SALSA USER GUIDE

Contents

1. Build and install

The source code of SALSA is hosted on Github: https://github.com/tuwiendsg/SALSA. In order to build the project, user clones the repository and build the parent module by maven:

```
$ mvn clean install
```

The artifacts will be created on particular modules and put all together in “standalone” module. For installing, please following these steps:

1. SALSA is packed as a web application, running on Tomcat server or standalone version with embedded Tomcat. There are two components:

   - The salsa engine: salsa-engine.war (with Tomcat) or salsa-engine.jar (standalone)

   - The salsa pioneer: salsa-pioneer.jar

2. Configure the salsa-engine's parameters in file **salsa.engine.properties** and put in the same folder with the **salsa-engine.jar** or in **/etc/**. (without the file, SALSA use default parameters)
   Note: The most important setting is the access IP (SALSA_CENTER_IP) that the salsa-pioneer can query the salsa-engine. For example: when SALSA is deployed on cloud, user should set the floating IP, otherwise the machine outside the cloud cannot query..

3. The salsa-pioneer.jar need to be put in the same folder with salsa-engine.[war/jar], or upload to a public repository and configured in step 2.

4. Deploy salsa-engine.war to the Tomcat or run the standalone version with port:

```
$ java -jar --httpPort <portNumber> salsa-engine.jar
```

   We also provide a service pattern to run the salsa-engine. Copy the salsa-engine-service script to /etc/init.d and run by:

```
$ sudo service salsa-engine-service start/stop/restart/status
```

5. Access to SALSA web interface with the IP and port setting above, for example:

   http://10.0.0.1:8080/salsa-engine

   Using RESTful API at: **http://10.0.0.1:8080/salsa-engine/rest**

6. If user access to cloud, a configuration have to be in the salsa-engine folder or in /etc. Please check Section 2.1.

2. Configuration stack levels: IaaS, container and application

SALSA support three deployment and configuration stacks:

2.1 IaaS level

SALSA contains a number of cloud connector for automatically provision IaaS cloud resources and managing virtual machines. The prototype support OpenStack, Flexiant, Stratuslab and LocalMachine. The LocalMachine enables user to test application without cloud resources. Please check Section 5 to see how to describe the virtual machine configuration. In order to setup SALSA to run with those clouds, user need to provide a configuration file (**./cloudUserParameters.ini** or **/etc/cloudUserParameters.ini)**. The parameters of all supported providers are put into a single file as below:

```
[<mysite>@openstack]
username=<OpenStack username>
password=<OpenStack password>
tenant=<Project name, tenant name>
keystone_endpoint=<URL to keystone service, e.g http://.../identity/v2.0/>
port=<keystone port>
sshKeyName=<the keypair name, which created before in openstack>
location=<optional localtion information>

[<mysite>@flexiant]
email=<account email>
customerUUID=
password=
endpoint=<The endpoint to JADE API of the datacenter>
vdcUUID=<default value>
defaultProductOfferUUID=<default value>
clusterUUID=<default value>
networkUUID=<default value>
sshkey=<ssh key which will be push to the VM>


[<mysite>@stratuslab]
Endpoint=<VM service endpoint>
pdisk_endpoint=<Storage service endpoint>
```

```
marketplace_endpoint=<Marketplace service endpoint>
username=<user name>
password=<password>
client_path=<local path to the client package>
```

Above configuration depends on the connector implemented and need to match with each provider.

2.2 Container level

SALSA enables the management with different type of containers (e.g. start, stop, create and remove containers) and configure the application level on top of this. The prototype support Tomcat and Docker.

- **Tomcat**: Developer defines the NodeType=”tomcat”. If there is no further information, SALSA automatically uses apt-get (on Ubuntu) to install Tomcat. Otherwise, user can provide a DeploymentArtifact, or define an action name “deploy” to manual configure Tomcat.
- **Docker**: Developer defines the NodeType=”docker”. If there is no further information, SALSA deploys default Ubuntu container and ready to deploy application stack on top of the container. If developer provides a DeploymentArtifact with type=”Dockerfile”, SALSA uses it for configure the Docker container. If the user-defined docker container contains Java, SALSA can push the pioneer to execute the upper stacks configuration.

