

Lab Report for Software Engineering course
Lab 1: Git & Dev Cloud

Wang, Chen
16307110064
School of Software
Fudan University

March 10, 2019

Contents

1	Background Knowledge of the lab	3
1.1	Version Control System	3
1.1.1	The application of VCS	3
1.1.2	The general usage of VCS	3
1.1.3	Various tools for VCS	4
1.2	Git	5
1.2.1	The feature of Git in comparison to other VCS tools . . .	5
1.2.2	The application of Git	7
1.2.3	The general usage of Git	7
1.2.4	The usage of Git in this lab	9
1.3	Java EE	10
1.3.1	The needs and application of Java EE	10
1.3.2	Common frameworks of Java EE	11
1.4	Spring Boot	12
1.4.1	The feature of Spring Boot in comparison to other Java EE frameworks	12
1.4.2	The usage of Spring Boot in this lab	12
1.5	Classroom platform of Huawei Cloud	12
1.5.1	The functionalities of Huawei Cloud	12
1.5.2	The functions of Huawei Cloud used in this lab	13
1.6	Life cycle in software engineering	13
1.6.1	The meaning of life cycle in software engineering	13
1.6.2	The life cycle displayed in this lab	14
2	Steps of accomplishing this Lab	15
2.1	Git operations	15
2.1.1	Log in	15
2.1.2	DevCloud console	15
2.1.3	Code Management	15
2.2	Spring Boot framework construction	21
2.3	Executions on the Spring Boot framework	21
2.3.1	Create a new task	21
2.3.2	Adjust the deploy sequence	21
2.3.3	Configure and install the JDK	22
2.3.4	Choose deployment origin	22
2.3.5	Configure startup service	22
2.3.6	Configure pending execution	22
2.3.7	Configure stopping service	22

<i>CONTENTS</i>	2
2.3.8 Start deployment	24
2.3.9 Test the deployed application	24
3 Conclusion	27

Chapter 1

Background Knowledge of the lab

1.1 Version Control System

1.1.1 The application of VCS

A component of software configuration management, version control, also known as revision control or source control, is the management of changes to documents, computer programs, large web sites, and other collections of information. Changes are usually identified by a number or letter code, termed the "revision number", "revision level", or simply "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

The need for a logical way to organize and control revisions has existed for almost as long as writing has existed, but revision control became much more important, and complicated when the era of computing began. The numbering of book editions and of specification revisions are examples that date back to the print-only era. Today, the most capable (as well as complex) revision control systems are those used in software development, where a team of people may concurrently make changes to the same files.

Version control systems (VCS) most commonly run as stand-alone applications, but revision control is also embedded in various types of software such as word processors and spreadsheets, collaborative web docs and in various content management systems, e.g., Wikipedia's page history. Revision control allows for the ability to revert a document to a previous revision, which is critical for allowing editors to track each other's edits, correct mistakes, and defend against vandalism and spamming in wikis.

1.1.2 The general usage of VCS

In computer software engineering, revision control is any kind of practice that tracks and provides control over changes to source code. Software developers

sometimes use revision control software to maintain documentation and configuration files as well as source code.

As teams design, develop and deploy software, it is common for multiple versions of the same software to be deployed in different sites and for the software's developers to be working simultaneously on updates. Bugs or features of the software are often only present in certain versions (because of the fixing of some problems and the introduction of others as the program develops). Therefore, for the purposes of locating and fixing bugs, it is vitally important to be able to retrieve and run different versions of the software to determine in which version(s) the problem occurs. It may also be necessary to develop two versions of the software concurrently: for instance, where one version has bugs fixed, but no new features (branch), while the other version is where new features are worked on (trunk).

At the simplest level, developers could simply retain multiple copies of the different versions of the program, and label them appropriately. This simple approach has been used in many large software projects. While this method can work, it is inefficient as many near-identical copies of the program have to be maintained. This requires a lot of self-discipline on the part of developers and often leads to mistakes. Since the code base is the same, it also requires granting read-write-execute permission to a set of developers, and this adds the pressure of someone managing permissions so that the code base is not compromised, which adds more complexity. Consequently, systems to automate some or all of the revision control process have been developed. This ensures that the majority of management of version control steps is hidden behind the scenes.

Moreover, in software development, legal and business practice and other environments, it has become increasingly common for a single document or snippet of code to be edited by a team, the members of which may be geographically dispersed and may pursue different and even contrary interests. Sophisticated revision control that tracks and accounts for ownership of changes to documents and code may be extremely helpful or even indispensable in such situations.

