ICDO: Integrated Cloud-based Development Tool for DevOps

Farshad Ahmadighohandizi and Kari Systä

Department of Pervasive Computing, Tampere University of Technology
Korkeakoulunkatu 10, FI-33720 Tampere, Finland
Email: firstname.lastname@tut.fi

Abstract. This research is based on three drivers. Firstly, software development and deployment cycles are getting shorter and require automatic building and deployment processes. Secondly, elastic clouds are available for both hosting and development of applications. Thirdly, the increasingly popular DevOps introduces new organizational and business culture. This paper presents a research prototype and demonstrator of an integrated development tool. The tool is cloud based and thus accessible from any Web-enabled terminal. Automation is maximized so that deployment cycles can be as fast as possible. Since the aim is to use cloud resources as a utility in a flexible manner, cloud brokering – i.e. finding the most suitable provider – is included in the system. The contributions of the paper include: an idea of a new kind of DevOps tool, description on how it can be implemented on top of standard components and implications to software development processes.

Keywords: Continuous deployment, Cloud brokerage, Cloud federation, DevOps, Software development, EASI-CLOUDS project

1 Introduction

Continuous Integration (CI) [2] is used in many organizations and has shown its power in improving the efficiency of software development. Continuous Delivery and Deployment [3] expand CI with automatic delivery and deployment and are often attached with mechanisms to immediate collection of data of user behavior and fast reaction to that feedback [12]. DevOps [4] builds on CI and CD but also includes organizational and cultural aspects – especially it removes the wall between developers and operation personnel. There are several tools for continuous integration with automatic building and testing – Jenkins[1] is perhaps the most well-known. For automatic deployment and management of operations separate tools like Puppet[2] and Rundeck[3] can be used. However, the assumption of these tools, i.e., the deployment is separated from the development, maintains the wall that DevOps wants to remove.

[1] `https://jenkins-ci.org/`
[2] `https://puppetlabs.com`
[3] `https://rundeck.com`

Cloud technologies provide new opportunities for both development and operation. With cloud-based tools, developers get scalable and pervasive aid for all actions like coding, building and testing. Cloud-based development tools like CoRED [7] and Cloud9[4] allow developers to access the tool if they just have access to network and browser. Therefore, developers do not need to install any software on their local machines, and the computing capacity for the tools can be elastically allocated according to current needs.

When applications and services are deployed into the cloud, several factors need to be considered. Enough resources need to be allocated for the hosting of the application to ensure the required service level under assumed usage load. Various legal and trust-related concerns can also affect the decision of the physical and virtual location of the service. Finally, the cost of running the service needs to be minimized.

ITEA2 project EASI-CLOUDS[5] has developed technologies for allocation and management of cloud resources based on SLA (Service Level Agreement) and enables a market place for cloud resources through brokering and federation. In other words, EASI-CLOUDS targets both cloud providers and consumers. It helps cloud providers utilize each other's resources through federation and brokerage. It also assists cloud consumers to adopt to multi-cloud architecture and select cloud services by simply requesting them by using manifests.

In this paper, we show how development tools like IDE and continuous integration systems can be combined with brokering-based deployment and service operation. The same integrated tool can be used for both development and management of operations. In other words, use of the cloud-based tool starts with code editor for programming, continues with finding the suitable cloud resources for application hosting, goes on with automatic building and testing, and finally ends with the application deployment. In addition, the deployed application can be managed via the same tool.

The first step of thus work was done as a demonstrator for EASI-CLOUDS project for communicating the results to the project consortium and to external stakeholders. In addition, we conducted research on how cloud-based development tools and brokered cloud resources complement each other and how the ideas of DevOps integrate with the ideas of EASI-CLOUDS.

The remainder of this paper is organized as follows. Section 2 explains the fundamentals our research is based on. Section 3 describes the integrated tool, its usage and gives some of the implementation details of our prototype. Section 4 analyses the prototype and its implications. Section 5 presents comparison of our research to related work. Finally, Section 6 concludes and states future work.

[4] `https://c9.io/`
[5] `http://easi-clouds.eu/`

2 Technology background

2.1 Cloud-based IDE

Department of Pervasive Computing at Tampere University of Technology has developed a cloud-based IDE. Its different versions include CoRED which mostly focuses on collaborative aspects of the development and MIDEaaS which adds a visual UI editor to CoRED [7]. In our research, we chose CoRED as there was no need for a visual UI editor. Important features of CoRED include:

- *Could-based.* Running in the cloud, CoRED frees developers from problems like installation, upgrades and maintenance. To access CoRED and to start development only an up-to-date browser and a network connection are needed.
- *Real-time collaboration on common programming tasks.* This is similar to collaboration on document writing with Google Docs[6]. CoRED provides developers with various means to collaborate. Collaboration and communication of developers let them know what they are doing as a team and can prevent redundancy or even conflicting work. The different collaboration features of CoRED are explained in more detail in [6] and [10].

