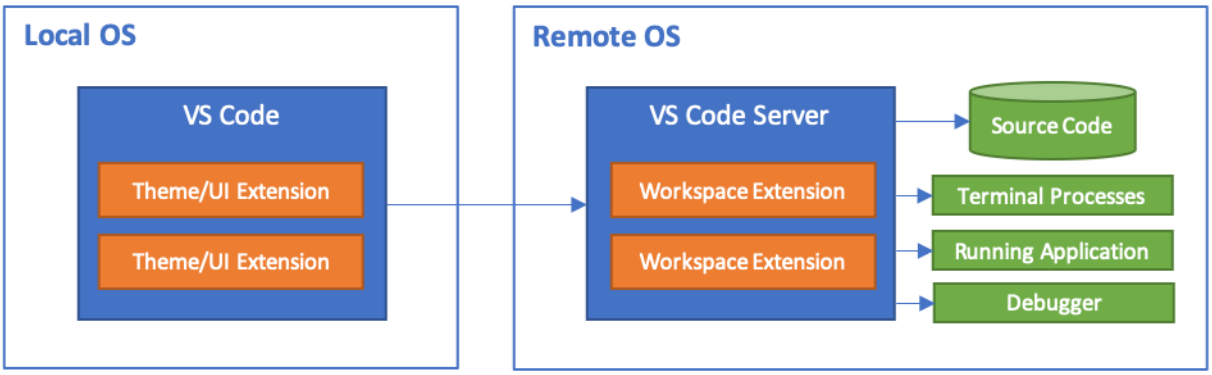


The following namespaces are available for use: “integration”, “sdp”, “csp”, “button”, “ncra”, “karoo”. For creating new namespaces or for any authorization request, contact the system team.

5.3 Visual Studio Code Remote Access

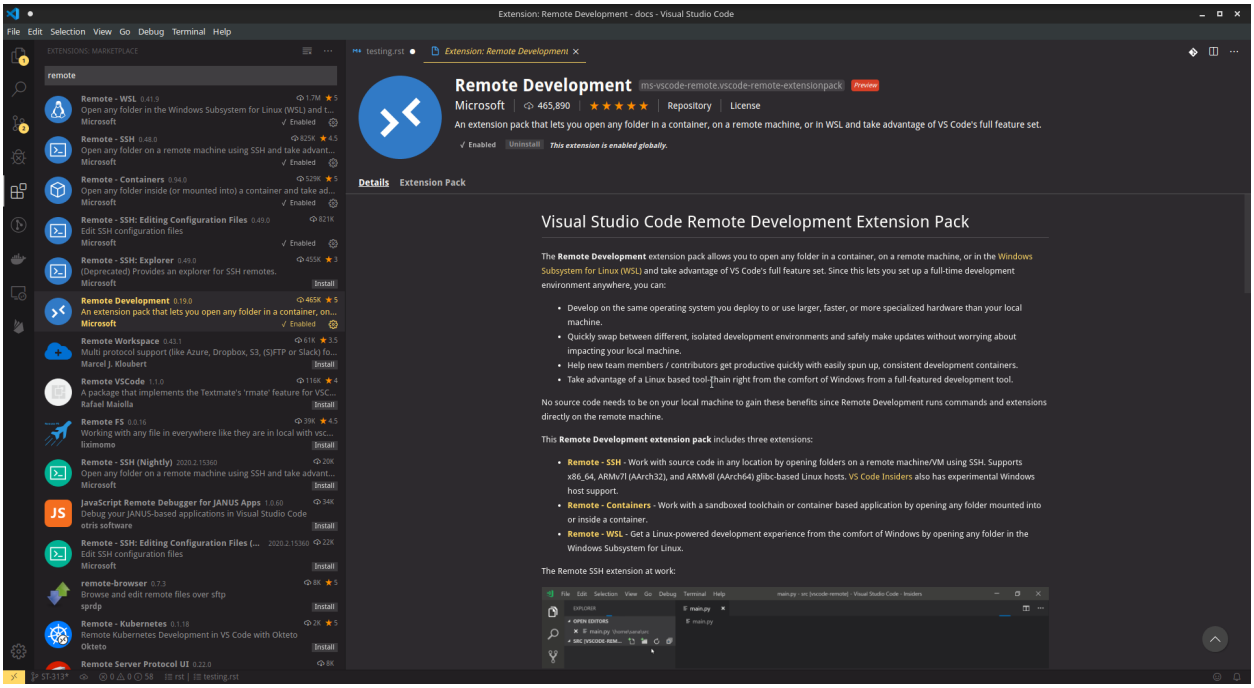
Visual Studio Code Remote Development allows you to use a container, remote machine, or the Windows Subsystem for Linux (WSL) as a full-featured development environment.

No source code needs to be on your local machine. Each extension in the Remote Development extension pack can run commands and other extensions directly inside a container, in WSL, or on a remote machine so that everything feels like it does when you run locally.



5.3.1 Install Extension

Before everything, we need to install the Remote Development extension from vscode.



5.3.2 Create SSH connection

On vscode, open the `Remote-SSH: Open Configuration File...`, copy and paste these properties:

```
Host connection-name          # connection-name -> name of your connection, give any␣
↪name you want
    HostName IP               # IP -> VM's floating IP
    User ubuntu
```

Finally, with the command `ssh connection-name` starts the ssh connection.

5.3.3 Connect to Openstack VM - Option 1

After you created a new ssh connection on your local machine: *Create SSH connection*.

After this, launch the remote extension inside vscode (bottom left icon or use the shortcut `ctrl+shift+P`) and select `Remote-SSH: Connect to Host...` and select the `connection-name` you previously created.