Revision control may also track changes to configuration files, such as those typically stored in `/etc` or `/usr/local/etc` on Unix systems. This gives system administrators another way to easily track changes made and a way to roll back to earlier versions should the need arise.

1.1.3 Various tools for VCS

Differences between distributive and non-distributive VCS systems

Distributed version control systems (DVCS) takes a peer-to-peer approach to version control, as opposed to the client-server approach of centralized systems. Distributed revision control synchronizes repositories by exchanging patches from peer to peer. There is no single central version of the codebase; instead, each user has a working copy and the full change history.

Advantages of DVCS (compared with centralized systems) include:

1. Allows users to work productively when not connected to a network.
2. Common operations (such as commits, viewing history, and reverting changes) are faster for DVCS, because there is no need to communicate

with a central server. With DVCS, communication is only necessary when sharing changes among other peers.

3. Allows private work, so users can use their changes even for early drafts they do not want to publish.
4. Working copies effectively function as remote backups, which avoids relying on one physical machine as a single point of failure.
5. Allows various development models to be used, such as using development branches or a Commander/Lieutenant model.
6. Permits centralized control of the "release version" of the project
7. On FOSS software projects it is much easier to create a project fork from a project that is stalled because of leadership conflicts or design disagreements.

Disadvantages of DVCS (compared with centralized systems) include:

1. Initial checkout of a repository is slower as compared to checkout in a centralized version control system, because all branches and revision history are copied to the local machine by default.
2. The lack of locking mechanisms that is part of most centralized VCS and still plays an important role when it comes to non-mergeable binary files such as graphic assets.
3. Additional storage required for every user to have a complete copy of the complete codebase history.

Some originally centralized systems now offer some distributed features. For example, Subversion is able to do many operations with no network. Team Foundation Server and Visual Studio Team Services now host centralized and distributed version control repositories via hosting Git.

1.2 Git

1.2.1 The feature of Git in comparison to other VCS tools

Strong support for non-linear development

Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. In Git, a core assumption is that a change will be merged more often than it is written, as it is passed around to various reviewers. In Git, branches are very lightweight: a branch is only a reference to one commit. With its parental commits, the full branch structure can be constructed.

Distributed development

Like Darcs, BitKeeper, Mercurial, SVK, Bazaar, and Monotone, Git gives each developer a local copy of the full development history, and changes are copied from one such repository to another. These changes are imported as added development branches and can be merged in the same way as a locally developed branch.

Compatibility with existent systems and protocols

Repositories can be published via Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), rsync (removed in Git 2.8.0), or a Git protocol over either a plain socket, or Secure Shell (ssh). Git also has a CVS server emulation, which enables the use of extant CVS clients and IDE plugins to access Git repositories. Subversion and svk repositories can be used directly with git-svn.

Efficient handling of large projects

Torvalds has described Git as being very fast and scalable, and performance tests done by Mozilla showed that it was an order of magnitude faster than some version-control systems; fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server.

Cryptographic authentication of history

The Git history is stored in such a way that the ID of a particular version (a commit in Git terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed. The structure is similar to a Merkle tree, but with added data at the nodes and leaves. (Mercurial and Monotone also have this property.)

Toolkit-based design

Git was designed as a set of programs written in C and several shell scripts that provide wrappers around those programs. Although most of those scripts have since been rewritten in C for speed and portability, the design remains, and it is easy to chain the components together.

Pluggable merge strategies

As part of its toolkit design, Git has a well-defined model of an incomplete merge, and it has multiple algorithms for completing it, culminating in telling the user that it is unable to complete the merge automatically and that manual editing is needed.

Garbage accumulates until collected

Aborting operations or backing out changes will leave useless dangling objects in the database. These are generally a small fraction of the continuously growing history of wanted objects. Git will automatically perform garbage collection when enough loose objects have been created in the repository. Garbage collection can be called explicitly using `git gc --prune`.