More information about CoRED and MIDEaaS is available through `http://cored.cs.tut.fi`. An open source version is also available.

2.2 Cloud Brokerage and EASI-CLOUDS

As more companies adopt cloud computing as a key enabler for their business, more cloud service providers emerge. This variety of cloud providers has generated new challenges for both providers and their end users. From the users' perspective, finding the appropriate cloud provider that fills the requirements on issues like security, billing, and supported technologies from number of possible providers is a challenging task.

EASI-CLOUDS project proposes brokerage and federation solutions for the above challenges. A broker is an entity that manages the use, performance and delivery of cloud services. It also negotiates relationships between cloud providers and cloud consumers [8]. Cloud federation provides cloud provider with a mean to send a cloud request to multiple cloud providers as if they were a single cloud provider [5]. EASI-CLOUDS project developed both technologies. The aim of the project was to build a system that assists in usage of offerings from several cloud providers and acts as an intermediary between providers and their users. It helps consumers to find and select the most suitable cloud resource among several providers, and to provision it in a unified mechanism by making providers interoperable.

EASI-CLOUDS project reuses components from an earlier project CompatibleOne[7]. CompatibleOne introduces a common description model for Cloud

[6] `https://www.google.com/docs/about/`
[7] `http://www.compatibleone.org`

resources and a platform which executes provisioning tasks for these resources [14]. The common description model is called CORDS (CompatibleOne Resource Description Model) and the execution platform is called ACCORDS (Advanced Capabilities for CORDS).

2.3 Provisioning Heterogeneous PaaS Resources

Different cloud providers may base their offerings to different PaaS technologies (e.g., Cloud Foundry[8], OpenShift[9], Jelastic[10], etc.) for provisioning their resources. Switching from a provider using a PaaS solution to another one with a different solution takes much effort and learning. So, a PaaS-independent solution to enable use of heterogeneous PaaS resources is required.

CompatibleOne Application Platform Service (COAPS) [11] offers this PaaS-independent solution. It provides a unified model to describe PaaS resources regardless of their hosting providers. It also proposes a generic API which is an abstraction layer for heterogeneous PaaS providers. COAPS API is shown in Figure 1.

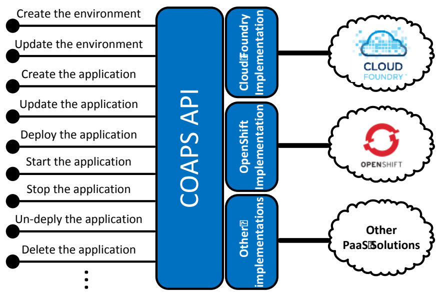


Fig. 1. COAPS API is an intermediary for management of heterogeneous PaaS resources [1]

The unified description model proposed by COAPS divides PaaS resources into two parts: *Application* resource and *Environment* resource which prepares the required environment to host the application. The description language for resources in COAPS is similar to CORDS description model in ACCORDS (see Subsection 2.2). This similarity facilitates their communication as they are designed to be used together.

For a cloud application to be deployed to a PaaS through COAPS two steps are needed as shown in Figure 2. Firstly, environment and application resources should be provisioned on the target PaaS by calling related COAPS APIs (*create the environment* and *create the application* APIs respectively). In this step,

[8] `https://www.cloudfoundry.org/`
[9] `https://www.openshift.com/`
[10] `https://jelastic.com/`

COAPS also associates application and environment resources with unique IDs and sends these IDs back to the caller for further references. In the second step, actual deployment and starting of the application is done with *deploy the application* and *start the application* methods in COAPS API.

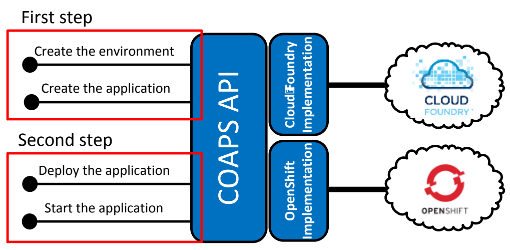
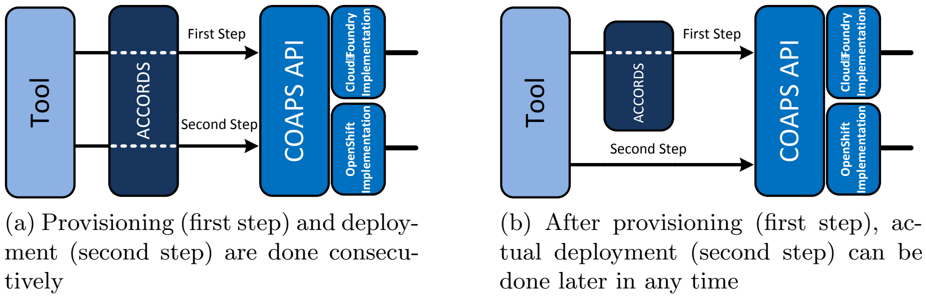