2.3 Application level

SALSA supports deploying and configuring application via script (written by app. developer). The script MUST be provided as DeploymentArtifact. At this stack, SALSA support writing components (see Section 3). Not only script, SALSA supports some more artifact type (see Section 5).

The stacks are defined by HOSTON relationship in the application description (see Section 4)

3. Wiring application components

Each node has a number of scripts related to deployment actions (e.g. deploy, undeploy, start, stop, registerTo, etc). The deployment scripts can be written a BASH script. In order to wire 2 components (e.g. register a web service to a load balancer, or a database endpoint to web page) the deployment scripts need to share some variable. SALSA provides 2 commands, which is available on all Vitual machine instances.

```
 - salsa-capability-set <capability ID> <value>
```

```
 - salsa-requirement-get <requirement ID>
```

In the commands above, the capability ID and requirement ID are defined in the application description (e.g. in TOSCA)

For specific case of getting IP, on the value is send to the node which needs and is stored in global env variable. The script can get the IP by:

```
. /etc/environment
IP_VALUE=$<ID_OF_SOURCE_NODE>_IP
or: IP_VALUE=$<ID_OF_RELATIONSHIP>_IP
```

The wired components are defined by CONNECTTO relationship in the application description (see Section 4)
The following figure depicts the process of wiring application components:

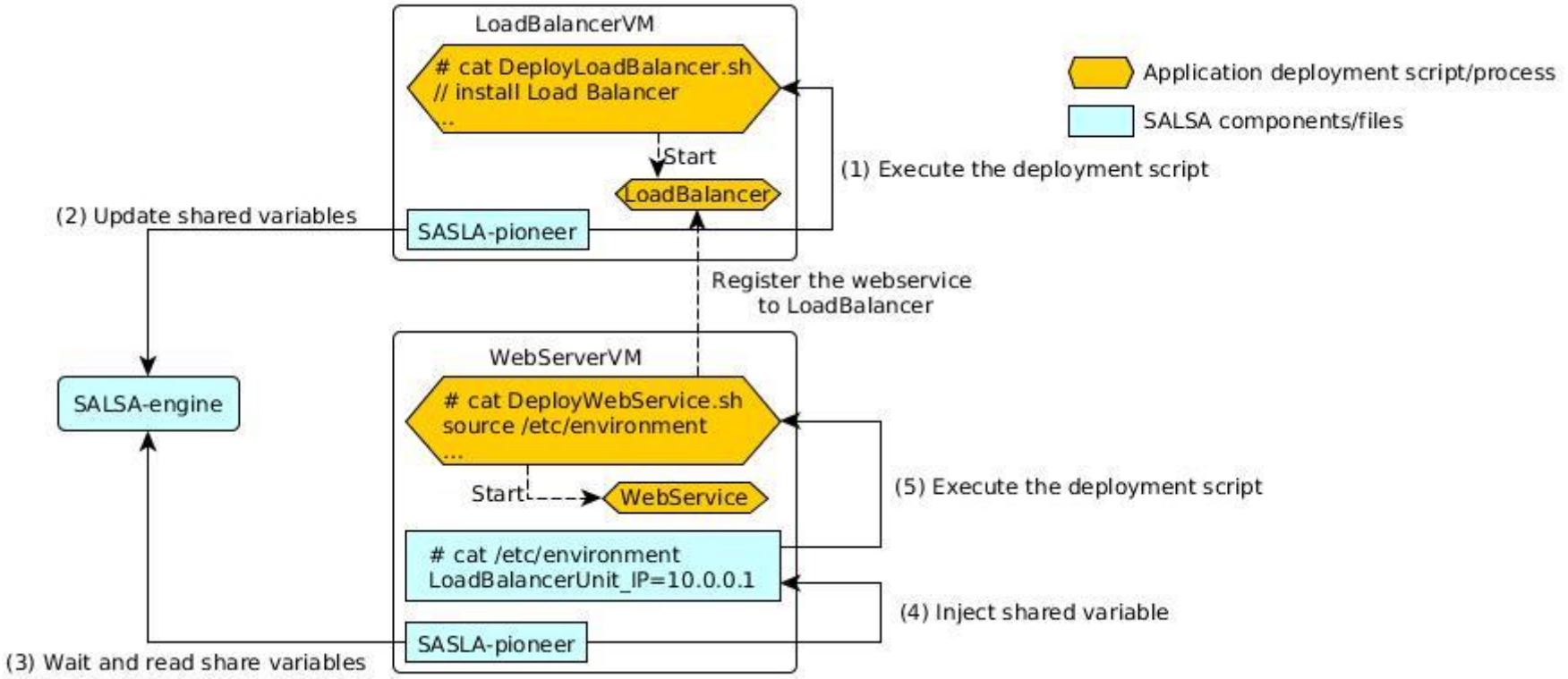


Figure 1- Example of wiring a WebService with a LoadBalancer

4. Application specification

4.1 TOSCA description

SALSA uses TOSCA for describing applications. An application is called a Tosca-Definition which contains a number of Service Template. In SALSA, a Service Template can be considered as an atomic service, which can be deployed separately.

A Service Template has **ONE** Topology Template which describes its abstract structure. Following the TOSCA concept, all application components

are considered as Tosca-Nodes, or Node Template. Node Templates have dependencies to others, and be described via Relationship Template.

For execute the deployment, each Tosca-Nodes will have a number of Deployment-Artifacts which point to the Artifact repositories of the Node materials. These templates can be defined by other Tosca-Elements such as Node-Types, Relationship-Types, Artifact-Types, etc.

4.2 Extended TOSCA for SALSA

As the property for NodeTemplate is Any, SALSA use some custom Properties for the description . These properties are define and put under the <tosca:Properties> tag of each NodeTemplate with specific type.

4.2.1 SALSA NodeTemplate type

The NodeTemplate types define which components can be place on where, enable SALSA to select right module to execute the configuration: using cloud connector for VM, pioneer for others. Also SALSA can parse the correct runtime information corresponding to the type. In the NodeTemplate, put an attribute as type="salsa:typeName". Currently, following type can be parsed:

```
 - salsa:software     => for the script-based artifacts
 - salsa:war          => describe a web application
 - salsa:docker       => default docker node
 - salsa:tomcat => default tomcat node
 - salsa:os           => default VM with Ubuntu
```

4.2.2 SALSA Artifact types

Artifact types define the artifact that is used for the deployment, enable SALSA can use appropriate mechanism to

In the ArtifactTemplate, put an attribute as type="salsa:typeName". Currently SALSA support:

```
 - salsa:sh           => bash script, using bash to execute the deployment script. The deployment script must
be finished.
 - salsa:contsh       => bash script which is called and run continuously. The running  process will be
managed by SALSA in order to stop latter.
 - salsa:war          => war file, deploy the war file to the tomcat webapp folder
 - salsa:chef         => using chef to install artifacts from chef community repository
 - salsa:chef-solo    => using chef-solo to install a local cookbook
 - salsa:apt          => using apt-get to install a local artifact
```

4.2.3 SALSA custom properties

SALSA parses custom properties which are wrapped in a MappingProperties structure. Currently, 2 type of properties can be parsed.

```
- salsa:os          => properties to describe VM
- salsa:action      => properties to define custom actions for application lifecycle management.
```

4.2.4 Virtual Machine Node Properties

Below is an example of the custom VM Node deployment information stated in <tosca:Properties>.

```
    <tosca:Properties>
         <MappingProperties>
              <MappingProperty type="salsa:os">
                   <property name="instanceType">000000960</property>
                   <property name="provider">dsg@openstack</property>
                   <property name="baseImage">8f1428ac-f239-42e0-ab35-137f6e234101</property>
                   <property name="packages">openjdk-7-jre</property>
                   </MappingProperty>
         </MappingProperties>
    </tosca:Properties>
```

The provider name is at the format: <site>@<connector>. This string is specified in the cloud configuration file, (default at ./cloudUserParameters.ini or /etc/cloudUserParameters.ini) with the parameters suitable for a cloud (e.g. user, password, endpoint). User need to define the instance type (the VM configuration) and the based image for the VM. The additional packages are optional, which describe the pre-configured environment inside the VM.