Periodic explicit object packing

Git stores each newly created object as a separate file. Although individually compressed, this takes a great deal of space and is inefficient. This is solved by the use of packs that store a large number of objects delta-compressed among

themselves in one file (or network byte stream) called a packfile. Packs are compressed using the heuristic that files with the same name are probably similar, but do not depend on it for correctness. A corresponding index file is created for each packfile, telling the offset of each object in the packfile. Newly created objects (with newly added history) are still stored as single objects, and periodic repacking is needed to maintain space efficiency. The process of packing the repository can be very computationally costly. By allowing objects to exist in the repository in a loose but quickly generated format, Git allows the costly pack operation to be deferred until later, when time matters less, e.g., the end of a work day. Git does periodic repacking automatically, but manual repacking is also possible with the `git gc` command. For data integrity, both the packfile and its index have an SHA-1 checksum inside, and the file name of the packfile also contains an SHA-1 checksum. To check the integrity of a repository, run the `git fsck` command.

1.2.2 The application of Git

Git is a distributed version-control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows. Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Its current maintainer since 2005 is Junio Hamano. As with most other distributed version-control systems, and unlike most client-server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server. Git is free and open-source software distributed under the terms of the GNU General Public License version 2.

1.2.3 The general usage of Git

Working with local repositories

git init This command turns a directory into an empty Git repository. This is the first step in creating a repository. After running `git init`, adding and committing files/directories is possible. Usage:

```
# change directory to codebase 1
$ cd /file/path/to/code 2
3
# make directory a git repository 4
$ git init 5
```

git add Adds files in the to the staging area for Git. Before a file is available to commit to a repository, the file needs to be added to the Git index (staging area). There are a few different ways to use `git add`, by adding entire directories, specific files, or all unstaged files. Usage:

```
$ git add <file or directory name> 1
```


git commit Record the changes made to the files to a local repository. For easy reference, each commit has a unique ID. It's best practice to include a message with each commit explaining the changes made in a commit. Adding a commit message helps to find a particular change or understanding the changes. Usage:

```
# Adding a commit with message 1
$ git commit -m "Commit_message_in_quotes" 2
```

git status This command returns the current state of the repository. *git status* will return the current working branch. If a file is in the staging area, but not committed, it shows with *git status*. Or, if there are no changes it'll return *nothing to commit, working directory clean*. Usage:

```
$ git status 1
```

git config With Git, there are many configurations and settings possible. *git config* is how to assign these settings. Two important settings are user.name and user.email. These values set what email address and name commits will be from on a local computer. With *git config*, a *-global* flag is used to write the settings to all repositories on a computer. Without a *-global* flag settings will only apply to the current repository that you are currently in. There are many other variables available to edit in *git config*. From editing color outputs to changing the behavior of *git status*. Usage:

```
$ git config <setting> <command> 1
```

git branch To determine what branch the local repository is on, add a new branch, or delete a branch. Usage:

```
# Create a new branch 1
$ git branch <branch_name> 2
3
# List all remote or local branches 4
$ git branch -a 5
6
# Delete a branch 7
$ git branch -d <branch_name> 8
```

git checkout To start working in a different branch, use *git checkout* to switch branches. Usage:

```
# Checkout an existing branch 1
$ git checkout <branch_name> 2
3
# Checkout and create a new branch with that name 4
$ git checkout -b <new_branch> 5
```

git merge Integrate branches together. *git merge* combines the changes from one branch to another branch. For example, merge the changes made in a staging branch into the stable branch. Usage:

```
# Merge changes into current branch 1
$ git merge <branch_name> 2
```

Working with remote repositories

git remote To connect a local repository with a remote repository. A remote repository can have a name set to avoid having to remember the URL of the repository. Usage:

```
# Add remote repository 1
$ git remote <command> <remote_name> <remote_URL> 2
3
# List named remote repositories 4
$ git remote -v 5
```

git clone To create a local working copy of an existing remote repository, use *git clone* to copy and download the repository to a computer. Cloning is the equivalent of *git init* when working with a remote repository. Git will create a directory locally with all files and repository history. Usage:

```
$ git clone <remote_URL> 1
```

git pull To get the latest version of a repository run *git pull*. This pulls the changes from the remote repository to the local computer. Usage:

```
$ git pull <branch_name> <remote_URL/remote_name> 1
```

git push Sends local commits to the remote repository. *git push* requires two parameters: the remote repository and the branch that the push is for. Usage:

```
$ git push <remote_URL/remote_name> <branch> 1
2
# Push all local branches to remote repository 3
$ git push -all 4
```

1.2.4 The usage of Git in this lab

In this lab, we are going to utilize Git to get and modify the code created by the teaching assistant, upload them to the server and then finish other consequent operations. Therefore, the critical steps in our lab include

1. Clone the code from the Huawei cloud repository;
2. Commit our changes to the Git system;
3. Push our commit onto the Huawei cloud repository.