5.3.4 Connect to Kubernetes - Option 2

The tango-base chart available in the skampi repository defines an ssh service which can be used within the vscode extension. The service is deployed in the same IP as the host machine and the port can be discovered with the command `kubectl get services -n integration` which will give you the following output:

```
kubectl get services -n integration
NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)␣
↪                              AGE
...
vscode-tango-base-test              NodePort    10.107.133.184   <none>        ␣
↪22:*PORT*/TCP                      5m24s
...
```

Create new ssh connection on your local machine: *Create SSH connection*. But with this configuration parameters:

```
Host connection-name          # connection_name -> name of your connection, give any␣
↪name you want
    HostName IP               # IP -> VM's floating IP
    Port port                 # port-> vscode server port
    User tango
```

After this, launch the remote extension inside vscode (bottom left icon or use the shortcut `ctrl+shift+P`) and select `Remote-SSH: Connect to Host...` and select the `connection-name` you previously created. Please request the password to the system team.

5.4 Testing Infrastructure as Code

There is a substantial amount of infrastructure and its constituent parts (e.g. Kubernetes resources and their configuration) that forms part of The Telescope. This configuration is orthogonal to the functionality of the software components that are deployed, but changes to them can result in faults in deployment and operation of the system.

Testing at the appropriate level will ensure faster feedback of changes, reducing frustration for everyone and ultimately improve the quality of the system. **Troubleshooting faults in a distributed system caused by a typo in configuration is no fun!**

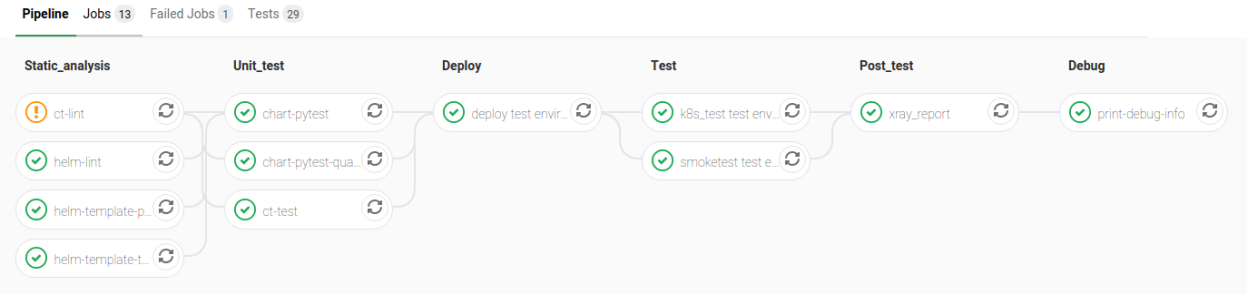
To support different levels of testing, various different jobs are executed as part of the SKAMPI build pipeline and some testware has been developed to aid in testing.

5.4.1 Pipeline Stages for Testing

The stages of the pipeline related to testing are outlined below:

Stage	Description
Static_analysis	Tests aspects of charts that do not require their deployment, e.g. linting
Unit_test [unit]	Tests here might deploy them to an ephemeral test environment.
Test	Tests to be executed in-cluster alongside the fully deployed SKAMPI prototype.

SKAMPI Gitlab CI Pipeline (as of January 2020):



5.4.2 Python testware

Some components have been developed to assist in testing the Helm charts using Python. They are intended to be used with pytest as a test runner and there are currently three jobs in the pipeline that are configured to executed them, filtered based on pytest markers:

Pipeline jobs

- *helm-template-pytest* runs as part of the *Static_analysis* stage in the pipeline executes Python tests marked with `no_deploy`.
- *chart-pytest* runs as part of the *Unit_test* stage and will execute tests marked with the `chart_deploy` marker [unit].
- *chart-pytest-quarantine* also runs during the *Unit_test* stage and executes tests marked with `quarantine` but do not fail the build if they do.

Pytest configuration

As per convention, Pytest is will collect all tests placed in the `/tests/` directory. The following markers are currently defined (see */pytest.ini* for more details):

`no_deploy` Indicates tests that will not require any resources to be deployed into a cluster. Generally, tests that parse and transform the source chart templates.

`chart_deploy` Indicates tests that requires resources to be deployed into cluster such as the Helm chart under test and any other collaborating testware.

`quarantine` Indicates tests that should be executed but not necessarily break the build. Should be used sparingly.

The following custom command-line flags can be passed to Pytest:

`--test-namespace <namespace>` Specify the namespace to use in the test session. Defaults to `ci`.

`--use-tiller-plugin` Indicates that all commands to Helm should be prefixed with `helm tiller run --`. Required when using the helm-tiller plugin.

Test lifecycle

The lifecycle (setup, execute, teardown) of tests are managed by pytest fixtures, defined in */tests/conftest.py*. The `infratest_context` fixture in particular will determine if tests that involve deployments are included in the pytest run, i.e. the `chart_deploy` marker is included. It will then:

1. invoke **kubectl** to create a namespace for the test resources(pods, services, etc.) to be deployed into
2. ensure this namespace is deleted after the test run

Note: the default namespace is `ci`, but can be overriden by specifying the custom pytest option, `--test-namespace`. When running inside the pipeline, this flag is set to `ci-$CI_JOB_ID` so each job will use its own namespace and resources, ensuring test isolation.

Test support

A collection of useful components and functions to assist in testing can be found in the `tests.testsupport` module (*/tests/testsupport/*):

`testsupport.util` Functions that may be useful in testing such as *wait_until* which allows polling, retries and timeouts.

`testsupport.helm.HelmChart` Represents a Helm chart that is the collection of YAML template files and *not necessarily a set of deployed Kubernetes resources*. Primarily used to assist in testing the policies in YAML specifications, i.e. `no_deploy` tests.

`testsupport.helm.ChartDeployment` Represents a deployed Helm chart and offers access to its resources in-cluster their metadata (by querying the Kubernetes API server).

`testsupport.helm.HelmTestAdaptor` A rudimentary adaptor class to manage the interaction with the Helm CLI.