Fig. 2. Steps to host and run an application through COAPS

2.4 Application Deployment methods in EASI-CLOUDS

To deploy an application to a multi-provider PaaS architecture using EASI-CLOUDS components, two methods are proposed as follows:



(a) Provisioning (first step) and deployment (second step) are done consecutively

(b) After provisioning (first step), actual deployment (second step) can be done later in any time

Fig. 3. Immediate (a) and deferred (b) deployment methods

Immediate Deployment. In this method, all requests for brokering, provisioning, and deployment are sent to ACCORDS. ACCORDS first finds the most appropriate PaaS provider through the brokerage. Then for the provisioning cloud resources (step 1 in Figure 2) and the deployment and running the application (step 2 in Figure 2), it interacts with COAPS. In other words, ACCORDS performs two steps shown in Figure 2 one after another as an atomic operation. Figure 3(a) shows how in immediate deployment method a tool should interact with ACCORDS to host and run an application in the selected PaaS.

Deferred Deployment. In his method, the only request for brokerage is sent to ACCORDS. After brokerage, ACCORDS interacts with COAPS of the selected

PaaS to create environment and application resources (step 1 in Figure 2). Actual deployment of the cloud application (step 2 in Figure 2) can be done later in any time by direct interaction with COAPS. Figure 3(b) illustrates how in deferred deployment method a tool should communicate with ACCORDS and COAPS to host and run an application.

3 The integrated development tool

3.1 Same tool for development, deployment and operation

CoRED is a cloud-based IDE and its original implementation already included a simple deployment that could be done after building. In this research, we added a programmable CI system that can control more complex build and test procedures. In addition, we integrated deployment to manageable PaaS platform and cloud brokering to the system. Thus, the developer can use the same tool to develop, build, allocate cloud resources and to deploy the application.

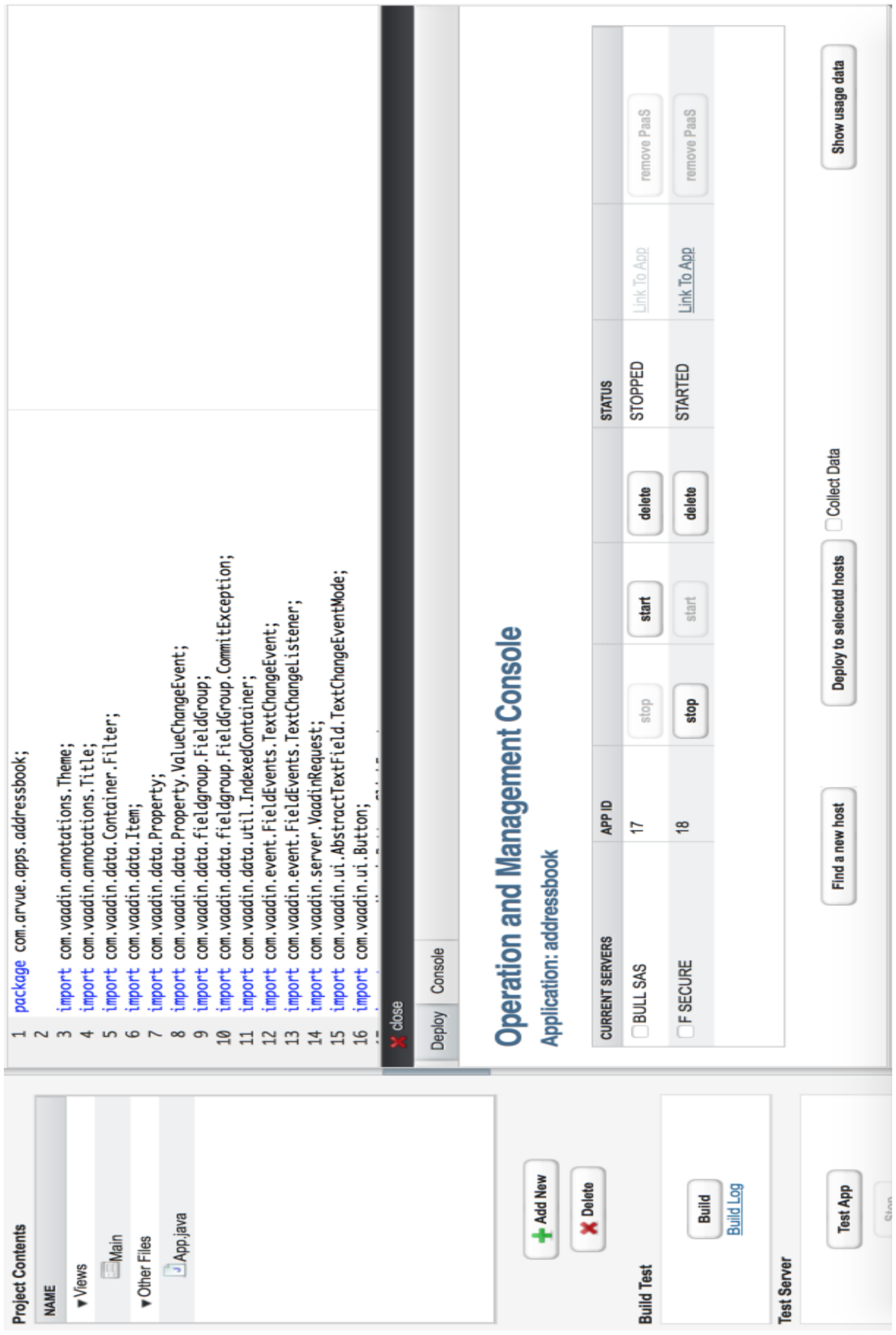
Our tool demonstrates DevOps by bringing development and operations into one integrated tool. Users of our tool are provided with an *operation and management* console. A snapshot of this console is shown at the bottom part of Figure 4. Users can start, stop, and delete the deployed applications via this console. There is also a hyperlink to each application so that the user can easily run the deployed applications. Figure 4 shows that one application named Addressbook has been deployed to two PaaS providers (F-Secure[11] and BULL SAS[12])[13]. It also shows that current status of the application deployed to BULL SAS is *stopped*, whereas the state of the application hosted at F-secure is *started*.