4.2.5 Custom actions properties

A NodeTemplate can have action property which map from a name with a system command.

```
         <tosca:Properties>
              <MappingProperties>
              <MappingProperty type="action">
                   <property name="start">sudo service myService start</property>
                   <property name="stop">sudo service myService stop</property>

                   <property name="restart">sudo service myService restart</property>
```

```
                    <property name="undeploy">./uninstall.sh</property>
                </MappingProperty>
            </MappingProperties>
            </tosca:Properties>
```

By default in SALSA, the "undeploy" action will firstly look for the "stop" action if available to execute to ensure the semantic of the application. If there is no "undeploy" script, SALSA will try to detect the artifact type and clean up the service by default.

4.2.6 Relationship types

The relationships represent the order of deployment between nodes. In a relationship, the source node will be deploy before the target node.

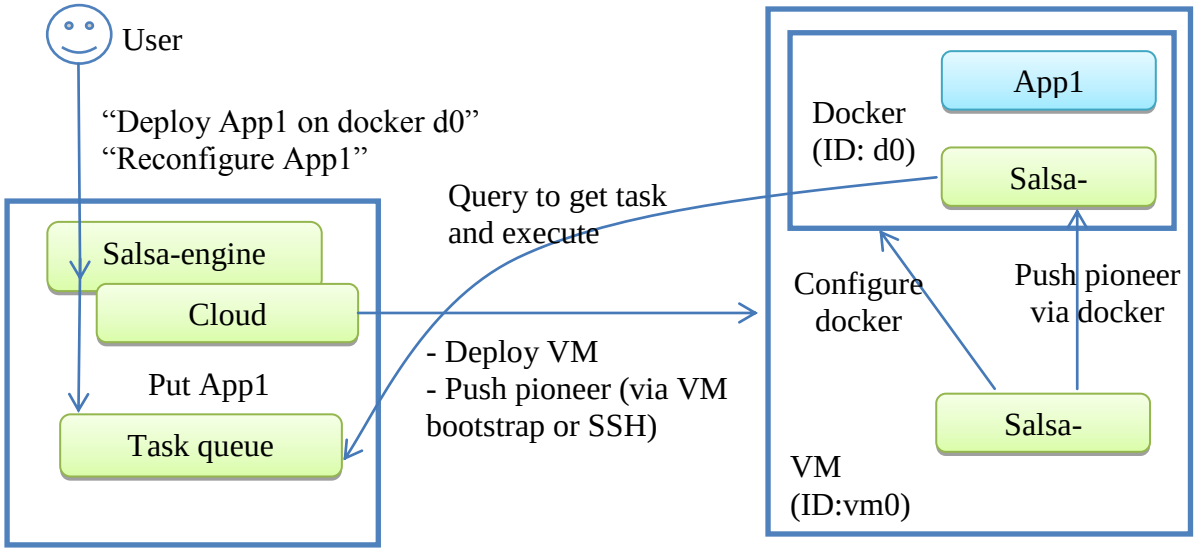
There are two types of relationship between nodes.

```
 - salsa:HOSTON => This represents one node will be deploy on top of other node. SALSA uses this to recognize
the stack of deployment. Currently, SALSA have VM stack and software stack, which show the software is
deployed on top of the VM.
 - salsa:CONNECTTO    => This represents the relationship between two software nodes. One node will waiting
for other nodes to be deployed, then get the IP of that node.
 - salsa:LOCAL        => Two node will be deployed on the same target. E.g. two software will be hosted by
single VM. This type is used only when "OS" node is not specified. This will be replaced by HOSTON
relationship when SALSA processes the TOSCA.
```

5. Configuration Q&A

This section is going to answer several questions which then guide users in using SALSA to satisfy the requirements of the complex configuration.
Question: How do users deploy something on a VM which are already running or when a container is already running, etc. How does SALSA work with multiple stacks?

