Application Config & Cert Documentation (Temporary)

Note, this Documentation is for El Alto, on a temporary basis, until it can be entered into "readthedocs" format

PTL Presentation 2019.07.29

WHAT IS TO BE DEMOED?:

AAF will demonstrate the "AAF Agent" container, which will, from a HELM Chart, do the following BEFORE a Client actually starts:

ON a Volume accessible to the Application

- Configure AAF Property Files
- Use this configuration to Contact a Running Certificate Manager
- Generate Certificates signed by ONAP's Test CA
- Validate that the client actually works.

WHAT IS THE EXAMPLE APP?:

The App is a small "Hello" app that is part of AAF, but configured and run separately. It is recommended that you use the actual HELM charts to understand how things work.

Both are provided with the "onap/authz" repository.

AAF provides 2 Kubernetes Helm style installations:

- Official "OOM" deployments
  - Source:
    Currently, in OOM Repo:
    https://gerrit.onap.org/r/gitweb?p=oom.git;a=tree;f=kubernetes/aaf/charts/aaf-hello;h=be5927fcb821138c04f8eb8b5fc1057e095a4713;hb=HEAD

    Note: Sometime during El Alto, it is expected that the OOM AAF Charts will move to "onap/authz" repo.
    https://gerrit.onap.org/r/gitweb?p=aaf%2Foom.git;a=summary
- "Helm" deployment
  - Source: https://gerrit.onap.org/r/gitweb?p=aaf/authz.git;a=tree;f=auth/helm/aaf-hello;h=417360697aa32d428a848d7c0da50293c2c5e4f1;hb=HEAD

WHAT ARE REQUIRED ELEMENTS FOR MY OOM APP?:

1. Correctly configured "Certificate Artifact"

   Jerry Flood created this "for Dummies" Doc, and has graciously allowed me to incorporate (eventually) into El Alto Docs
   AAF Certificate Management for Dummies
2. An App needs a persistent Volume tied to the Kubernetes Namespace
   a. Create a "pv" yaml (Example, see oom/kubernetes/aaf/charts/aaf-hello/templates/aaf-hello.pv.yaml")
   b. Create a "pvc" yaml (Example, see oom/kubernetes/aaf/charts/aaf-hello/templates/aaf-hello.pvc.yaml")
   c. Configure your Volume to your Deployment - NOTE: AS anyone dealing with HELM Charts knows, NO Tabs are allowed, and spacing is CRITICAL!!!. Use the ACTUAL helm chart, and not this summary.
   ..
   kind: Deployment
   ..
   spec:
   ..
   template:
   ..
   spec:
   volumes:
   name: aaf-hello-vol
   persistentVolumeClaim:
   claimName: {{ .Release.Name }}-aaf-hello-pvc
   d. In BOTH the initContainer AND your own Container, make sure you have the SAME Volume Mount
   volumeMounts:
   - mountPath: "/opt/app/osaaf"
   name: aaf-hello-vol

e. Set YOUR Apps' values in "values.yaml"

```
# application image
service:
  fqdn: "aaf-hello"
  agentImage: onap/aaf/aaf_agent:2.1.15-SNAPSHOT
  image: <YOUR IMAGE>
  app_ns: <YOUR AAF Namespace, for "Hello", this is "org.osaaf.aaf" >
  fqi: <YOUR AAF Identity, for "Hello", this is "aaf@aaf.osaaf.org" >
  fqdn: <YOUR FQDN (how your App is known in K8s).  This will be the main Entry for Certificate.  You can add others with SAN.  For
"Hello", this is "aaf-hello" >
  public_fqdn: <This it the PUBLIC name for the Kubernetes Cluster.  For AAF's Demo, this is "aaf.osaaf.org" >
  port: < This is Hello's Internal Port... configure your OWN ports accordingly "8130" >
  public_port: < This is Hello's EXTERNAL Port... configure your OWN ports accordingly "31116" >
  deploy_fqi: < For ONAP Envs, use the "Deployer's Identity" "deployer@people.osaaf.org" >
  cadi_latitude: < Latitude of Installation (will be working on OOM to declare on Nodes) for now, anything reasonable "38.0" >
  cadi_longitude: < Longitude of Installation (will be working on OOM to declare on Nodes) for now, anything reasonable "-72.0" >
```

i. Using these, your "init Container" can be:

```
initContainers:
- name: {{ include "common.name" . }}-config
  image: {{ .Values.global.repository }}/{{.Values.service.agentImage}}
  imagePullPolicy: {{ .Values.global.pullPolicy | default .Values.pullPolicy }}
  volumeMounts:
  - mountPath: "/opt/app/osaaf"
    name: aaf-hello-vol

  command: ["bash","-c","exec /opt/app/aaf_config/bin/agent.sh"]
  env:
    - name: APP_FQI
      value: "{{ .Values.service.fqi }}"
    - name: aaf_locate_url
      value: "https://aaf-locate.{{ .Release.Namespace}}:8095"
    - name: aaf_locator_container
      value: "oom"
    - name: aaf_locator_container_ns
      value: "{{ .Release.Namespace }}"
    -  name: aaf_locator_fqdn
      value: "{{ .Values.service.fqdn }}"
    - name: aaf_locator_app_ns
      value: "{{ .Values.service.app_ns }}"
    - name: DEPLOY_FQI
      value: "{{ .Values.service.deploy_fqi }}"
  # Note: We want to put this in Secrets or at LEAST ConfigMaps
    - name: DEPLOY_PASSWORD
      value: "demo123456!"
  # Note: want to put this on Nodes, evenutally
    - name: cadi_longitude
      value: "{{ .Values.service.cadi_longitude }}"
    - name: cadi_latitude
      value: "{{ .Values.service.cadi_latitude }}"
  # Hello specific. Clients don't don't need this, unless Registering with AAF Locator
    - name: aaf_locator_public_fqdn
      value: "{{.Values.global.aaf.public_fqdn}}"
```