3.2 Integrated User Interface

Several existing tools are used as components of our prototype. Each of them has initially designed to perform specific tasks such as editing code, brokering, building, and deployment. The goal of our research is to provide a uniform interface to the user. Thus, the user does not need to switch between several tools.

Figure 4 shows a snapshot of our prototype. The left column is for management of the project files, the top right part is for editing code, and the bottom part has two tabs. Tab named *deploy* is for brokerage, initiation of the build and deployment, and management of operations. *Console* tab displays logs and possible errors appeared in build and deployment phase as illustrated in Figure 5. The left panel of console tab shows a high-level view (i.e., pushing code to the repository, build, and deployment), the middle panel shows the build log, and the right panel shows the deployment log.

[11] `https://www.f-secure.com`
[12] `http://www.bull.com/`
[13] F-secure and Bull were two partners of EASI-CLOUDS project



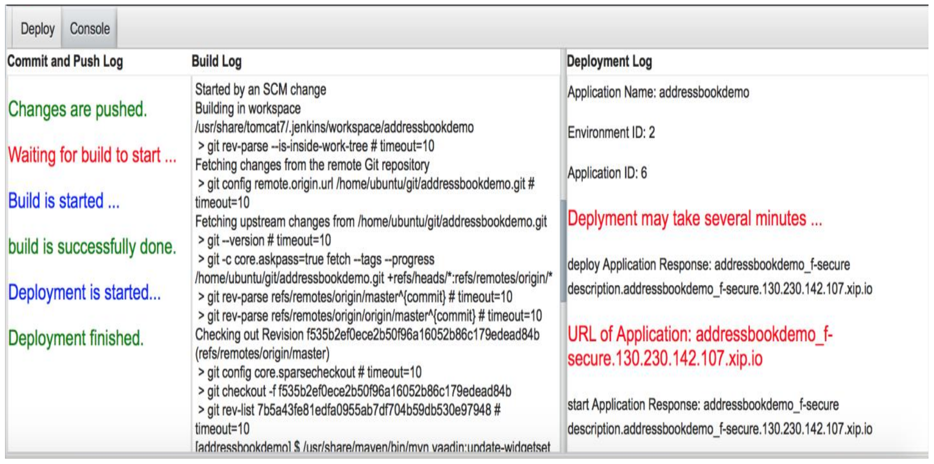
```
package com.arvue.apps.addressbook;

import com.vaadin.annotations.Theme;
import com.vaadin.annotations.Title;
import com.vaadin.data.Container.Filter;
import com.vaadin.data.Item;
import com.vaadin.data.Property;
import com.vaadin.data.Property.ValueChangeEvent;
import com.vaadin.data.fieldgroup.FieldGroup;
import com.vaadin.data.fieldgroup.FieldGroup.CommitException;
import com.vaadin.data.util.IndexedContainer;
import com.vaadin.event.FieldEvents.TextChangeEvent;
import com.vaadin.event.FieldEvents.TextChangeListener;
import com.vaadin.server.VaadinRequest;
import com.vaadin.ui.AbstractTextField.TextChangeEventMode;
import com.vaadin.ui.Button;
```