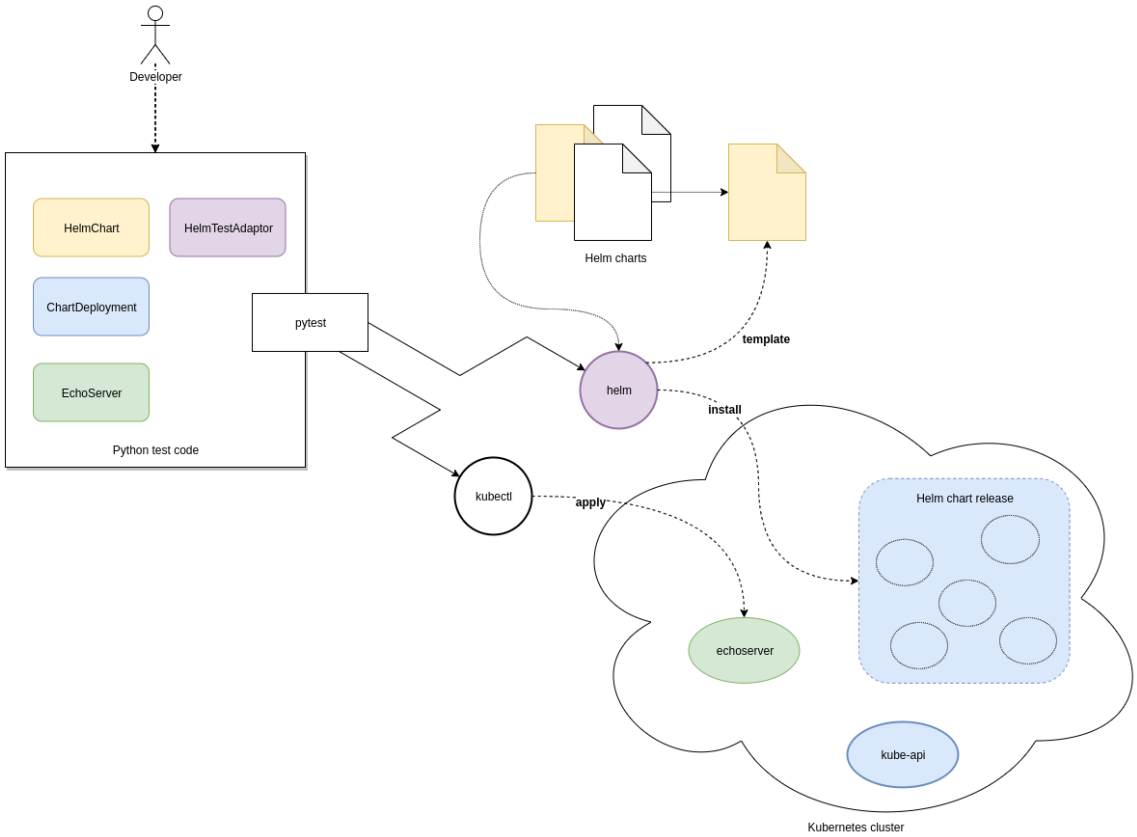
`testsupport.extras.EchoServer` Represents a pod that can be deployed alongside the chart under test, containing a basic Python HTTP server that can receive commands. Currently it only supports echoing any HTTP POST sent to the */echo* path. A handle to this is provided by the *print_to_stdout* method.

Charts are deployed via Helm and the *HelmTestAdaptor*. It's available as a Pytest fixture or you can import it from the `tests.testsupport.helm` module.

The `ChartDeployment` class is an abstraction to represent a deployed chart and offers access to its resources in-cluster (by querying the Kubernetes API) and metadata (such as `release_name`).

In fact, **instantiating a ChartDeployment in code will deploy the specified chart**. A useful pattern is to create Pytest fixture that represents the chart to be deployed and yields a `ChartDeployment` object. It can also call `.delete()` to ensure the chart is deleted and Pytest fixture scope can be used to limit a chart's lifespan. For an example of this see the `tango_base_release` fixture in */tests/tango_base_chart_test.py*.

The diagram below illustrates the relationship between the Python classes in test code, CLI tools and the cluster.



Running locally

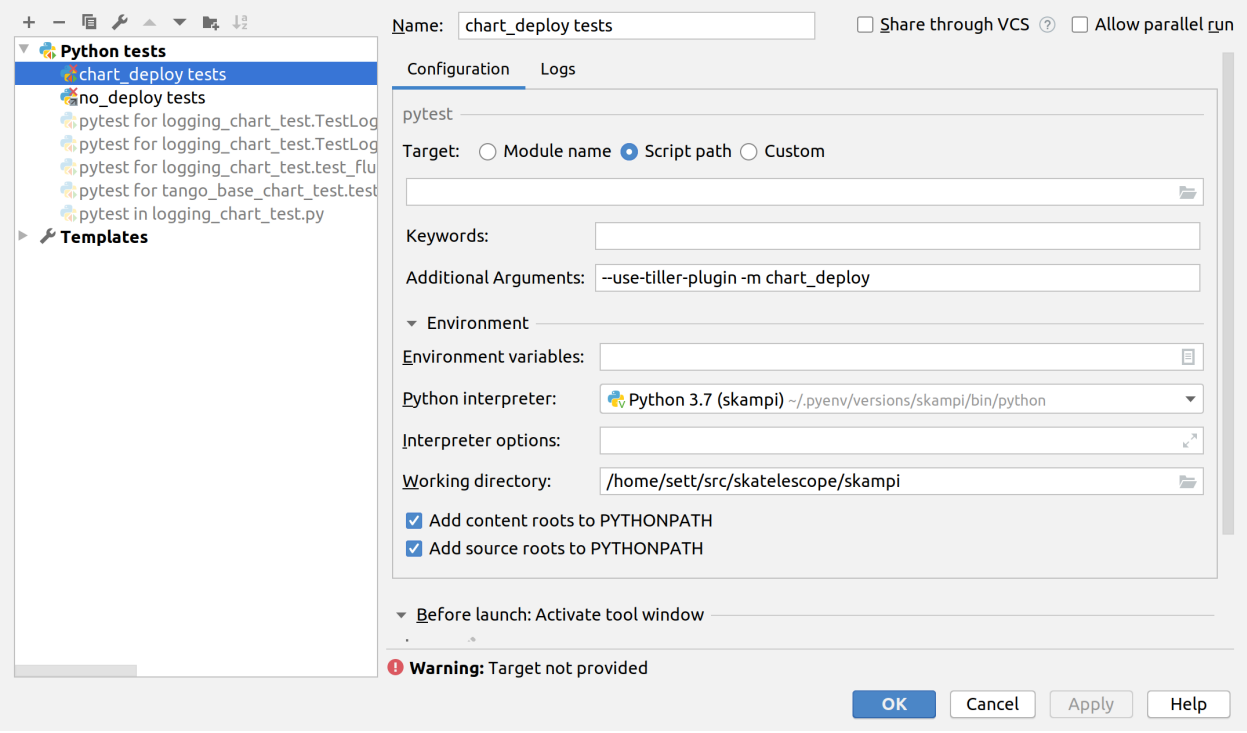
Requirements:

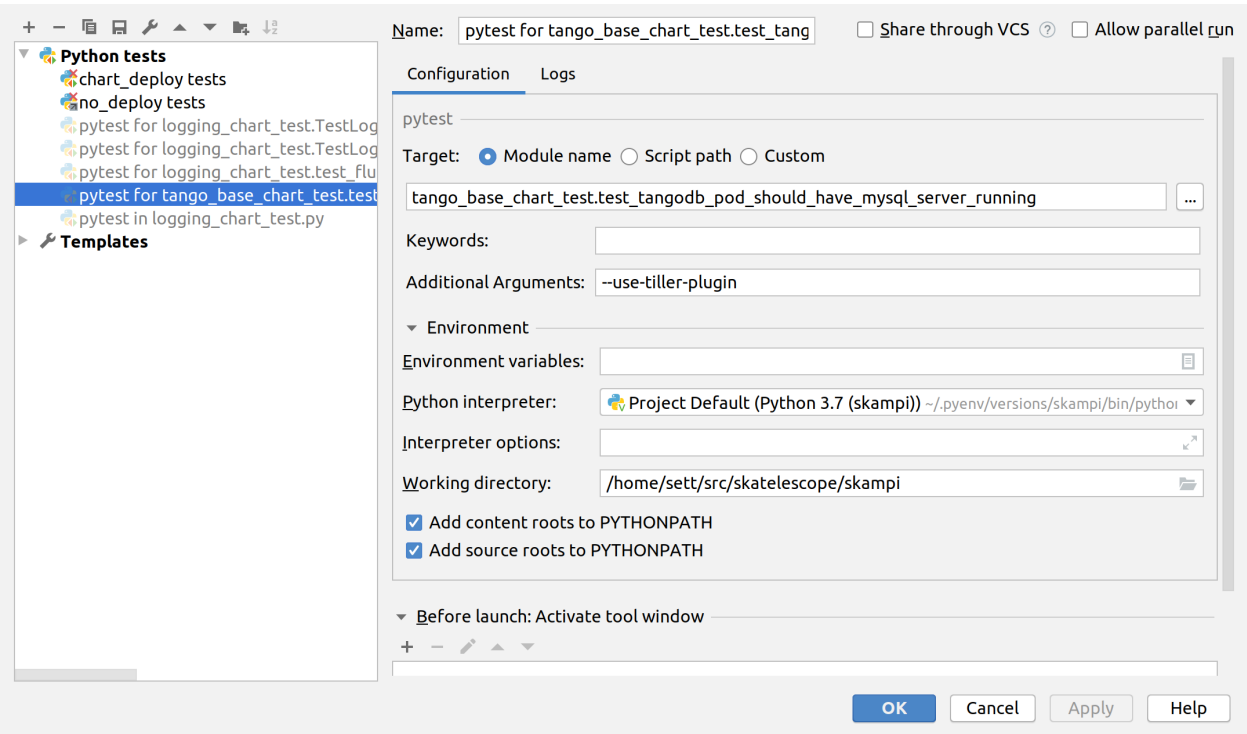
- A Kubernetes cluster (minikube).
- **kubectl** authorized to create namespaces and deploy resources to the cluster.
- **helm v3.0.2**
- **Python 3.7+**

1. Install Python dependencies: `pip install -r test-requirements.txt`
2. Execute only the `no_deploy` tests: `pytest -m "no_deploy and not quarantine"`
3. Execute only the `chart_deploy` tests: `pytest -m "chart_deploy and not quarantine"`
4. Execute the quarantined tests: `pytest -m quarantine`

PyCharm integration

PyCharm as an IDE can be used to run and debug the tests, just be sure to edit the Run/Debug configuration so that it has the appropriate "Additional Arguments" and "Working Directory" (SKAMPI root directory and not */tests*).





Third-party libraries

The following third-party libraries are included in the *test-requirements.txt* and used by the tests and various supporting testware components:

- python kubernetes client is the official kubernetes API client for Python. It's provided as a pytest fixture, `k8s_api` and also used by `ChartDeployment` to obtain a list of deployed pods(see get_pods method).
- testinfra is a library that allows connecting to pods and asserting on the state of various things inside them such as open ports, directory structure, user accounts, etc.
- elasticsearch-py is the official, low-level Python client for ElasticSearch.
- requests is the popular HTTP client library.

