2020-02538 - Integration of reconfiguration academic tools to the DevOps community

Niveau de diplôme exigé: Bac + 5 ou équivalent
Autre diplôme apprécié: Thèse en informatique
Fonction: Cadre dirigeant
Niveau d'expérience souhaité: De 3 à 5 ans

A propos du centre ou de la direction fonctionnelle

The Inria Rennes - Bretagne Atlantique Centre is one of Inria's eight centres and has more than thirty research teams. The Inria Center is a major and recognized player in the field of digital sciences. It is at the heart of a rich R&D and innovation ecosystem: highly innovative SMEs, large industrial groups, competitiveness clusters, research and higher education players, laboratories of excellence, technological research institute, etc.

Contexte et atouts du poste

Configuring and deploying complex distributed software for heterogeneous distributed infrastructures is a non-trivial and technical task (e.g., containers and virtual machines provisioning, configuration files, service registration, etc.). A deployment, because of its error-prone and complex nature, needs to be automated and needs programming support. Furthermore, both distributed infrastructures and software systems are nowadays evolving towards more dynamic behaviors. For instance, in Fog and Edge infrastructures, many non-amortized computational devices frequently enter or leave the network because of their mobility or because of failures. Thus, network interruptions and faults are becoming usual and the norm. Another example comes with IoT and associated Smart systems (e.g., smart cities, industry 4.0) creating many different services as micro-services that rotate through time according to dynamic information from sensors. This high dynamics of systems and applications leads to dynamic changes (i.e., adaptations) of configuration that, again and even more, require automation and programming support. Different levels of adaptation could be considered, such as the behavioral level (or functional level) and the management level (or service) level. We consider the management level to handle the passage from one configuration to another. This is called reconfiguration in the literature.

Most contributions on reconfiguration rely on principles of Component-Based Software Engineering (CBSE) that help in structuring the code in distinct sub-functionalities that can communicate and make a clear separation of concerns between the domain-specific code and the assembly of black-boxes of codes (i.e., components) that expose their interfaces. Thus, CBSE makes easier distributed software programming and facilitates the dynamic choice of components that participate in the assembly (reconfiguration). For instance, CBSE concepts are applied in micro-service- and service-based applications. One of the key concerns of reconfiguration at the management level is to be able to handle the lifecycle of each component or service. Indeed, when considering complex distributed software systems, adding or removing components and their associated connections is not a trivial task. Most of the time, this cannot be performed anytime because of current data manipulation or ongoing transactions and communications (i.e., stateful). Instead, each component has to be placed in a safe state of its life-cycle such as “stopped” or “suspended” for instance (i.e., quiescence).

Most contributions on reconfiguration consider a fixed coarse-grain lifecycle common to every component. For instance, the “start” and “stop” is a very basic 2-states life-cycle proposed by many models. Other models handle more states in the life-cycle, Deployment for instance with six states: install, configure, start, manage, stop, uninstall, and uninstall. Modeling the same way every life-cycle has the advantage of facilitating its management and writing. However, this choice reduces the flexibility of life-cycle management and their customization and optimization. For instance, it has been shown that a start/stop sequence is used to avoid errors and inconsistencies, leads to stopping almost the entire system because of the contamination of the “stop” to every communicating components. In other words, such a model leads to service interruption. Another example is the impossibility to expose parallel patterns in life-cycle management and reconfiguration, thus limiting its efficiency.

Researchers of the STACK project team are contributing to enhancing the expressiveness of lifecycle management and reconfiguration compared to the related work. In particular, Madeus for deployment (initial configuration) and Concerto for reconfiguration, handle programmable, and fine-grained life-cycle modeling that automatically exposes and handle parallelism and asynchronism. These contributions have been validated on real use-cases (e.g., OpenStack) and have shown high-efficiency improvements compared to its closest related work Aeonos and Ansible. Furthermore, both Madeus and Concerto have been implemented in Python and are available online under a GPL license.

The intuition under Madeus and Concerto is to model each component lifecycle by a control component that each place is a milestone in the deployment or reconfiguration process, and each transition is a set of actions to perform to reach the next place (e.g., apt-get). In addition to this structure, ports are used to model interfaces with other control components, providing services or data, and requiring services or data from other components. The assembly of components consists of connecting instances of control components with compatible ports. This assembly is dynamic and can be adapted thanks to the Concerto reconfiguration language that offers ways to reach safe states for each component, start or delete connections and components. Madeus and Concerto are executable models that safely orchestrate services and reconfigurations. Though, if Madeus and Concerto have enhanced the flexibility and efficiency of deployments and reconfigurations compared to the related work, this flexibility is also responsible for an additional complexity for the user. For this reason, the team currently working on the integration of higher abstraction level mechanisms to ease their use and their integration in existing well known DevOps tools. Because of the continuous and quick evolution of the DevOps community technical support is needed for developments, technical watches, and experiments.

