Alejandro Molina Zarca
Rodrigo Asensio Garriga
Jordi Ortiz Murillo

INSPIRE5GPlus SMD Control Fabric from scratch

This document provides detailed information about how to instantiate the UMU Integration fabric from scratch to ease its adoption by the rest of the partners.

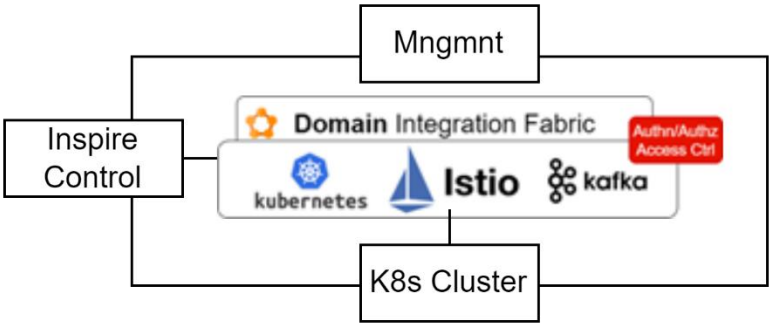

Figure 1: Interfaces

Network configuration:

- Management interface: To manage the VM.
- Inspire Control: Connected to the control plane.
- K8s Cluster: To communicate with other cluster nodes (if any). As master, k8s will be listening in that interface for join requests.

Update the dns:

Update the DNS address with the DNS you will use to resolve your queries.

```
sudo nano /etc/systemd/resolver.conf

DNS=X.X.X.X
```

K8S

Kubeadm install

- Install docker
  - $ curl -fsSL https://get.docker.com -o get-docker.sh
  - $ sudo sh get-docker.sh
  - sudo usermod -aG docker $USER
  - relogin for applying the new group
- Install kubernetes by using the kubernetes script (k8s website) or execute:

```
sudo apt-get update && sudo apt-get install -y apt-transport-https curl

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

```
cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list

deb https://apt.kubernetes.io/ kubernetes-xenial main

EOF

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl
```

- Disable swap before executing init or join
  - `sudo swapoff -a`
- To do it permanent (required) create rc.local and put the command before exit 0
  - `sudo nano /etc/rc.local`

```
#!/bin/sh -e

#

# rc.local

#

# This script is executed at the end of each multiuser runlevel.

# Make sure that the script will "exit 0" on success or any other

# value on error.

#

# In order to enable or disable this script just change the execution

# bits.

#

# By default this script does nothing.

swapoff -a

exit 0
```

- `sudo chmod 744 /etc/rc.local`

Kubeadm init

- `sudo kubeadm init --apiserver-advertise-address=172.16.0.1 --pod-network-cidr=192.168.0.0/16`
  - apiserver-advertise-address: Nodes will use it for connecting to the master
  - pod-network-cidr: This range is for Calico network plugin

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
```

```
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

  sudo chown $(id -u):$(id -g) $HOME/.kube/config

 Alternatively, if you are the root user, you can run:

   export KUBECONFIG=/etc/kubernetes/admin.conf



You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

  https://kubernetes.io/docs/concepts/cluster-administration/addons/

 Then you can join any number of worker nodes by running the following on each as
root:

kubeadm join 172.16.0.1:6443 --token sxisyc.8nzqereu49var2af \

      --discovery-token-ca-cert-hash
sha256:83ca86d88147dd20050ca9b7094801b45d706263c133ed64d2108ccb59837981
```

- `mkdir -p $HOME/.kube`
- `sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config`
- `sudo chown $(id -u):$(id -g) $HOME/.kube/config`

Kubeadm test

- `kubectl get nodes`
  - It will appear as not ready since we need to install the network plugin

Adding the network plugin

- `curl https://docs.projectcalico.org/manifests/calico.yaml -O`
- `kubectl apply -f calico.yaml`
- `kubectl get nodes`
- Now you should see master as ready

Important: If you do not add more nodes you will need to allow deployments on the control plane:

```
kubectl taint nodes --all node-role.kubernetes.io/master-
```

Adding the GUI

Create admin user:

```
kubectl create serviceaccount dashboard-admin-sa

kubectl create clusterrolebinding dashboard-admin-sa --clusterrole=cluster-
admin --serviceaccount=default:dashboard-admin-sa

kubectl get secrets
```

```
kubectl describe secret dashboard-admin-sa-token-xxxxx
```

- Copy the token

Exec the server from the master:

```
kubectl proxy
```

Create a tunnel between the master and your desktop machine:

```
ssh -L 9001 :127.0.0.1:8001 -N -f -l user IP
```

Open the browser:

```
http://localhost:9001 /api/v1/namespaces/kubernetes-
dashboard/services/https:kubernetes-dashboard:/proxy/
```

- Paste the token in the login page

Adding ISTIO

Install istio

- `curl -L https://istio.io/downloadIstio | sh -`
- `export PATH="$PATH:/home/ants/istio-1.10.2/bin"`
- `istioctl x precheck`
- `istioctl install --set profile=demo`