Operation and Management Console

Application: addressbook

CURRENT SERVERS	APP ID				STATUS		
BULL SAS	17	stop	start	delete	STOPPED	Link To App	remove PaaS
F SECURE	18	stop	start	delete	STARTED	Link To App	remove PaaS

Fig. 4. A snapshot of our demonstrator



Commit and Push Log

```
Changes are pushed.
Waiting for build to start ...
Build is started ...
build is successfully done.
Deployment is started...
Deployment finished.
```

Build Log

```
Started by an SCM change
Building in workspace
/usr/share/tomcat7/.jenkins/workspace/addressbookdemo
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url /home/ubuntu/git/addressbookdemo.git #
timeout=10
Fetching upstream changes from /home/ubuntu/git/addressbookdemo.git
 > git --version # timeout=10
 > git -c core.askpass=true fetch --tags --progress
/home/ubuntu/git/addressbookdemo.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision f535b2ef0ece2b50f96a16052b86c179edead84b
(refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f f535b2ef0ece2b50f96a16052b86c179edead84b
 > git rev-list 7b5a43fe81edfa0955ab7df704b59db530e97948 #
timeout=10
[addressbookdemo] $ /usr/share/maven/bin/mvn vaadin:update-widgetset
```

Deployment Log

```
Application Name: addressbookdemo

Environment ID: 2

Application ID: 6

Deplyment may take several minutes ...

deploy Application Response: addressbookdemo_f-secure
description.addressbookdemo_f-secure.130.230.142.107.xip.io

URL of Application: addressbookdemo_f-
secure.130.230.142.107.xip.io

start Application Response: addressbookdemo_f-secure
description.addressbookdemo_f-secure.130.230.142.107.xip.io
```

Fig. 5. Build and deployment steps together with their log is reported to the user

3.3 Detailed workflow and implementation

This section walks through the operation of our cloud-based development tool and describes the implementation details of the proposed solution. Figure 6 shows the components of our tool. These components are referenced and their roles are explained in the rest of this section.

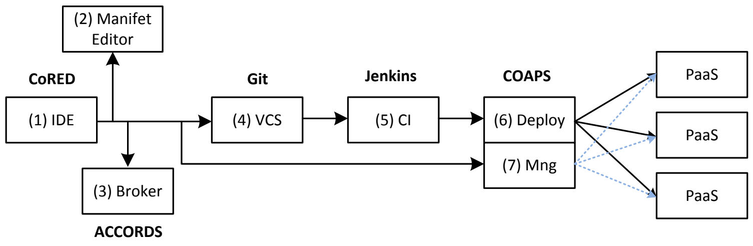


Fig. 6. Cloud-based development tool and our demonstrator

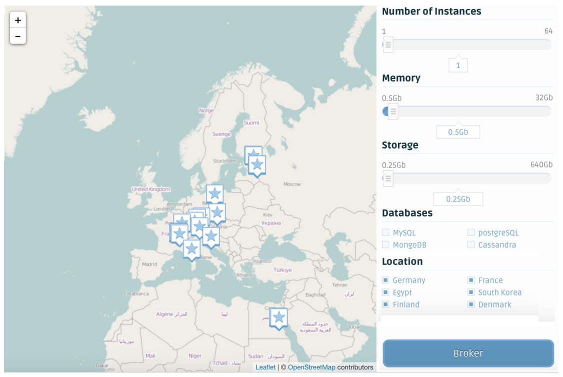
As the editor, (1) in Figure 6, we used CoRED (Subsection 2.1). The initial version of CoRED only supported Vaadin[14] applications. Although support for a range of new languages has later been added to CoRED, our research has been done on developing Vaadin applications in Java programming language.

To deploy an application into a multi-cloud architecture we integrated CoRED with cloud brokering solution, (3) in Figure 6, of EASI-CLOUDS (see Subsection 2.2). We chose the deferred deployment method (see Subsection 2.4) to enable

[14] `https://vaadin.com/`

users to manage the operations of the deployed applications. More details about this choice are given in Subsection 4.2.

In the deferred method, brokering is done before deployment. In the user interface shown in Figure 4, brokerage is initiated by pressing 'Find a new host' button. Then users will then be provided with a graphical tool, (2) in Figure 6, to define requirements for the needed cloud resources and to adjust quantities like the amount of memory and disk space, required database solution, the number of instances, and desired geographical location of PaaS provider. These settings will be sent to the brokerage system that will choose the most suitable PaaS provider. Although we tested usage of ACCORDS for brokering, it was replaced by a mock-up in the final implementation since the current ACCORDS do not have proper support for deferred brokering.