Mission confiée

DevOps community integration

1. Integration of Madeus with Ansible. We plan to work on the integration of Madeus with Ansible, a very well known infrastructure management tool provided by RedHat. For the
DevOps who are used to shell scripts, Ansible has become a popular configuration management tool since it relies on a simple syntax written in YAML and does not require agents on administrated servers. System tasks are managed using only SSH and Python which are commonly installed on every Linux distribution. Ansible features software engineering aspects such as separation of concerns by defining the notion of roles, which can be seen as control components. Each role contains a set of files that describe a sequence of tasks. To define a composition, a specific file called an Ansible playbook is used for mapping the desired roles to the groups of nodes they will be applied to. Those groups of nodes are defined in a separate file called the inventory. The idea is, on one hand, to add a very light descriptive language to Ansible to declare service dependencies between the different tasks of a given role, and between tasks of different roles. Data dependencies, on the other hand, would be automatically detected by analyzing Jinja2 templates of the Ansible playbook. Indeed, Jinja2 is used in Ansible to transfer information from one role to another. From this information a Madeus deployment will be generated and will directly call sub-roles of the initial playbook that will be split automatically. As a result, from any existing well-formed Ansible playbooks, and by writing a simple additional description of dependencies, efficiency and safety of deployments and infrastructure procedures will be guaranteed. Furthermore, from the Ansible repository Galaxy, or from GitHub, learning techniques could be used to classify the most usual deployment dependencies such that even the additional dependency file could be avoided. We already have two real use-cases implemented from an initial Ansible playbook that could be re-used in experiments: The OpenStack deployment by using Kolla-Ansible, and the deployment of a Galera cluster of MySQL by using Juju.

2. Integration of Concerto in Juju. If Ansible suites well for integration of Madeus, it is less adapted to reconfiguration capabilities such as the ones offered by Concerto. For this reason, we plan to integrate Concerto concepts to orchestration tools such as Juju or OpenStack Heat or Kubernetes. After a detailed related work study, it seems that the integration will be easier to Juju that follows a line close to Concerto with the concepts of charms (i.e., components), hooks (i.e., tasks), and relations (i.e., dependencies). Furthermore, Juju already offers ways to monitor and manage deployed applications. Finally, learning techniques could also be studied from the Chams store to infer dependencies from existing charms.

3. Technical watch and possible relations with Nix/NixOps. Nix has been published in 2006 and offers a functional and immutable way of handling computing facilities. In Nix (and later NixOS) any change to the Linux system creates a new hashed version of the system while keeping untouched the older version thus enhancing rollbacks, reproducibility, and safety for system administrators. NixOps uses the same concepts of immutability but at a cluster level. If the immutability vision goes against the in-place reconfiguration vision, we think that these visions could be combined to find a good trade-off between flexibility, speed, optimization (i.e., in-place), and easy rollback and reproducibility (i.e., immutability). A technical watch and basic manipulations on NixOps will be asked to the engineer.

**Experiments and use-cases**

1. Experimental setup. To evaluate the capability of software systems to resist unexpected faults in a short period of time, thanks to the reconfiguration mechanisms, the engineer will be asked to design a complex experimental setup on Grid’5000 that uses for instance ChaosMonkey developed by Netflix. Ideally, this experimental setup will be written by using Madeus or Concerto itself at a meta-level (to get better efficiency), or by using Grid’5000 tools such as EnosLib. Furthermore, the generalization of this experiment will be studied for adoption by researchers working on decentralized and fault-tolerant systems and infrastructures.

2. Reconfiguration of OpenStack. The deployment of OpenStack has already been studied and written with Madeus. This long and tedious work can be leveraged to build reconfiguration use-cases. Multiple reconfiguration cases will be studied: the reconfiguration of OpenStack from centralized to decentralized databases; the reconfiguration of a multi-region deployment of OpenStack; the rolling upgrade of OpenStack.

3. Reconfiguration in the DAO project. The DAO project is hosted at the Artic University of Norway in Tromsø 9. Nowadays, biological and climate researchers carry wild life sensors, cameras and other observation devices into the field, manually configure the devices while on the abroad places. They then fetch the collected data several months later, by hand. This approach does not scale. Deployment, configuration, observations, reporting of devices and data must be automated. The Distributed Arctic Observatory (DAO) project proposes a hardware and software solution to these problematics. A DAO observation unit (DO) is a configurable computer node along with a set of sensors. These units will autonomously monitor themselves and the environment, run software layers able to configure and run new applications, synchronize data and finally gather data to centralized servers to execute analysis processes. The engineer will exchange with the researchers of the DAO project to identify and design reconfiguration use-cases.

**Principales activités**

**Roadmap and expected profile**

**Year 1:**
- Understanding of Concerto, design of basic examples;
- design of a new real use-case for Concerto;
- developments for the integration of Madeus to Ansible;
- experiments on Grid’5000;
- technical watch Juju and NixOps.

**Year 2:**
- Design of a second real use-case for Concerto;
- integration of Concerto in Juju;
- experimental setup with ChaosMonkey;
- experiments on Grid’5000.

**Avantages**
- Subsidised catering service
- Partially-reimbursed public transport

**Rémunération**
PhD student: monthly gross salary amounting to 1982 euros for the first and second years and 2085 euros for the third year.