CLAMP videos

AMSTERDAM Videos

- What is CLAMP ?

3 min video to showcase what CLAMP purpose is and how it materialize a closed loop within ONAP

- CLAMP Holmes Walk through
  This video (49min) walks through the setup of an Holmes Closed loop using Amsterdam release

BEIJING Videos

Design part in SDC

Step1-SDC_Create_Service.mp4

Step2-SDC_Push_Service_To_Test.mp4

Step3-SDC_Approve_Service_In_Test.mp4

Distribution to CLAMP

Step4-SDC_Distribute_Service.mp4

Step5-CLAMP_Check_Distribution.mp4

Configure Closed loop in CLAMP UI

Step6-CLAMP_Configure_ClosedLoop.mp4

Submit Configuration and Operational Policies to Policy Engine

Step7-CLAMP_Submit_ClosedLoop.mp4

Step8-POLICY_Check_Policies.mp4

Deploy/Control the Closed Loop on DCAE

Step9-CLAMP_Deploy_ClosedLoop.mp4

Step10-CLAMP_Deployment_Operations.mp4

CASABLANCA Videos

Control Dasboard part 1 (preview): ingestion of dmaap events file

CL_dashboard_part_01.mp4

Control Loop Dashboard part 2 (preview): Kibana Dashboard

CL_dashboard_part_02.mp4

vCPE Control Loop Animation

vCPE_closed_loop_ons.mp4

Closed Loop with Spark POC

Clamp deploy on spark cluster

FRANKFURT Videos

In this release CLAMP has been completely changed because of the self-serve feature. The policy UIs are not written in the stone anymore, instead they are generated from the policy TOSCA files.

In a nutshell, these are the CLAMP features:

1. Get and store the microservice blueprint (DCAE) + Service data (like VNF data, Vfmodules data) from SDC distribution, use the frankfurt TCA blueprint located here: https://git.onap.org/dcaegen2/platform/blueprints/tree/blueprints/k8s-tca-clampnode.yaml?h=frankfurt
2. Get the CDS workflow inputs from CDS during SDC CSAR installation
3. Store each SDC distribution in "Loop Template", they can be instantiated as a "Loop Instance"
4. Get and store all TOSCA policy files + PDP groups from the policy engine
5. Insertion/Removal of policies in the loop instance.
6. Each policy can be configured according to the TOSCA definition
7. Read TOSCA metadata section to give possible values to the user when it's possible (VNF + VFModules data, CDS payload values)

The SDC distribution did not change since Casablanca, please check the previous video to see how to attach a DCAE blueprint defining the micro-services and distribute it to CLAMP.

Clamp-aaf.mp4

Connection to CLAMP UI

Clamp uses AAF to authenticate the user and get the different permissions.

You can either use the default certificate loaded into AAF for CLAMP Admin user or create a new user in AAF and assign the roles "org.onap.clamp.clds.admin.dev", "org.onap.clamp.clds.designer.dev", "org.onap.clamp.clds.vf_filter_all.dev" and "org.onap.clamp.service".

The default certificate can be found here: https://gerrit.onap.org/r/gitweb?p=clamp.git;a=blob;f=src/main/resources/clds/aaf/org.onap.clamp.p12;h=268aa1a3ce56e01448f8043cc0b05b5fceb5a47d;hb=HEAD

The password is: "China in the Spring"

The certificate must be loaded into your favorite browser before trying to load the CLAMP UI.

Create a loop instance from a loop template

When SDC distributes the service information to CLAMP, loop templates are created in CLAMP for each DCAE blueprint found in the CSAR.

Loop instance can be created from a loop template, each template is therefore related to a "service"

Add or Remove the policies in the loop instance

TOSCA based policies can be added or removed from the loop instance.

CLAMP gets periodically the TOSCA based policies available on the policy engine and stores them into the database, so that the user can select what policies he wants to use in the loop instance.

Add a legacy policy

Operational legacy policy can also be added. This one is not TOSCA based and is the same as the one used in El Alto.

It uses a legacy code in CLAMP

Configure each policy

Each policies in the loop instance must be configured by clicking on each box.

The PDP group must now also be set for the policy engine deployment.

CLAMP gets periodically the available PDP groups from the policy engine per TOSCA based policy

Submit to policy engine

In this video, we show you how to submit all the policies defined in the loop instance to the policy engine.

Deploy microservices to DCAE

In this video, we show you how to deploy the microservices defined in the loop instance (those are initially defined by the DCAE blueprint and distributed from SDC)

Stop/Restart the loop

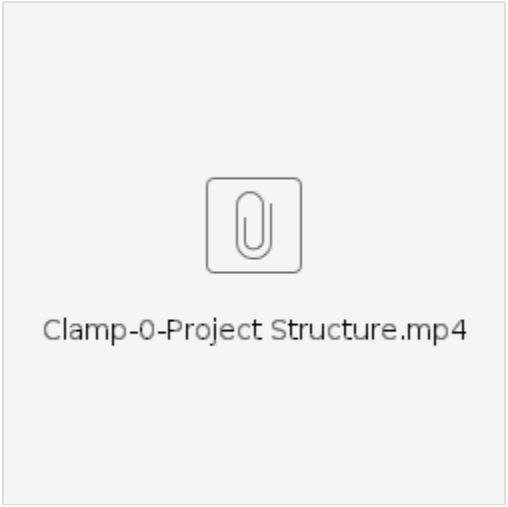
In this video, we show you how to stop or restart the loop instance, it obviously undeploy or re-deploy all the policies from/to the policy engine.