Number of Instances
1 64
1

Memory
0.5Gb 32Gb
0.5Gb

Storage
0.25Gb 640Gb
0.25Gb

Databases
☐ MySQL ☐ postgreSQL
☐ MongoDB ☐ Cassandra

Location
☑ Germany ☑ France
☑ Egypt ☑ South Korea
☑ Finland ☑ Denmark

Fig. 7. Graphical tool with which users can specify required cloud resources

The automated building and deployment pipeline is started by pressing *Deploy to the selected hosts* button (see Figure 4) from the IDE. In the first step of automated pipeline the edited source files of the project are committed and pushed to source code repository, (4) in 6, that is based on Git[15] revision management system. Git repository informs related Jenkins task about changes. Consequently, the Jenkins task fetches the updated project and invokes build environment, (5) in 6, using Maven[16] technology. Maven manages the building process and creates the final WAR file. Each of these steps and examples of their corresponding logs are shown to the user as shown in Figure 5.

One important reason behind our choice of Jenkins is its plugin support. A Jenkins task can be extended by implementing extension points defined in different stages of its life cycle. For example, to perform deployment, post build extension point was extended to use COAPS for deployment – see (6) in Figure 6.

[15] `https://git-scm.com`
[16] `https://maven.apache.org`

Users are also provided with an *operation and management* console, (7) in Figure 6, through which operations of the deployed applications can be managed (see Subsection 3.1).

Our tool is running on TUT infrastructure which is based on OpenStack[17]. One virtual machine hosts CoRED and COAPS, whereas Jenkins CI server and Git server runs on another virtual machine. In addition, one PaaS for hosting applications is running Cloud Foundry on a separate virtual machine.

4 Analysis

The resulting system has been tested with a few applications during the project. Different development phases of the tool have been demonstrated to various stakeholders. Based on experiments with those examples we can say that the proposed system is possible and the demonstrator communicates the idea. This section summarizes our key findings regarding development practices and use of DevOps together with cloud brokering. In addition, we report the implementation related issues that need to be solved before commercial exploitation of the ideas.

4.1 Development process and practices

The CoRED code editor has been designed for collaborative coding where several developers can participate in changing the same file simultaneously. However, many organizations use revision control systems (RCS) to manage the collaboration since developers check the code in when they think that others should see it. In our demo the check-in always triggers the building process, but since we use Git we could have implemented traditional RCS-controlled collaboration by using two separate branches, one for continuous integration and the other for sharing code between developers. The different nature of collaboration with cloud-based IDE and with RCS have been discussed in [9]. In that paper, the authors note that with cloud-based editors three new paradigms are possible. Here we discuss these paradigms in the light of this research.

1. **Real-time collaboration between developers instead of revision control based collaboration.** This has also been the default assumption in our case, i.e., tools help collaboration instead of preventing simultaneous work on the same file. Our domain and assumption of fast release cycles also emphasize the need for collaborative coding since applications are developed as a series of feature increments and each developer develops a specific feature instead of a module or file as in traditional software development.
2. **Automatically created versions.** Revision control has actually several purposes: control of the collaboration, provide ways to backtrack to old version and to trigger automated building and deployment pipeline. The research in [9] notes that modern IDEs have a compiler-like functionality as

[17] `http://www.openstack.org`

a background system for syntax checking and that system could recognize coherent systems to be inserted in the revision database automatically. Although we do not see a need for automatically created deployable version, the amount and nature of automatic quality check integrated into IDE could be more extensive and elastic cloud resources could be used for implementation of such systems.

3. **Co-operation for a jointly created version or solving a conflict.** This would not be needed in our current demo set-up but would be very important use case if components are imported from other projects – especially when the project is been created.

4.2 DevOps

DevOps and organizations that are both software developers and service providers were among core assumptions of our research. Thus, the target was to develop a demonstrator of a tool that supports both development and operation through an integrated user interface.

As discussed in Section 3 we selected deferred deployment but struggled with some implementation issues when we wanted to integrate the ACCORDS brokering platform. ACCORDS would have been easier to integrate if we had selected immediate deployment. However, our work started to follow a simple demo plan that assumed the deferred deployment. After implementing and working with the prototype, we have learned that the selection between those approaches is not obvious.

- Immediate deployment assumes that the user has the built and tested binary before brokering. On the other hand, the user should be given feedback on the results of the brokering – and even have a veto – before allocation of cloud resources and deployment. This means that the whole chain from the version management to deployment cannot be automated since the user needs to be consulted about the results of brokering after building but before brokering. This means that deployment would actually be a separate step.
- In the immediate deployment ACCORDS assumes that it will manage the cloud resources after deployment. However, we wanted to demonstrate a DevOps tool with an integrated user interface to control the whole chain from development to deployment. In the immediate deployment method, interactions between ACCORDS and COAPS are hidden and not accessible to our integration core (all components were integrated to CoRED). For example, provisioning and deployment logs could not be shown in the integrated UI. Also offering the management view would not be possible since access to COAPS through ACCORDS was incomplete.
- Most of the time application deployments are updates to an existing application. In the case of update, deployed application could simply replace the old version, in some other cases it is better to allocate new PaaS with new cloud resources for the new version and to make the switch over as a separate and controlled action. Use of immediate deployment, at least with the current implementation of ACCORDS, would not give enough control for this.