CHAPTER 6

Triaging and managing Skampi bugs

This document defines a process for Triaging and managing Skampi bugs so that any SKA team member knows how to handle the funnel of incoming bugs, the allocation, distribution and management of them.

The standard process for changing software includes the following phases:

- Problem/modification identification, classification, and prioritization
- Analysis
- Design
- Implementation
- Regression/system testing
- Acceptance testing
- Delivery

The above process is no different for triaging and managing a bug in skampi. In the present document we will focus on how to identify a problem or bug from incoming information and event notifications and how to assign it to the right team(s).

6.1 Problem identification

The problem identification phase starts when there is an indication of a failure. This information can be raised by a developer (in any shared slack channel like the team-system-support) or by an alert in the following slack channels:

- ci-alerts-mvp
- prometheus-alerts

Any project member can join these channels to gain visibility of this information.

If the information comes from the ci-alerts-mvp then the primary source of detailed information for analysis are the gitlab pipeline logs available here.

Other source of information are:

- kibana (require VPN)
- Node dashboard
- Gitlab runner dashboard
- Gitlab CI Pipeline dashboard
- Docker monitoring dashboard
- K8s cluster summary dashboard
- Ceph Cluster dashboard
- Elasticsearch dashboard

6.2 Allocating ownership to teams

The following are general rules for allocating ownership to teams:

- The primary responsibility for a failed pipeline is the owner of the first commit to the branch since the last successful run of the pipeline. It is therfore the responsibility of the committer to follow up on the pipeline status after each git push.
- For every test case failing, the creator(s) of the test must be involved in order to assign the bug to the appropriate team.
- The System Team should be involved in the problem identification in order to understand whether the problem is infrastructure related (related to a k8s cluster or any layer below it - docker, VM, virtualization etc).
- For prometheus alerts, the system team must provide the analysis of the alert details in order to understand the cause, and give input into assigning it to the right team(s).

6.3 Raising bugs

Bugs are raised following the SKA Bug management guidelines.

CHAPTER 7

Running X based application on SKAMPI

The X Window System is basically a client-server application that allow to display graphical content (i.e. window). In order to allow a container inside the k8s cluster to display GUI we need to reference of the X server. This means that we need to send the X authority file (environment variable XAUTHORITY) for authentication and the address of the server for the communication (environment variable DISPLAY). The latter parameter contains also the indication of the port of the server (if 0, the port is 6000, if 1 the port is 6001 and so on). Unfortunately the engageska cluster does not allow that one of its servers connects outside the internal network at those ports.

7.1 OPTION #1

To avoid the problem, it is possible to use ssh and the X option enabled to connect the engage machine sharing the socket for the X manager of the initial machine (laptop). There are few other options to set that has been included in the deploy_tangoenv playbook. The tango-base chart is already equipped with the possibility to pass the XAUTHORITY and DISPLAY parameters. The following is an example code to display jive:

```
ssh -X -i ~/cloud.key ubuntu@192.168.93.24
git clone https://gitlab.com/ska-telescope/skampi.git
cd skampi
make deploy KUBE_NAMESPACE=integration XAUTHORITY=~/.Xauthority DISPLAY=192.168.100.
↪28:10
```

Note that the IP address 192.168.93.24 is the floating ip while the 192.168.100.28 is the internal private ip address. The number 10 comes from the DISPLAY variable created by ssh.

7.2 OPTION #2

It is possible to enable some tango java applications just exposing few services of the k8s cluster. This is the case of the hdb++ viewer where the exposed services are the archiverdb and the databaseds. In specific, it is possible to work with it exporting the following environment variables:

```
HDB_MYSQL_HOST=192.168.93.47
HDB_MYSQL_PORT=32642
HDB_NAME=hdbpp
HDB_PASSWORD=tango
HDB_TYPE=mysql
HDB_USER=tango
TANGO_HOST=192.168.93.47:32164
```

The port number for the archiver db (HDB_MYSQL_HOST) and for the tango database ds (TANGO_HOST) can be found looking at the deploy_all output:

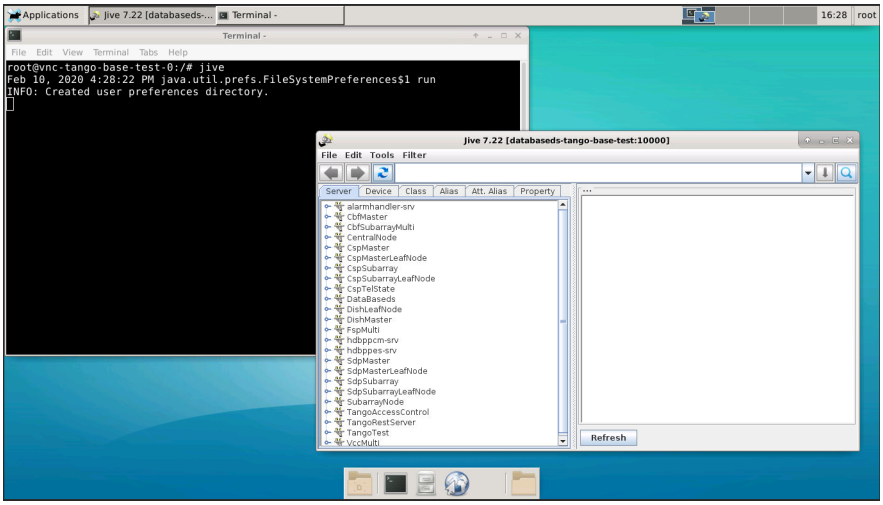
```
service/archiverdb-archiver-test            NodePort    10.100.123.58    <none>
↪3306:32370/TCP                  10h
service/databaseds-tango-base-test          NodePort    10.97.60.174     <none>
↪10000:31088/TCP                 10h
```

7.3 OPTION #3

The X Window System can also be installed into the container itself using a virtual screen. For this purpose it has been developed a docker image called tango-vnc dockerfile to enable vnc and no vnc. It is possible to test this possibility in the pipeline machine at the following link: http://192.168.93.47:31955/vnc.html where the port number (31955) can be found in the deploy_all output:

```
service/vnc-tango-base-test                 NodePort    10.102.239.60    <none>
↪5920:30040/TCP,6081:31955/TCP   10h
```

Please request the password to the system team. Once inside the container, it is possible to open jive with a new terminal window as shown in the figure below.