Undeploy the loop

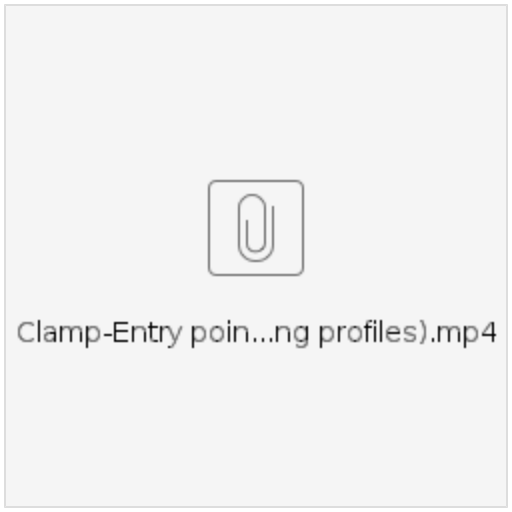
In this video, we show you how to undeploy the microservices from DCAE

Developer's Guide Videos (based on latest release)

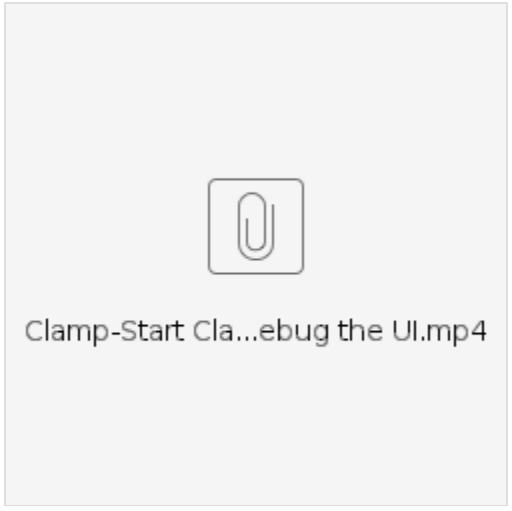
1. The clamp project is structured as a single pom.xml (no maven submodule), it builds the Frontend and the Backend docker images, publishes a Jar on nexus and a NPM package that can be used to get the clamp UI react components. The frontend is located in the "ui-react" folder, and build with NPM into an NGINX docker container. The backend is a standalone Springboot JAR created from the src folder.

2. The backend uses Springboot and Camel, the code entry points are located in "scheduled beans" (policy controller/Sdc Controller) and defined in Camel routes. All Rest api endpoints are defined in Camel. The backend uses also the "spring profiles" to enable or disable features (like sdc controller, aaf cadi, policy controller, etc ...)



3. In the clamp structure, there is a folder extra/bin-for-dev that contains scripts for the developer. They are used to start the mariadb container (preload available), the policy/dcae emulator, the backend from the maven target/JAR (debug port opened) and the frontend from the target/ui-react code (using target folder NodeJS). The main language used between the frontend and backend is JSON, so looking at the Firefox Debugger Networks tab, it's possible to see all the data exchanged and understand the flow and debug the javascript code



4. The code is entirely monitored in term of coverage (used by Sonar), the single maven module facilitates that Sonar report unification. The frontend uses JEST (with snapshots) to do the unit testing of the javascript. The backend is tested with 2 phases, the maven test phase (Junit) and the maven Integration test phase (Junit with SpringBootTest). The integration test phase is crucial as it tests Clamp against a real mariadb, the emulator, and more recently, the Robotframework tests have been introduced during that phase. There are therefore a good tooling chain to validate Clamp code and improves the coverage. **VIDEO TO BE DONE**

5. The backend code has an SDC controller that is used to deploy the service/resources/blueprints created and distributed in SDC. It uses the SDC Client that uses the DMaap notification. The CsarInstaller class is responsible of deploying those artifacts to the clamp database, it creates some "loop templates" that can be used to create a "loop instance". The policy controller is also present to get periodically all the Tosca files defined in the Policy Engine. Those are available to the user for configuring the control loop instance. **VIDEO TO BE DONE**

6. The Frontend does not require any code for the policy UI presented to the user, instead the frontend uses a library that convert a Json Schema to a nice UI. Therefore the backend has a ToscaToJson Converter that is used to convert all tosca based policies. This converter is built to be modular in a sense that plugins can be added to fetch info in third party systems (like CDS), this mechanism is called Json enrichment. The user is therefore able to see in the UI the data coming from others onap/non onap components. The backend supports also a tosca dictionary mechanism that can do simple enrichment for specific tosca keywords defined in that dictionary. **VIDEO TO BE DONE**

7. The backend has an authentication mechanism that is used for each Rest query, there currently 2 Springboot profiles "modules" that be exclusively enabled, either the "cldsDeaultUser" or the AAF one, that uses CADI filter. When using AAF, the user permission are retrieved from the AAF instance, the user can choose a X509 or "basic authentication" login. The front end obviously forward those info to the backend. The frontend is just a renderer without any logic. **VIDEO TO BE DONE**

8. Clamp auto generates the Swagger JSON from the Camel Rest routes automatically. To export that json created only when the backend is up, we use one of the Integration test to export that json into the docs folder. The clamp pom then defines some plugins to convert that JSon to HTML or PDF. There is currently no documentation around those swagger info, it's just a raw Swagger json.