4.3 Implementation issues

Although we showed that the presented integration is possible, we did not use the different components in a way that was intended by their developers. This led to some integration problems. In the following, we list some of the issues we had in our implementation.

Selection between immediate and deferred deployment method in brokering was a difficult choice. At first, the immediate method seemed the natural choice. However, in the immediate deployment method, interactions between ACCORDS and COAPS are hidden from CoRED's point of view. For example, provisioning and deployment logs cannot be shown in CoRED. In addition, there is no way to call COAPS APIs through ACCORDS which would make implementation of the management view in the integrated tool impossible. In addition to reasons discussed in Subsection 4.2, these limitations of ACCORDS led us to the deferred deployment method. In the deferred method, the application deployment can be carried out via a direct interaction with COAPS at any time after resource provisioning. This direct interaction with COAPS allows access to the wide range of generic APIs of COAPS. For example, each button in management view should call the related COAPS API; Stop and delete buttons call *stop the application* and *delete the application* APIs respectively.

During the EASI-CLOUDS project, we installed an instance of ACCORDS and tested it together with our integrated tool. However, our current demonstrator does not use ACCORDS, since the implementation of its deferred deployment method was not as mature as immediate deployment. Consequently, integration to our tool was beyond the time and resources we had to complete the project. Instead, we use the graphical tool, shown in Figure 7, to collect resource requirements for brokering and we simulated the brokering behavior of ACCORDS. This interface was developed in parallel and by another organization, and the user experience is not that well integrated as other parts of the tool.

Although we deployed applications to different PaaS providers (e.g., University of Helsinki, Public Cloud Foundry, etc.) through our tool, deployment to all providers offered in the graphical tool is not implemented. Thanks to COAPS, new PaaS providers could be easily added to our tool.

Software development tools and conventions deal with various information about the application. Examples of such information include dependencies, external components, and required operating systems and services. That information is an important input to brokering and should not be manually queried from the user. All that information can be included in the data that is stored in the revision control database. Actually the data includes all input to brokering, SLA agreements and output of brokering. This information is not necessarily logical part of the revision since a single revision may be deployed to several locations. In our demo and proof of concept, we stored the result of the brokerage among the other files of the project and thus it was available for CI and CD pipeline. In a real implementation, the system should separate revisioning of the application and deployment information. Both of them should be stored in RCS as separate but interdependent entities.

5 Related work

Individual cloud-based development tools have been implemented for different purposes regarding software development and deployment. These tools range from simple code editors to tool sets that complete several parts of development and deployment pipelines. However, we believe that our tool is unique in the sense that it offers some features that have never been implemented before in one integrated tool.

There are several cloud-based IDEs and many of them could be used as a component of the integrated tool. One of such IDEs is Cloud9. Its backend has been implemented using Node.js and its frontend is based on a JavaScript-based editor called ACE. The editor used in our research, CoRED, is also based on ACE editor, whereas our backend has been implemented based on Vaadin framework. Cloud9 supports deployment to various services (e.g., Heroku, Google App Engine, Cloud foundry, etc.). But deployment to each of them needs a user to install related command line tool and learn how to use it.

One of the tools similar to our work is IBM Bluemix[18]. It can be used to build, run, and manage different kinds of apps targeted to the web, mobile, and smart devices. Bluemix offers a wide variety of runtimes (e.g. Java, Go, PHP, etc.) and a web-based editor for online coding. While Bluemix editor gives the same experience as the best desktop tools for certain programming languages like JavaScript, syntax-highlighting is the only significant feature when it comes to Java. As a contrast, CoRED supports more advanced features like error checking, code completion, and collaborative features. Moreover, Bluemix is built on Cloud Foundry. But we leverage COAPS to add support for heterogeneous PaaS technologies. Finally, Bluemix lacks cloud brokerage which is the key feature of our work.

Using DevOps tools such as Chef[19] and Juju[20], one can automate deployment processes and configure the underline infrastructure. However, these tools support certain artifacts. So, implementation of a comprehensive automated deployment pipeline requires several tools. [13] describes how to orchestrate these artifacts by transforming them into a standards-based TOSCA model[21]. In our research, we orchestrate various DevOps artifacts supported by different PaaS technologies (e.g., Cloud Foundry, Openshift, etc.) through COAPS. COAPS defined a unified model for representation of DevOps artifacts and proposes generic APIs. Moreover, in our research, the DevOps experience is provided through a single and integrated tool.