Clone

To check out the code from the repository on the Huawei Cloud platform, we ought to navigate to `classroom.devcloud.huaweicloud.com`, login via IAM method, enter the project and navigate to the code management section. The Lab1 section is shown in the list allowing us to clone/download the repository from the cloud. As we select the "via HTTPS" method, we get a link enabling us to clone the code from the Git console. After getting the link, we can clone the code by entering the following the command below:

```
git clone https://codehub.devcloud.huaweicloud.com/2019      1
    rjgc_Lab1_xzstudent00300001/Lab1.git
```

After cloning the repository without specifying the path to save the contents, we will get the downloaded files in the user folder by default.

Commit

After changing the source code, we can view the changes we have made by using the **git status** command in the git root folder. The changes will be displayed in the command line window. As we confirm the changes we have made, we can commit the changes by input the **git commit** command to commit the changes. It should be noted that the commit message is required for each commit we have made. A commit successful message will be shown after a successful commit.

Push

After committing the specified codes into the local repository, we can push the commits into the cloud repository. If we have made multiple commits, they will be pushed to the cloud in the single push command.

Since we have specified the remote repositories where we pulled the codes from, we will be free from entering the remote address of the repository again. A single **git push** command will be able to instruct the git system to push all the commits to the remote repository at the Huawei Cloud.

1.3 Java EE

1.3.1 The needs and application of Java EE

Java Platform, Enterprise Edition (Java EE) is the standard in community-driven enterprise software. Java EE is developed using the Java Community Process, with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals. Each release integrates new features that align with industry needs, improves application portability, and increases developer productivity.

Today, Java EE offers a rich enterprise software platform and with over 20 compliant Java EE implementations to choose from.

As stated above, the Java EE platform is designed to help developers create large-scale, multitiered, scalable, reliable, and secure network applications. A shorthand name for such applications is enterprise applications, so called because these applications are designed to solve the problems encountered by large

enterprises. Enterprise applications are not only useful for large corporations, agencies, and governments, however. The benefits of an enterprise application are helpful, even essential, for individual developers and small organizations in an increasingly networked world.

The features that make enterprise applications powerful, like security and reliability, often make these applications complex. The Java EE platform reduces the complexity of enterprise application development by providing a development model, API, and runtime environment that allow developers to concentrate on functionality.

1.3.2 Common frameworks of Java EE

According to some online materials, I have discovered that there are a list of Spring framework that are most commonly used by enterprises across the world. Ordered by popularity, these frameworks are:

1. **Spring MVC.** Old but gold, Spring MVC is still ahead of the curve after more than a decade since its first release. After its expansion to embrace complete MVC framework, Spring kept on evolving adopting changes and turned into a full-scale Java framework for Internet-facing applications, offering software engineers a powerful toolkit for web application development and application configuration as well as for security projects.
2. **Struts 2.** To elaborate even more on existing Java frameworks that are widely used by modern software engineers, we decided to refer to the successor of Apache's Struts 1, Struts 2. This Java framework is quite a find for engineers who work with building contemporary Java EE web apps.
3. **Hibernate.** Although not on RebelLabs' list either, it is worth mentioning Hibernate when debating the best Java framework. This mapping Java framework cracks object-relational impedance mismatch issues by substituting persisting DB accesses high-level object handling functions.
4. **JSF.** Being a part of Java EE, JavaServer Faces is supported by Oracle. Although this one is not the best frameworks for speedy Java development, it is easy to utilize because of great documentation provided by Oracle.
5. **Vaadin.** Using GWT for rendering the end web page, Vaadin became one of the uber popular frameworks modern developers choose when creating applications for business. Utilizing a well-known component-based approach, Vaadin takes the burden off developer's shoulder by communicating the changes made to the browser.
6. **Google Web Toolkit.** GWT is an another free Java framework allowing coders to create and optimize sophisticated web-based apps.
7. **Grails.** This particular web framework is regarded as a dynamic tool enhancing engineers' productivity due to its opinionated APIs, sensible defaults, as well as its convention-over-configuration paradigm. Seamless Java integration makes this particular framework one of the top choices for plenty of programmers worldwide.

1.4 Spring Boot

1.4.1 The feature of Spring Boot in comparison to other Java EE frameworks

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that we can “just run”.

