Project Proposals Domino

Project Name:

- Proposed name for the project: Template Distribution Service (Domino)
- Proposed name for the repository: Domino

Project description:

- With NFV evolution, carriers need to manage not only transport resources, but also compute and storage resources spread over a large geographical footprint. To support variety of resource types and geographical spread, multiple resource orchestrators and controllers need to coexist. Yet, there is also a need to define end-to-end (e2e) services that need to be orchestrated over these heterogeneous set of resources and orchestrators/controllers consistently with a top-down approach. At the top of the hierarchy, service models and policies are combined together to define an overall service scenario in a single service template. Once a service template is generated, it must be processed, partitioned, translated, and distributed to various domains where local orchestrators and controllers manage their local resources properly and enforce the local policies. Thus, service template creates a domino effect on multiple control domains in terms of resource scheduling, lifecycle management, service scaling, etc.

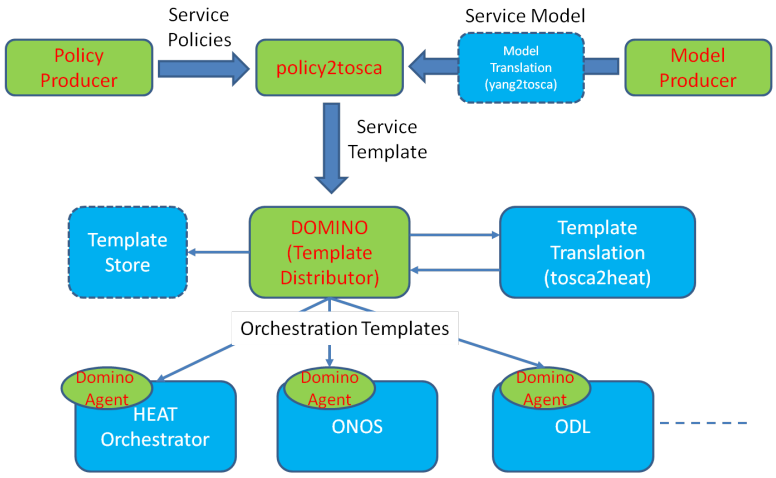

Figure 1: High Level Architecture

- Figure 1 depicts the main building blocks in the high level architecture. The first step is the generation of service policies and service models. These policies and models for a particular service can be generated by multiple (service and model) producers. Although we will define some specific policies and models for testing purposes, how to express policies and models is not the focus of the project. Instead, we will reuse existing tools to express policies and describe models (e.g., Copper, Model, Congress, yang/yaml models, NEMO, protocol buffers, etc.).
- The first main component and deliverable of the Domino project is the policy2tosca module, which parses the policy rules and service models to generate a single, coherent service template. Policy Producer, Model Producer, and policy2tosca module together constitute Template Producer.
- The second main component of the project is the Domino Template Distributor. Domino module serves Template Producers and Template Consumers. Its main services constitute mapping and translation (i.e., determine which portion of the service template is targeting which consumer and perform template conversion to the format understood by each consumer), serialization (i.e., to determine the workflow dependencies between and within Consumer domains), and distribution (i.e., transfer the converted templates to the corresponding consumers). To utilize Domino module, Template Producers and Template Consumers register with the Domino services. Domino module itself will utilize template translation services provided by other OPNFV and upstream projects (e.g., Parser Project).
- Template Consumers should be capable of supporting one or more resource orchestration templates. In the case of OpenStack Heat Orchestrator and Kubernetes, such support natively exists. In other cases where template support is not present, a template based orchestration agent must be deployed. For instance, in the case of SDN controllers, this may be done by adding a controller application that registers with Domino service from which it receives an orchestration template and utilizes SDN API or services to perform the local orchestration tasks. Note that in reference to the ETSI NFV architecture, the template producers and receivers can be part of EMS, NFVO, VNFM, or VIM depending on the workflow and hierarchy of controllers. OSS/BSS functions are the main source of service policies and models at the top of the service provisioning hierarchy [see IFA 009 V0.5.0 (2015-12)].