6 Conclusion

In this research, we have combined automation and brokered cloud to establish an integrated development tool. By using our tool, one can develop applications

[18] `http://www.ibm.com/cloud-computing/bluemix/`
[19] `https://www.chef.io/chef/`
[20] `https://jujucharms.com/`
[21] `https://www.oasis-open.org/committees/tosca/`

in a cloud-based IDE, find the most suitable cloud provider, and deploy the application in the target platform. To enable the fast development and delivery cycle, we automated all steps. Firstly, cloud brokerage automatically finds the deployment target out of several cloud platforms. Secondly, automatic provisioning and deployment are done through a unified interface for heterogeneous PaaS. Finally, continuous deployment pipeline automates code integrations, builds, and deployments.

Our proof-of-concept also demonstrates DevOps by enabling users to manage the deployed applications from a single tool. Users are provided with an operation and management console inside the tool with which they can stop, start, and delete the deployed applications.

For the future work, we first plan to connect our graphical tool to an instance of ACCORDS instead of simulating its behavior. Moreover, we believe that there are many automation opportunities besides build, testing and deployment steps while developing software. For instance, applications can be automatically instrumented so that they collect usage data. By the help of such system, the developers receive fast feedback of which features are important to users, thereby focusing on them first.

Acknowledgments

Major part of the work has been done in ITEA2 project EASI-CLOUDS (2012-2014) and the authors would like to thank all members of the consortium, and especially Mario Lopez-Ramos, Janne Lautamäki, and Jamie Marshall. Part of this research has also been done in Tekes-funded Digile program Need for Speed[22] (2014-17).

References

1. Computer Science department. Telecom SudParis.: The compatible one application and platform service (coaps) api specification. `http://www.compatibleone.com/community/wp-content/uploads/2014/05/COAPS-Spec.v1.5.3.pdf`.
2. Fowler, M.: Continuous integration (May 2006) `http://www.martinfowler.com/articles/continuousIntegration.html`.
3. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. 1st edn. Addison-Wesley Professional (2010)
4. Humble, J., Molesky, J.: Why enterprises must adopt devops to enable continuous delivery. IT Journal **24**(8) (2011) 6–12
5. Jekal, M., Krebs, A., Nurmela, M., Peltonen, J., Röhr, F., Plogmeier, J.F., Altmann, J., Gagnaire, M., Lopez-Ramos, M.: Deliverable 1.5 final business models for easi-clouds. easi-clouds project report. (2014) `http://easi-clouds.eu/wp-content/uploads/2014/12/Deliverable_1_5_Final_business_models.docx`.

[22] `http://www.n4s.fi/en/`

6. Kilamo, T., Nieminen, A., Lautamäki, J., Aho, T., Koskinen, J., Palviainen, J., Mikkonen, T.: Knowledge transfer in collaborative teams: Experiences from a two-week code camp. In: Companion Proceedings of the 36th International Conference on Software Engineering. ICSE Companion 2014, New York, NY, USA, ACM (2014) 264–271
7. Lautamäki, J., Nieminen, A., Koskinen, J., Aho, T., Mikkonen, T., Englund, M.: Cored: Browser-based collaborative real-time editor for java web applications. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work. CSCW '12, New York, NY, USA, ACM (2012) 1307–1316
8. Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., Leaf, D.: NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292). CreateSpace Independent Publishing Platform, USA (2012)
9. Mikkonen, T., Nieminen, A.: Elements for a cloud-based development environment: Online collaboration, revision control, and continuous integration. In: Proceedings of the WICSA/ECSA 2012 Companion Volume. WICSA/ECSA '12, New York, NY, USA, ACM (2012) 14–20
10. Nieminen, A., Lautamäki, J., Kilamo, T., Palviainen, J., Koskinen, J., Mikkonen, T.: Collaborative coding environment on the web: A user study. Developing Cloud Software Algorithms, Applications, and Tools,(60) (2013) 275–300
11. Sellami, M., Yangui, S., Mohamed, M., Tata, S.: Paas-independent provisioning and management of applications in the cloud. In: Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on. (June 2013) 693–700
12. Suonsyrjä, S., Mikkonen, T.: Designing an unobtrusive analytics framework for java applications. In: Accepted to IWSM Mensura 2015, to appear
13. Wettinger, J., Breitenbücher, U., Leymann, F.: Standards-based devops automation and integration using tosca. In: Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing. UCC '14, Washington, DC, USA, IEEE Computer Society (2014) 59–68
14. Yangui, S., Marshall, I.J., Laisne, J.P., Tata, S.: Compatibleone: The open source cloud broker. Journal of Grid Computing **12**(1) (2014) 93–109