The designers of Spring Boot take an opinionated view of the Spring platform and third-party libraries so we can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.

Specifically, Spring Boot has the following features:

1. Create stand-alone Spring applications;
2. Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files);
3. Provide opinionated “starter” dependencies to simplify your build configuration;
4. Automatically configure Spring and 3rd party libraries whenever possible;
5. Provide production-ready features such as metrics, health checks and externalized configuration;
6. Absolutely no code generation and no requirement for XML configuration.

1.4.2 The usage of Spring Boot in this lab

The main part of the project is based on the Spring Boot framework. In the framework, we have reset the Spring Application run class to be Lab1Applocation, then the controller of the Spring Boot Application is reset to Hello Controller class, whose mapping is requested at “/Hello”. The index method under the Hello Controller class is quite simple, returning a String showing the name and student ID, which each student should change to their own information.

In addition to these properties, the Spring Boot application should specify a assigned port number, indicating that it will not run on the default port. The reason for this is that a couple of students are running their applications on the same server, who has a unique IP address. However, if all of them use the same port number, part of them won’t be able to start the service due to port resource conflict. Assigning each student a unique port number enables them to start the service concurrently.

1.5 Classroom platform of Huawei Cloud

1.5.1 The functionalities of Huawei Cloud

HUAWEI CLOUD, the cloud service brand of the Huawei marquee, packages the 30-plus years of expertise in ICT infrastructure products and solutions so customers can build what they need into their profile with building block-like ease. The brand is committed to providing stable, secure, reliable, and sustainable cloud services to help organizations of all sizes grow in the intelligent world. Complementing the already impressive offerings, the Inclusive AI strategy from

HUAWEI CLOUD allows everyone to get what they need at an affordable price point, efficiently make easy use of cutting-edge technologies, and rest assured that their profile remains fully secure throughout the entire life cycle.

HUAWEI CLOUD provides a powerful computing platform and easy-to-use development platform to support Huawei's full-stack, all-scenario AI strategy. At HUAWEI CONNECT 2018, HUAWEI CLOUD released its one-stop AI development platform ModelArts, AI vision application development platform HiLens, and quantum computing simulator and programming framework HiQ. Huawei also set up its AI Developer Enablement Program to foster collaboration with developers, partners, universities, and research institutions, with the ultimate goal of making AI more inclusive.

1.5.2 The functions of Huawei Cloud used in this lab

Within the scope of this lab, we are going to finish all periods of the life cycle of our project on the Huawei Cloud platform.

First of all, the start-up codes that the teaching assistants have prepared are put on the Code Management section on the cloud platform. We ought to clone the codes from the repository to our local desktop to make modifications. After proper modification, we need to commit our changes and push them to the remote repository on the Huawei Cloud. After the codes of our version are well-prepared, we can deploy the project with the help of the modularized steps of the Huawei Cloud in the deploy section. Each of us is assigned a server machine account, the system being Linux CentOS. After the steps of installing the JDK, configuring the Spring Boot environments and choosing the project source files, our project can be started.

During the period of the service running, we should visit our server via the IP address and port number and will be able to see the hello page if our service is started successfully, the Spring Boot service listens on the specified port and the launcher class is invoked successfully.

1.6 Life cycle in software engineering

1.6.1 The meaning of life cycle in software engineering

The systems development life cycle (SDLC), also referred to as the application development life-cycle, is a term used in systems engineering, information systems and software engineering to describe a process for planning, creating, testing, and deploying an information system. The systems development life cycle concept applies to a range of hardware and software configurations, as a system can be composed of hardware only, software only, or a combination of both. There are usually six stages in this cycle: analysis, design, development and testing, implementation, documentation, and evaluation.

A systems development life cycle is composed of a number of clearly defined and distinct work phases which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems. Like anything that is manufactured on an assembly line, an SDLC aims to produce high-quality systems that meet or exceed customer expectations, based on customer requirements, by delivering systems which move through each clearly

defined phase, within scheduled time frames and cost estimates. Computer systems are complex and often (especially with the recent rise of service-oriented architecture) link multiple traditional systems potentially supplied by different software vendors. To manage this level of complexity, a number of SDLC models or methodologies have been created, such as waterfall, spiral, Agile software development, rapid prototyping, incremental, and synchronize and stabilize.