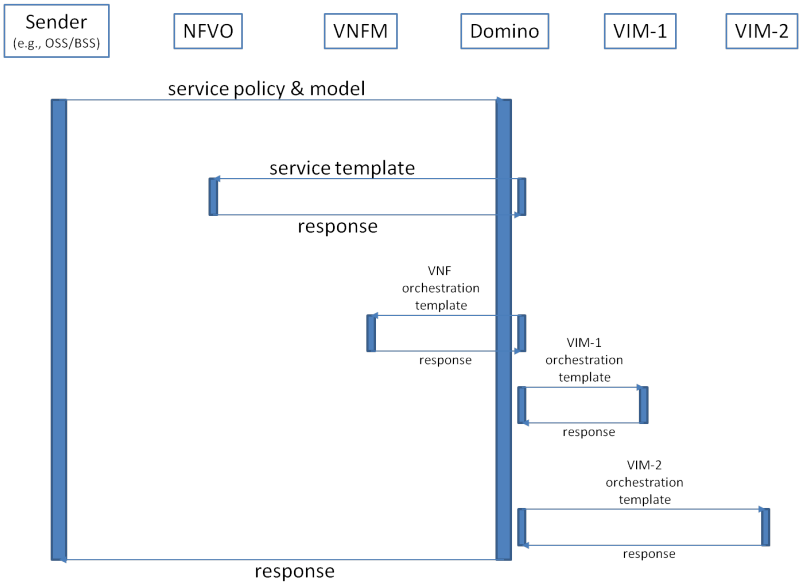

Figure 2: An example flow diagram for template distribution

- An example of successful template generation and distribution with respect to the MANO components defined in ETSI reference architecture is shown in Figure 2. OSS/BSS in this example generates service policies and models. Note that OSS/BSS can also directly generate a TOSCA service template. Domino creates a service template that combines policies and models in one service descriptor file. OSS/BSS can inject new policies, update existing policies, or update service models at a later time. Domino keeps track of these changes and reflect them into the service template, optionally versioning the previous service templates for the particular sender. Next, Domino passes the service template to NFV Orchestrator (NFVO). Domino also generates VNF orchestration templates and VIM orchestration templates to pass them to the respective VNF managers (VNFMs) and Virtual Infrastructure Managers (VIMs). If all the templates can be successfully generated, distributed, and accepted by the corresponding template receivers (i.e., NFVO, VNFMs, VIMs), then a success response is sent back to the policy/model generator. Note that the actual data flow can be different with first templates offered to each template receiver and upon receiving positive reply from each receiver, a subsequent commit or abandon message can be sent before sending a final response back to the service policy & model producers.

Use Case 1: Using templates for policy based delegation of resource orchestration

End to end service provisioning typically traverses multiple control domains, where different orchestrators/controllers manage the “local” resources in their respective domains. Locality here is in general refers to a relative proximity rather than absolute proximity. These orchestrators/controllers have several advantages, but the most prominent ones are (1) being closer to the resources whose states can be observed with higher fidelity and their actions can be controlled at finer granularity and (2) having domain specific knowledge to deliver the desired service behavior by configuring and programming the resources accordingly. In such a scenario, NFVO that performs service orchestration over multiple control domains can first generate a service template. Then, NFVO can use Domino Service for automatically generating the local orchestration templates and sending them to each local controller/orchestrator. These local templates can specify what actions should be taken autonomously by local controllers/orchestrators by prescribing constraints, conditions, workflow, pseudo-algorithms, expected service behavior, etc. For instance, NFVO can request an auto-scaling service from a target VIM, but also define various resource units (e.g., CPU size, memory size, VM flavor, etc.), workload conditions, and prescribe which resource unit to use to increase/reduce service capacity under what workload conditions

Use Case 2: Intent based API with domain specific rendering via templates

OPNFV architecture relies heavily on northbound APIs (NBIs) for resource management and orchestration. Keeping NBI simple and generic increases the usability and potentially prevents issues with backward compatibility. A simple API call to a particular VIM such as SDN controller or OpenStack controller can be interpreted in various ways if multiple options are available.