Answer: Current architecture of SALSA enables to deploy and dynamic configure at the infrastructure and application level. On the infrastructure level, a lightweight client called salsa-pioneer is injected.



Whenever a salsa-pioneer is inside the VM or OS container (like docker) which refer to the infrastructure level, it will continuously queries for tasks that assigned to it (see figure above). There are following cases that the VM is already running:
- VM, container and application are defined in the same TOSCA at the beginning: they can be deployed/configured at any time as pioneers are always ready.
- A none-managed VM is bound to the service at runtime, which no pioneer is inside, we need to run an pioneer as well as the configuration file inside. This can be done by salsa-engine via RESTful which will try to use SSH to connect to the VM, or manually install/run the pioneer.

Question: Users may also have an agent to be pre-deployed in a gateway/VM which will download other code and do the deployment, how you SALSA plan to reuse it?

Answer: In order to use application-specific configuration tools, SALSA will need an adapter which implement the *salsa-pioneer instrument interface*. Please check:
https://github.com/tuwiendsg/SALSA/blob/master/salsa-core-pom/salsa-pioneer-vm/src/main/java/at/ac/tuwien/dsg/cloud/salsa/pioneer/instruments/InstrumentInterface.java
When recognize the type of the artifact, SALSA will forward the request and necessary information into the instrument module, which eventually invoke the external user-defined configuration module.

Question: How a new deployment algorithm can be implemented and configured - so i can have different algorithms (and i see which one is the best)

Answer: During the configuration, SALSA contains two point to apply algorithms

- Before the configuration: SALSA refines and enriches the application description (which is TOSCA). In current implementation, all of the algorithms are developed in following URL and will be called at the new orchestration.
  https://github.com/tuwiendsg/SALSA/tree/master/salsa-core-pom/salsa-engine/src/main/java/at/ac/tuwien/dsg/cloud/salsa/engine/smartdeployment
  TODO: Refactor the above package to support the attachment and selection of algorithms.
- During configuration: SALSA contains the algorithm for orchestration the configuration and for placing components at runtime which are implemented in the class:
  https://github.com/tuwiendsg/SALSA/blob/master/salsa-core-pom/salsa-engine/src/main/java/at/ac/tuwien/dsg/cloud/salsa/engine/impl/SalsaToscaDeployer.java
  Currently, the orchestration is design by parallel threads which can change to work-flow style or queue-based style. The placement is based on checking maximum instance number.
  TODO: Refactor to extract the algorithms outside of the routine.

Question: How to combine different deployment and configuration algorithms to have a new one.
Answer: Currently SALSA do not support multiple algorithms yet. We vision that when SALSA is able to deal with multiple algorithms, which apply for different part of the service (e.g. on service unit, on one topology or whole application), user can annotate the application specification to guide SALSA.
Question: how to have configuration processes - which are application/system-specific - who writes what and salsa will do what
Answer:
With the cloud providers, SALSA supports basic operations which implement the cloud-connector interface. The interface can be extended to support more capabilities from providers in general.
With the application level, SALSA support to defined custom actions can be invoked at runtime. In current implementation, we support user to execute custom script at runtime. By this, users provide a set of scripts to execute various configuration capabilities and specify with the service unit. At runtime, these actions can be executed via RESTful API with action name as input. For example, here is a piece of specification.

```
<tosca:Properties>
	<MappingProperties>
		<MappingProperty type="action">
			<property name="start">sudo service myService start</property>
			<property name="stop">sudo service myService stop</property>
			<property name="reconfig">./reconfig.sh</property>
			<property name="undeploy">./uninstall.sh</property>
		</MappingProperty>
	</MappingProperties>
</tosca:Properties>
```

Please refer to the UserGuide for more detail on “action” property.