SDLC can be described along a spectrum of agile to iterative to sequential methodologies. Agile methodologies, such as XP and Scrum, focus on lightweight processes which allow for rapid changes (without necessarily following the pattern of SDLC approach) along the development cycle. Iterative methodologies, such as Rational Unified Process and dynamic systems development method, focus on limited project scope and expanding or improving products by multiple iterations. Sequential or big-design-up-front (BDUF) models, such as waterfall, focus on complete and correct planning to guide large projects and risks to successful and predictable results. Other models, such as anamorphic development, tend to focus on a form of development that is guided by project scope and adaptive iterations of feature development.

1.6.2 The life cycle displayed in this lab

In this lab, we can see the full period of the life cycle in a project in software engineering. Specifically, the steps are displayed in the following ways:

1. **Planning.** The platform to place the codes and accomplish the management work, the framework to accomplish the back end and the front end of the server, the server system environment, the displaying user interface and the way to hand in the works are already worked out by the teaching assistants.
2. **Analysis.** The teaching assistants should consider how long the time limit should be given to the students, how detailed the documentation should be written and whether the difficulty is suitable for the students.
3. **Design.** After analysis, the teaching assistants should make appropriate frameworks, make sure that they can be started successfully, the interfaces are clearly specified and finally, place the codes on the Huawei Cloud platform.
4. **Implementation.** The students should follow the instructions of the teaching assistants and implement the concrete methods. Here specifically, we need only to modify a String and a port number.
5. **Maintenance.** Here this version of the Lab requires no maintenance after submission.

Chapter 2

Steps of accomplishing this Lab

2.1 Git operations

2.1.1 Log in

First of all, we should log in to the Classroom platform at `classroom.devcloud.huaweicloud.com/`, log in via *IAM Log in* with the assigned account name and set password. Then we can select the homework section to enter the homework in the software engineering class.

2.1.2 DevCloud console

Then we need to select the Lab1 in the homework list, create a Dev Cloud project from the teacher's template. After waiting for the system to finish creating the project, we can enter the Dev Cloud console.

2.1.3 Code Management

Code Repository

Then we can select the code management section to view the repository for this Lab. In order to clone the code from the remote repository to the local machine, we need to select the *HTTPS* visiting method and copy the address of the repository.

Clone the repository

Enter the following code in the local terminal, followed by the account name and password to clone the repository to the local machine.

```
git clone https://codehub.devcloud.huaweicloud.com/2019    1
    rjgc-Lab1_xzstudent00300001/Lab1.git
```

After cloning the repository without specifying the path to save the contents, we will get the downloaded files in the user folder by default. The cloned files are like the figure shown in (Figure 2.1)

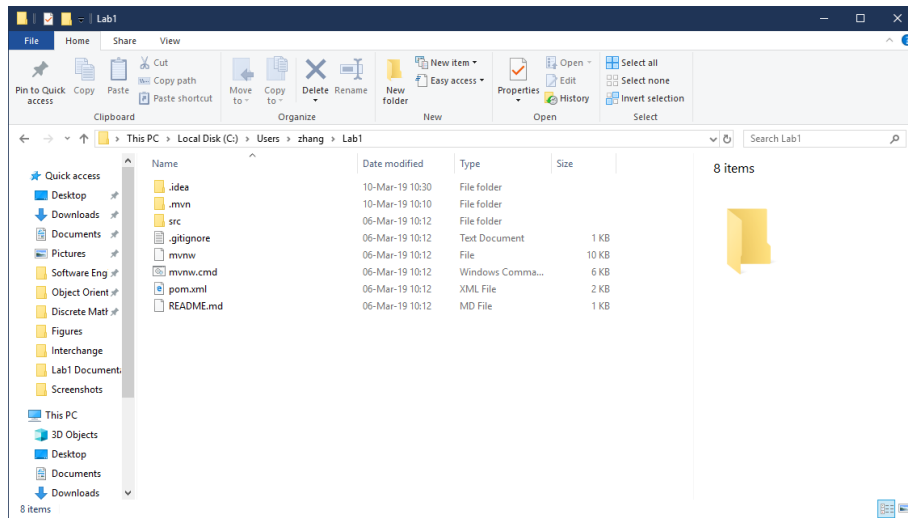


Figure 2.1: Clone the repository to the local machine

Modify the source code

There are two places that we need to modify the source code. In the file `\src\main\java\fudan\se\lab1\Lab1Application.java`, I need to modify the String to “Hello, Name: Wang Chen, Id: 16307110064”; and in the file `\src\main\resources\application.properties`, here we need to modify the port number to the specified number. The modified files are shown as the following pictures Figure 2.2 and Figure 2.3.