3. WHAT DO I DO WITH MY EXISTING "cadi.properties"?
   a. In MOST cases, you can replace your old "cadi.properties" generated file by using "cadi_prop_files=/opt/app/osaaf/local/<ns>.props" as property in your Container.
   b. IF you need additional properties, for ONAP, it is best to REMOVE any properties generated in the new directory from your existing "cadi.properties", then include the new ones by adding:

      cadi_prop_files=/opt/app/osaaf/local/<ns>.props

      In your existing prop files.
4. DEMO TESTS
   a. OOM (elalto) - use
      kubectl -n onap scale --replicas=1 deployment.extensions/elalto-aaf-hello
      to start the Hello Instance
   b. Helm (in Helm, aaf-hello is a clearly separate app) - use
   c. helm --namespace onap -n hello install aaf-hello
   d. To validate, look at Logs, Container logs and/or actual Volumes
5. NOTES:
   a. Latitude, Longitude and Public FQDN (if required) entries should be attached to actual Nodes/ENV. Will work with OOM
   b. DEPLOY_FQI and DEPLOY PASSWORD are more appropriate as Real-time Administrator entered "Secrets".
      i. ONAP's "Start from Scratch Daily" requirement makes this impossible for ONAP Test Environments. Config Maps may be more appropriate for ONAP Tests.
      ii. Exactly how to get REAL users to use Secrets method while allowing for TESTING Automation is TBA (to be determined)
6. AS STATED ABOVE, this Temporary Documentation will be moved to official "READ THE DOCS" documentation during EL ALTO.

TESTING LOCALLY

If you are testing locally (i.e. DEV Box), remember that TLS (Certs) out-of-the-box requires DNS Entries. If the DNS (name) of, for instance, aaf.osaaf.org, doesn't exist, put it in your /etc/hosts, and TLS will use those instead of DNS

Working on Container Info

To use Container Info, you need to gain access to the volume with the "agent" container. A script is available for you to. Example. See Helm "aaa-hello"

- bash agent.sh
  - This will read the "values.yaml" to get the parameters
- Once in the Command prompt, an "Alias" is provided for you (to see how defined, cat ~/.bashrc)
- You can run important tools, example
  - agent read
    - Will read the FQI/FQDN Certificate Artifact (authorization record)
      2019-08-06T14:24:06.032+0000 INFO [cadi] AAFLocator enabled using https://aaf-locate.onap:8095
      AppID: aaf@aaf.osaaf.org
      Sponsor: aaf_admin@osaaf.org
      Machine: aaf-hello
      CA: local
      Types: pkcs12,script
      Namespace: org.osaaf.aaf
      Directory: /opt/app/osaaf/local
      O/S User: root
      Renew Days: 30
      Notification mailto:
      2019-08-06T14:24:07.124+0000: Trans Info
      Read Artifact 1080.7137ms
  - agent showpass
    - Will decrypt the passcodes etc. Note: You must have logged in as the "Deployer" to do this (with perm to "showpass" (TODO more info on PERM)

      $ agent showpass

      cadi_truststore_password=Tx}WUvfbN#N,lL7h,fW&bU%a
      cadi_key_password=8LZ4aSEP^Qouq[J5m{{(h5+c
      cadi_keystore_password=8LZ4aSEP^Qouq[J5m{{(h5+c
      cadi_keystore_password_p12=8LZ4aSEP^Qouq[J5m{{(h5+c
      Challenge=*z(#X2[kTp3&Y)3HUzKKAw$s
      2019-08-06T14:26:27.500+0000: Trans Info
  - agent validate
    - Will check the configuration, and use to contact AAF for Permissions

      $ agent validate

      ...

      Success connecting to https://aaf-service.onap:8100
        Permissions for aaf@aaf.osaaf.org
      org.access|*|*
      org.osaaf.aaf.access|*|*
      org.osaaf.aaf.cache|*|clear
      org.osaaf.aaf.cache|all|clear
      org.osaaf.aaf.cache|role|clear
      org.osaaf.aaf.password|*|create,reset
      org.osaaf.people.access|*|*
  - Direct access to the "CADI Tool", use agent again. It is also available, if you have locally, "aaf-cadi-core-<VERSION>.jar" in your maven libs, etc.
    $ agent cadi

```
######################################################################
Note: Cadi CmdLine is a separate component.  When running with
Agent, always preface with "cadi",
ex: cadi keygen [<keyfile>]
######################################################################
Usage: java -jar <this jar> ...
  keygen [<keyfile>]                    (Generates Key on file, or Std Out)
  digest [<passwd>|-i|] <keyfile>       (Encrypts Password with "keyfile"
                                         if passwd = -i, will read StdIn
                                         if passwd is blank, will ask securely)
  undigest <enc:...> <keyfile>          (Decrypts Encoded with "keyfile")
  passgen <digits>                      (Generate Password of given size)
  urlgen <digits>                       (Generate URL field of given size)
  encode64 <your text>                  (Encodes to Base64)
  decode64 <base64 encoded text>        (Decodes from Base64)
  encode64url <your text>               (Encodes to Base64 URL charset)
  decode64url <base64url encoded text>  (Decodes from Base64 URL charset)
  sha256 <text> <salts(s)>              (Digest String into SHA256 Hash)
  md5 <text>                            (Digest String into MD5 Hash)
```

```
$ agent cadi passgen 12

79r[WR1{G0E}
```