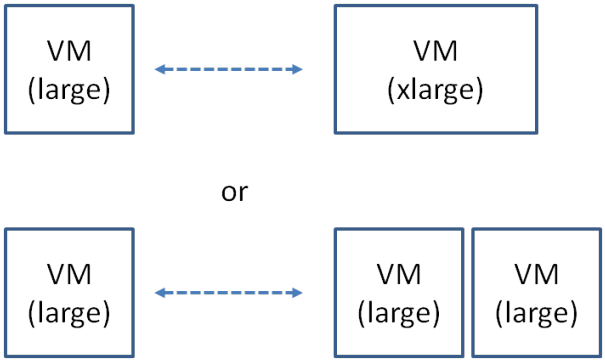

Figure 3: VNF Scaling Options

In Figure 3, an example for VNF scaling is provided. VNFM can simply make a call such as VNF.scale(scaling.factor), where the input parameter *scaling. factor _ determines the factor of capacity increase. If for instance _scaling.factor* is specified as two, then VIM is supposed to double the capacity after receiving the API call. The two options shown in the figure are both meaningful choices in the absence of additional clues about the service. In the first option, VIM changes the configuration of the VM by doubling its CPU, memory, and disk resources (i.e., it scales up the VM instance). This may be done by first pausing the VM, taking a snapshot, and then rebooting the snapshot image with a larger hardware configuration. This clearly interrupts the existing workloads and sessions on the scaled VNF. If the VIM had the capability of doubling the resources without any service interruption (e.g., it can add virtual hardware resources without requiring any image snapshot and reboot), scaling up a given VNF would have a different consequence on the actual services that utilize this particular VNF. Thus, even though the VNF is simply moved to a larger instance size, the service implications are quite different based on VIM capabilities. In the second option, VNF capacity is doubled by simply launching another VM with the same size (i.e., VNF is scaled out). For stateless network functions, this is relatively a straight forward and easy way of doubling the service capacity as long as the VMs do not have a shared bottleneck. If VNF has stateful implementation, state synchronization between the VMs to handle the existing sessions might be a major issue. The main point of this discussion is that without knowing the context of service and behavior/performance modeling of these VNFs going from one configuration to the other, an arbitrary decision by VIM can lead to poor performance. This undesired outcome could have been completely avoided if enough conditions and rules (i.e., prescriptions) were passed onto the VIM from VNFM.

Even for single VNF scaling, as the example shows, templating NBI calls is useful. In reality, service scenarios can be significantly more complex. For instance, VNF definition can be quite complex and can consist of many functions (e.g., vIMS, vEPC). Another popular example is services that are composed of VNF graphs or service chains. Managing the lifecycle and performance of such VNFs or services often require a series of well-orchestrated low level API calls and state maintenance. Thus, capturing all this orchestration only with a high level intent API in many use cases is not an option. Instead the consumer of an API should be able to describe the requested workflow in a resource orchestration template and provide this template to the producer of the API before actually utilizing the API.

A byproduct of supplementing an API definition with a service template is that the service template producer can change the run-time behavior of a given API without requiring any changes in NBI definitions. NBI consumer simply changes the service template associated with the API call. This is quite useful for instance when the resource models, VNF models, service models, or service policies change over time.

Scope:

- Develop new functionality for template based multi-site service orchestration and intent resolution for abstract, intent-based NBI.
- New Interface: Domino Template Distributor defines new interfaces for template consumers and producers to join the distribution and template translation services.
- New APIs will be defined for Parser, Template Producer, Template Consumer
- Specify testing and integration:
    - Testing will be done by simple Template Producer and Template Consumer agents. Template Producer will define simple service with simple VNF (packed in a single VM, complex service with complex VNF (micro services distributed over multiple interconnected VMs), and complex service with multiple VNFs connected through a VNFFG. We will change the service template via altering service model or policy rules, log the resource allocation decisions, verify that the final and transient states are in line with the expected behavior. We will also test API level calls to scale VNF, service, and VNFFG and test if the resource allocation/reallocation happens according to the service model.
- Debugging and Tracing:
    - Basic trace logging in info, debug, warning, and error modes will be supported.
- Unit/Integration Test plans: TBD
- New functionalities:
    - policy2tosca module: merges service policies and service models into one coherent TOSCA yaml template
    - Domino Template Distributor: creates a registry of service template producers and service template consumers; creates workflows out of the service template; performs required template partitioning, translation, & distribution tasks
    - Domino Agents: register as template producers and consumers
- In scope:
    - Service template generation
    - Policy specifications for various service use cases
    - Service model specifications for various service use cases
    - Service template parsing, partitioning into one or more orchestration templates, template translation to consumer accepted formats, template distribution
    - Template definitions for consumers, template agents for carrying out resource scheduling and orchestration
- Out of scope:
    - Defining a new modeling or policy language
- Extensions:
    - Project can be merged into or evolve into a multi-site service orchestration solution