CHAPTER 8

A&A

In this section there are some instruction on how it is possible to setup the A&A (Authentication and Authorization) in k8s/minikube.

8.1 Static File Authentication

To enable authentication with static file in a minikube environment, start it with the following command (consult the kubernetes documentation for the formatting of *users.csv*):

```
sudo -E minikube start --vm-driver=none \
    --extra-config=kubelet.resolv-conf=/var/run/systemd/resolve/resolv.conf \
    --extra-config=apiserver.basic-auth-file=/var/lib/minikube/certs/users.csv
```

8.2 OpenID Connect Tokens Authentication

To enable authentication with OpenID Connect Tokens in a minikube environment, for instance with Gitlab, start it with the following command:

```
export CLIENT_ID=f84585f68a80c8d6292ec13bb691a19889d80635ffae4e821285c9d3c1980343
sudo -E minikube start --vm-driver=none \
    --extra-config=kubelet.resolv-conf=/var/run/systemd/resolve/resolv.conf \
    --extra-config=apiserver.authorization-mode=RBAC \
    --extra-config=apiserver.oidc-issuer-url=https://gitlab.com \
    --extra-config=apiserver.oidc-username-claim=sub  \
    --extra-config=apiserver.oidc-client-id=$CLIENT_ID
```

The parameter `apiserver.oidc-client-id` must correspond to the application id created in gitlab.

Once the minikube is started, to configure the kubectl tool, it is possible to use gangway. To install it:

```
export from_gitlab_
↪applicationid=f84585f68a80c8d6292ec13bb691a19889d80635ffae4e821285c9d3c1980343
export from_gitlab_
↪applicationsecret=432899cbbb1f0d4dcbef60d38013e5cbfc5b0c6e60d3356207e811508a6ddebc
make gangway CLIENT_ID=$from_gitlab_applicationid \
    CLIENT_SECRET=$from_gitlab_applicationsecret \
    INGRESS_HOST=integration.engageska-portugal.pt \
    API_SERVER_PORT=8443 \
    API_SERVER_IP=xxx.xxx.xxx.xxx
```

The result will be a new ingress at the link *gangway.integration.engageska-portugal.pt*. Remember to modify the file */etc/hosts* adding the following lines:

```
xxx.xxx.xxx.xxx     integration.engageska-portugal.pt
xxx.xxx.xxx.xxx     gangway.integration.engageska-portugal.pt
```

The clusters available in skampi are enabled with the OpenID Connect Tokens Authentication.

CHAPTER 9

Authorization

There are two possibilities for authorization in k8s: the first one is called RBAC (Role-based access control) and the second one is called ABAC (Attribute-based access control).

9.1 RBAC

RBAC allows authorization based on the roles of individual users within an enterprise. A role contains a set of rules which define * an API group (all the k8s api is divided into a set of groups), * a set of resources like pod, deployment and so on, * a set of verbs like get, list and so on

Each role is related to the users with a resource called RoleBinding. The file *roles.yaml* shows an example of Role and RoleBinding which make the user "matteo" able to work (do anything) on the "integration" namespace.

The clusters available in skampi are enabled with RBAC.

9.2 ABAC

ABAC allows authorization according to a set of policies which combine attributes together. The authorization policy is specified into a file with format one JSON object per line. Each line is a policy object containing which specify versioning information and specification, for example:

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1",
  "kind": "Policy",
  "spec": {
    "user": "matteo",
    "namespace": "integration",
    "resource": "",
    "apiGroup": ""}}
```

CHAPTER 10

KUBECONFIG

The command *kubectl config view* shows the current configuration of the running minikube instance. In order to reproduce the PoC of this folder it is necessary to modify it adding the context for the user to access the local cluster (the file *kubeconfig* shows how it has been modified). More information can be found here

CHAPTER 11

Available resources

The folder called “resources” is a collection of resources used for testing and for configuration.

11.1 Makefile targets

This project contains a Makefile which defines the following targets:

Makefile target	Description
vars	Display variables - pass in DISPLAY and XAUTHORITY
k8s	Which kubernetes are we connected to
k8s_test	test the application on K8s
apply	apply resource descriptor k8s.yml
get_versions	lists the container images used for particular pods
logs	POD logs for descriptor
rm	delete applied resources
namespace	create the kubernetes namespace
deploy_all	deploy ALL of the helm chart
deploy_etcd	deploy etcd-operator into namespace
deploy	deploy the helm chart
show	show the helm chart
delete_all	delete ALL of the helm chart release
delete_etcd	Remove etcd-operator from namespace
delete	delete the helm chart release
traefik	install the helm chart for traefik (in the kube-system namespace)
delete_traefik	delete the helm chart for traefik
gangway	install gangway authentication for gitlab (kube-system namespace)
delete_gangway	delete gangway authentication for gitlab
poddescribe	describe Pods executed from Helm chart
podlogs	show Helm chart POD logs
localip	set local Minikube IP in /etc/hosts file for apigateway
help	Show the help summary

11.2 Ansible

It is possible to setup a local SKAMPI environment using the ansible playbook available here.

11.2.1 Creating a K8s Web UI Dashboard

The resources available in this folder enable to create a dashboard (locally available on port 8001) for testing purpose.

The following lines are the commands to run to create it:

```
// create dashboard and add an admin user:
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/master/aio/
↪deploy/recommended/kubernetes-dashboard.yaml