Create namespace with istio

- `kubectl create namespace ants`
- `kubectl label namespace ants istio-injection=enabled`
- `kubectl create namespace admin`
- `kubectl label namespace admin istio-injection=enabled`

Istio dashboard

- Install kali (+prometheus+grafana+jeager):
- cd istio folder
- `kubectl apply -f samples/addons`
- `kubectl rollout status deployment/kiali -n istio-system`
- Exec kiali dashboard
  - `istioctl dashboard kiali`
- Redirect port for kiali
  - `ssh -L 9002 :127.0.0.1:20001 -N -f -l user IP`
  - `ssh -L 9001:127.0.0.1:8001 -L 9002:127.0.0.1:20001 -N -f -l user IP (if you want to maintain also for kubernetes admin dashboard)`
- Enter from the browser in the desktop machine where the redirect port was performed
  - `http://localhost:9002/kiali`

More info at: https://istio.io/latest/docs/setup/getting-started/

MetalLB

```
# see what changes would be made, returns nonzero returncode if different

kubectl get configmap kube-proxy -n kube-system -o yaml | \

sed -e "s/strictARP: false/strictARP: true/" | \

kubectl diff -f - -n kube-system
```

```
 # actually apply the changes, returns nonzero return code on errors only

kubectl get configmap kube-proxy -n kube-system -o yaml | \

sed -e "s/strictARP: false/strictARP: true/" | \

kubectl apply -f - -n kube-system
```

Install metalLB

- kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.5/manifests/namespace.yaml
- kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.9.5/manifests/metallb.yaml
- # On first install only
    - kubectl create secret generic -n metallb-system memberlist --from-literal=secretkey="$(openssl rand -base64 128)"

Put the virtual IP in config.yaml

```
apiVersion: v1

kind: ConfigMap

metadata:

  namespace: metallb-system

  name: config

data:

  config: |

    address-pools:
```

```
  - name: default
    protocol: layer2
    addresses:
    - 10.204.4.3-10.204.4.3
```

- `kubectl apply -f config.yaml`

Verify now have the istio-ingress gw with external-ip

- `kubectl get svc istio-ingressgateway -n istio-system`

More info at: https://metallb.org/

DNS Entry

Request a wildcard entry for your domain in your DNS provider

```
*                   IN      A       10.204.4.3
```

UMU lab example for k8s.gaialab zone:

```
nslookup. kafka.k8s.gaialab
```

Examples and tests

INSPIRE-5GPlus Integration fabric conf files and examples – UMU PoC repository includes examples to use k8s + istio + metalLB + kafka as ZSM integration fabric instantiation inside the INSPIRE-5GPlus H2020 European Project. It also provides examples for deploying control plane services as part of the control plane of the Security Management Domain (SMD) inside the k8s that will use integration fabric capabilities.

Repository URL:

```
https://ants-gitlab.inf.um.es/inspire-5gplus-release/integration-fabric-umu-basics
```

If UMU CA is not recognized in your device, you can use "git -c http.sslVerify=false clone …"

Important!!: The yamls provided for each service must be updated with your own DNS values (e.g., kafka.k8s.gaialab => kafka.YOURDOMAIN) according to your DNS records**.**

Deployment:

```
cd integration-fabric-umu-basics
 ./deploy.sh
```

It will deploy a kafka service, two instances of nginx (with load balancing example properties 90/10), an external service configuration and an internal service to perform different kind of tests. In general, a service definition is composed of three yamls:

- **Service.yaml**: The definition of the service itself.
- **Routing.yaml**: The definition of the routing. How the service will be accessed through the istio mesh.
- **Deployment.yaml**: The definition of the deployment. How many containers, versions etc.

Running the tests:

Test folder contains different tests to verify internal/external messaging, APIs and security features of the fabric.

- cd integration-fabric-umu-basics/test
  - kafka folder:
    - ./external-kafka-consumer-test.sh: Start a kafka consumer from the external metalLB IP
    - ./external-kafka-producer-test.sh: Start a kafka producer from the external metalLB IP
    - ./internal-kafka-consumer-test.sh: Start a kafka consumer from the internal service, and using the internal name (kafka-service)
    - ./internal-kafka-producer-test.sh: Start a kafka producer from the internal service, and using the internal name (kafka-service)
  - smd-service folder:
    - ./external-smd-service-test.sh: Test the nginx service from external metalLB IP
    - ./internal-smd-service-test.sh: Test the nginx service from the internal service, and using the internal name
    - ./show_log.sh [v1/v2]: show the log for the different nginx instances
  - external-service folder:
    - ./external-service-test.sh: HTTP request against a service placed outside of the SMD
  - security folder:
    - Different examples for applying istio security policies

Clean-up:

```
cd integration-fabric-umu-basics
 ./clean-up.sh
```