Testability:

- Interoperability with OpenStack Heat and SDN Controllers are the main targets.
- Unit tests are to be provided with sample policy rules and service models, description of expected outcome, and validation against these expected outcomes.
- Local lab resources and Pharos Labs are to be used for integration and testing.

Documentation:

- Presentations:
    - opnfv_technical_discussion_-_domino_proposal.pptx
- API Docs (TBD)
- Functional block description (TBD)

Dependencies:

- Project has dependencies on Parser, OpenStack Heat, ONOSFW. Project has also dependency on SFC solutions (e.g., OpenStack neutron extensions, ODL SFC, ONOS SFC, etc.) at the API level.
- Although it does not have any particular dependency on OpenStack Tacker, the project has overlaps in the context of multi-site resource orchestration and parsing/translating/mapping TOSCA templates. Project has also potential overlaps with multi-site project.
- Open-O and Tacker are the main upstreams

(See Dependency Analysis for more detailed analysis)

Committers :

- Ulas Kozat (ulas.kozat@huawei.com)
- Prakash Ramchandran (prakash.ramchandran@huawei.com)
- Shiv Charan (shiv-charan.m-s@hpe.com)

Contributors :

- Artur Tyloch (artur.tyloc@canonical.com)
- Hai Liu (hai.liu@huawei.com)
- Linghui Zeng (linghui.zeng@huawei.com)
- Arthur Berezin (arthur@gigaspaces.com)

Planned deliverables:

- The project release package will include:
  - OPNFV test suite: service models, policy rules
  - OPNFV functions: new Parser modules (policy2tosca), Domino template distribution
  - Upstream ONOSFW, ODL: Template agent
  - Upstream Heat: HOT templates and new requirements documents
  - OPNFV requirements: lifecycle management artifacts for VNF and service scaling
- Dependencies
  - Parser project: policy2tosca module is to be implemented in Parser. Also tosca2heat and yang2tosca modules implemented in Parser projects will be utilized.
  - OpenStack Heat: project will use Heat as one of the template consumers
  - SDN Controllers: project will add SDN (e.g., ONOS and ODL) applications to act as template consumers

Proposed Release Schedule:

Release C plans:

- Specification for example service policy for VNF auto-scaling and service model
- Specification for example service policy for VNFFG auto-scaling and service model
- Implement template distribution to a single OpenStack Heat instance
- Implement template distribution to two OpenStack Heat instances
- Implement template distribution to a Heat instance and an ONOS instance
- Implement use case 1 (orchestration templating) and use case 2 (API templating)

Key Project Facts

Project Name: Template Distribution Service (Domino)
Repo name: Domino
Lifecycle State: Proposal
Primary Contact: Ulas Kozat (ulas.kozat@huawei.com)
Project Lead: Ulas Kozat (ulas.kozat@huawei.com)
Jira Project Name: Template Distribution Service
Jira Project Prefix: [Domino]
mailing list tag [Domino]
Committers:
Ulas Kozat (ulas.kozat@huawei.com)
Prakash Ramchandran (prakash.ramchandran@huawei.com)
Link to TSC approval: Example http://meetbot.opnfv.org/meetings/opnfv-meeting/2015/opnfv-meeting.2015-03-03-15.01.html
Link to approval of additional submitters: Example http://meetbot.opnfv.org/meetings/opnfv-meeting/2015/opnfv-meeting.2015-03-03-15.01.html