// to access the dashboard it is needed a secret token
kubectl -n kube-system get secret
kubectl -n kube-system describe secret *token* // default generally called default-
↪token-*****

kubectl proxy
```

More information on https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

It is also included an example of graphql query for the webjive application. The graphQl Engine is available in the following path of the integration web server: /gql/graphiql/

11.2.2 Traefik

It is possible to install traefik in different ways:

```
# Install using tiller
helm install stable/traefik --name traefik0 --namespace kube-system --set␣
↪externalIP=xxx.xxx.xxx.xxx

# or Install traefik controller manually (deprecated)
kubectl apply -f traefik-minikube.yaml

# Install using the Makefile
make traefik EXTERNAL_IP=xxx.xxx.xxx.xxx
```

Note that the external ip should be the internal ip of the machine.

11.2.3 Ingress controller commands

```
# The controller is in the kube-system namespace
kubectl get pods -n kube-system
kubectl logs -n kube-system *nginx-ingress-controller-name*
kubectl exec -it -n kube-system *nginx-ingress-controller-name* cat /etc/nginx/nginx.
↪conf
```

Documentation Status

CHAPTER 12

SKA Integration on Kubernetes

The following are a set of instructions of running the SKA application on Kubernetes, and has been tested on minikube v0.34.1 with k8s v1.13.3 on Ubuntu 18.04.

CHAPTER 13

Minikube

Using Minikube enables us to create a single node stand alone Kubernetes cluster for testing purposes. If you already have a cluster at your disposal, then you can skip forward to 'Running the SKA Integration on Kubernetes'.

The generic installation instructions are available at https://kubernetes.io/docs/tasks/tools/install-minikube/.

Minikube requires the Kubernetes runtime, and a host virtualisation layer such as kvm, virtualbox etc. Please refer to the drivers list at https://github.com/kubernetes/minikube/blob/master/docs/drivers.md .

On Ubuntu 18.04 for desktop based development, the most straight forward installation pattern is to go with the `none` driver as the host virtualisation layer. CAUTION: this will install Kubernetes directly on your host and will destroy any existing Kubernetes related configuration you already have (eg: /etc/kubernetes, /var/lib/kubelet, /etc/cni, . . .). This is technically called 'running with scissors', but the trade off in the authors opinion is lower virtualisation overheads and simpler management of storage integration including Xauthority details etc.

The latest version of minikube is found here https://github.com/kubernetes/minikube/releases . Scroll down to the section for Linux, which will have instructions like:

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/v1.2.0/minikube-
↪linux-amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

Now we need to bootstrap minikube so that we have a running cluster based on kvm:

```
sudo -E minikube start --vm-driver=none --extra-config=kubelet.resolv-conf=/var/run/
↪systemd/resolve/resolv.conf
```

This will take some time setting up the vm, and bootstrapping Kubernetes. You will see output like the following when done.

```
$ sudo -E minikube start --vm-driver=none --extra-config=kubelet.resolv-conf=/var/run/
↪systemd/resolve/resolv.conf
  minikube v0.34.1 on linux (amd64)
  Configuring local host environment ...

  The 'none' driver provides limited isolation and may reduce system security and␣
↪reliability.
```

(continues on next page)

(continued from previous page)

```
  For more information, see:
  https://github.com/kubernetes/minikube/blob/master/docs/vmdriver-none.md

  kubectl and minikube configuration will be stored in /home/ubuntu
  To use kubectl or minikube commands as your own user, you may
  need to relocate them. For example, to overwrite your own settings:

     sudo mv /home/ubuntu/.kube /home/ubuntu/.minikube $HOME
     sudo chown -R $USER /home/ubuntu/.kube /home/ubuntu/.minikube

  This can also be done automatically by setting the env var CHANGE_MINIKUBE_NONE_
↪USER=true
  Creating none VM (CPUs=2, Memory=2048MB, Disk=20000MB) ...
  "minikube" IP address is 192.168.86.29
  Configuring Docker as the container runtime ...
  Preparing Kubernetes environment ...
     kubelet.resolv-conf=/var/run/systemd/resolve/resolv.conf
  Pulling images required by Kubernetes v1.13.3 ...
  Launching Kubernetes v1.13.3 using kubeadm ...
  Configuring cluster permissions ...
  Verifying component health .....
  kubectl is now configured to use "minikube"
  Done! Thank you for using minikube!
```

The `--extra-config=kubelet.resolv-conf=/var/run/systemd/resolve/resolv.conf` flag is to deal with the coredns and loopback problems - you may not need this depending on your local setup.

Now fixup your permissions:

```
sudo chown -R ${USER} /home/${USER}/.minikube
sudo chgrp -R ${USER} /home/${USER}/.minikube
sudo chown -R ${USER} /home/${USER}/.kube
sudo chgrp -R ${USER} /home/${USER}/.kube
```

Once completed, minikube will also update your kubectl settings to include the context `current-context: minikube` in `~/.kube/config`. Test that connectivity works with something like:

```
$ kubectl get pods -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-86c58d9df4-5ztg8            1/1     Running   0          3m24s
...
```

13.1 Helm Chart

The Helm Chart based install of the SKA Integration relies on Helm (surprise!). The easiest way to install is using the install script:

```
curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get | bash
```

13.2 Cleaning Up

Note on cleaning up:

```
minikube stop # stop minikube - this can be restarted with minikube start
minikube delete # destroy minikube - totally gone!
rm -rf ~/.kube # local minikube configuration cache
# remove all other minikube related installation files
sudo rm -rf /var/lib/kubeadm.yaml /data/minikube /var/lib/minikube /var/lib/kubelet /
↪etc/kubernetes
```

13.3 Running the SKA Integration on Kubernetes

Note: your Xserver needs to allow TCP connections. This will be different for each window manager, but on Ubuntu 18.04 using gdm3 it can be enabled by editing /etc/gdm3/custom.conf and adding:

```
[security]
DisallowTCP=false
```

In order for these changes to take effect you will need to restart X (it's just easier to reboot...).

Once the Helm client is installed (from above) and TCP based Xserver connections are enabled, change to the k8s/ directory. The basic configuration for each component of the Integration is held in the `values.yaml` file of each chart.

The mode that we are using Helm in here is purely for templating - this avoids the need to install the Tiller process on the Kubernetes cluster, and we don't need to be concerend about making it secure (requires TLS and the setup of a CA).

On for the main event - we launch the Integration with:

```
$ make deploy_all KUBE_NAMESPACE=integration
```

Or we can launch each chart separately with:

```
$ make deploy KUBE_NAMESPACE=integration HELM_CHART=tango-base
$ make deploy KUBE_NAMESPACE=integration HELM_CHART=tmc-proto
$ make deploy KUBE_NAMESPACE=integration HELM_CHART=webjive
```

etc.

This will give extensive output describing what has been deployed in the test namespace:

```
kubectl describe namespace integration || kubectl create namespace integration
Name:         integration
Labels:       <none>
Annotations:  <none>
Status:       Active

No resource quota.

No resource limits.
configmap/tango-config-script-integration-tmc-webui-test created
persistentvolume/rsyslog-integration-tmc-webui-test created
persistentvolumeclaim/rsyslog-integration-tmc-webui-test created
persistentvolume/tangodb-integration-tmc-webui-test created
persistentvolumeclaim/tangodb-integration-tmc-webui-test created
service/databaseds-integration-tmc-webui-test created
statefulset.apps/databaseds-integration-tmc-webui-test created
service/rsyslog-integration-tmc-webui-test created
```

(continues on next page)

(continued from previous page)

```
statefulset.apps/rsyslog-integration-tmc-webui-test created
service/tangodb-integration-tmc-webui-test created
statefulset.apps/tangodb-integration-tmc-webui-test created
service/webjive-integration-tmc-webui-test created
ingress.extensions/webjive-integration-tmc-webui-test created
statefulset.apps/webjive-integration-tmc-webui-test created
pod/tangotest-integration-tmc-webui-test created
pod/tmcprototype-integration-tmc-webui-test created
```

Please wait patiently - it will take time for the Container images to download, and for the database to initialise. After some time, you can check what is running with:

```
watch kubectl get all,pv,pvc,ingress -n integration
```

Which will give output like:

```
Every 2.0s: kubectl get all,pv,pvc -n integration          osboxes: Fri Mar 29␣
↪09:25:05 2019

NAME                                            READY   STATUS             RESTARTS␣
↪AGE
pod/databaseds-integration-tmc-webui-test-0     1/1     Running            3␣
↪117s
pod/rsyslog-integration-tmc-webui-test-0        1/1     Running            0␣
↪117s
pod/tangodb-integration-tmc-webui-test-0        1/1     Running            0␣
↪117s
pod/tangotest-integration-tmc-webui-test        1/1     Running            2␣
↪117s
pod/tmcprototype-integration-tmc-webui-test     4/5     CrashLoopBackOff   2␣
↪117s
pod/webjive-integration-tmc-webui-test-0        4/4     Running            0␣
↪117s

NAME                                              TYPE        CLUSTER-IP      EXTERNAL-
↪IP   PORT(S)           AGE
service/databaseds-integration-tmc-webui-test     ClusterIP   None            <none>␣
↪    10000/TCP         117s
service/rsyslog-integration-tmc-webui-test        ClusterIP   None            <none>␣
↪    514/TCP,514/UDP   117s
service/tangodb-integration-tmc-webui-test        ClusterIP   None            <none>␣
↪    3306/TCP          117s
service/webjive-integration-tmc-webui-test        NodePort    10.100.50.64    <none>␣
↪    8081:31171/TCP    117s

NAME                                                READY   AGE
statefulset.apps/databaseds-integration-tmc-webui-test   1/1     117s
statefulset.apps/rsyslog-integration-tmc-webui-test      1/1     117s
statefulset.apps/tangodb-integration-tmc-webui-test      1/1     117s
statefulset.apps/webjive-integration-tmc-webui-test      1/1     117s

NAME                                                CAPACITY   ACCESS MODES␣
↪RECLAIM POLICY   STATUS   CLAIM                                                 ␣
↪STORAGECLASS   REASON   AGE
persistentvolume/rsyslog-integration-tmc-webui-test   10Gi       RWO            ␣
↪Retain           Bound    integration/rsyslog-integration-tmc-webui-test   standard␣
↪               117s
```

(continues on next page)

(continued from previous page)

```
persistentvolume/tangodb-integration-tmc-webui-test   1Gi        RWO
↪Retain           Bound    integration/tangodb-integration-tmc-webui-test   standard
↪               117s

NAME                                                        STATUS   VOLUME
↪                    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/rsyslog-integration-tmc-webui-test   Bound    rsyslog-
↪integration-tmc-webui-test   10Gi       RWO            standard       117s
persistentvolumeclaim/tangodb-integration-tmc-webui-test   Bound    tangodb-
↪integration-tmc-webui-test   1Gi        RWO            standard       117s

NAME                                                    HOSTS             ADDRESS
↪    PORTS   AGE
ingress.extensions/webjive-integration-tmc-webui-test   integration.ska   193.204.1.
↪157   80      117s
```

If you find that sdp-prototype containers are failing, check whether there is a `test-sdp-prototype-etcd` pod running. If there is not, try running

```
$ make deploy_all KUBE_NAMESPACE=integration
```

again.

To clean up the Helm Chart release:

```
$make delete_all KUBE_NAMESPACE=integration
```

CHAPTER 14

Indices and tables

- genindex
- modindex
- search

Bibliography

[unit] A unit in this context is a Helm chart that can be deployed and tested.