View repository status

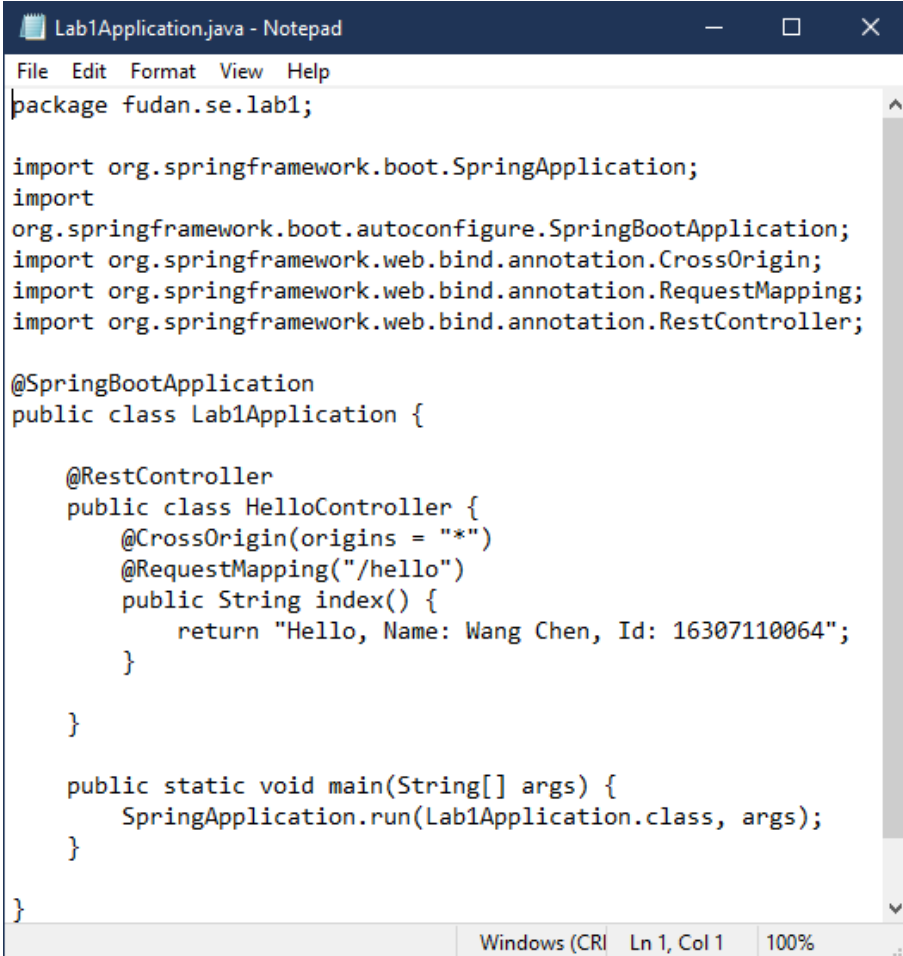
After I have modified the codes, I should navigate to the repository folder and view the status of the repository. In the bash shell terminal, I input the commands `cd Lab1` and `git status` specifically. Then the modified file is shown in red color. In the screenshot commit operation, I only changed the first file, leaving the next file’s change to the next commit. The status of this change is shown in the Figure 2.4 below.

Commit changes

The command `git add .` under the working folder adds the changes to the local cache area of the Git. Afterwards, the `git commit . -m “commit message”` command will commit the changes, where the commit message is specified to be “update name and id”. The commit result is shown as the Figure 2.5 below.

Push commits

As we finish making changes and committing them, we can use the `git push` command to push the changes to the remote repository. The result of the `push` command is shown as the Figure 2.6 below.



```
Lab1Application.java - Notepad
File Edit Format View Help
package fudan.se.lab1;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
public class Lab1Application {

    @RestController
    public class HelloController {
        @CrossOrigin(origins = "*")
        @RequestMapping("/hello")
        public String index() {
            return "Hello, Name: Wang Chen, Id: 16307110064";
        }
    }

    public static void main(String[] args) {
        SpringApplication.run(Lab1Application.class, args);
    }
}
```

Windows (CR... Ln 1, Col 1 100%

Figure 2.2: Changes of the string

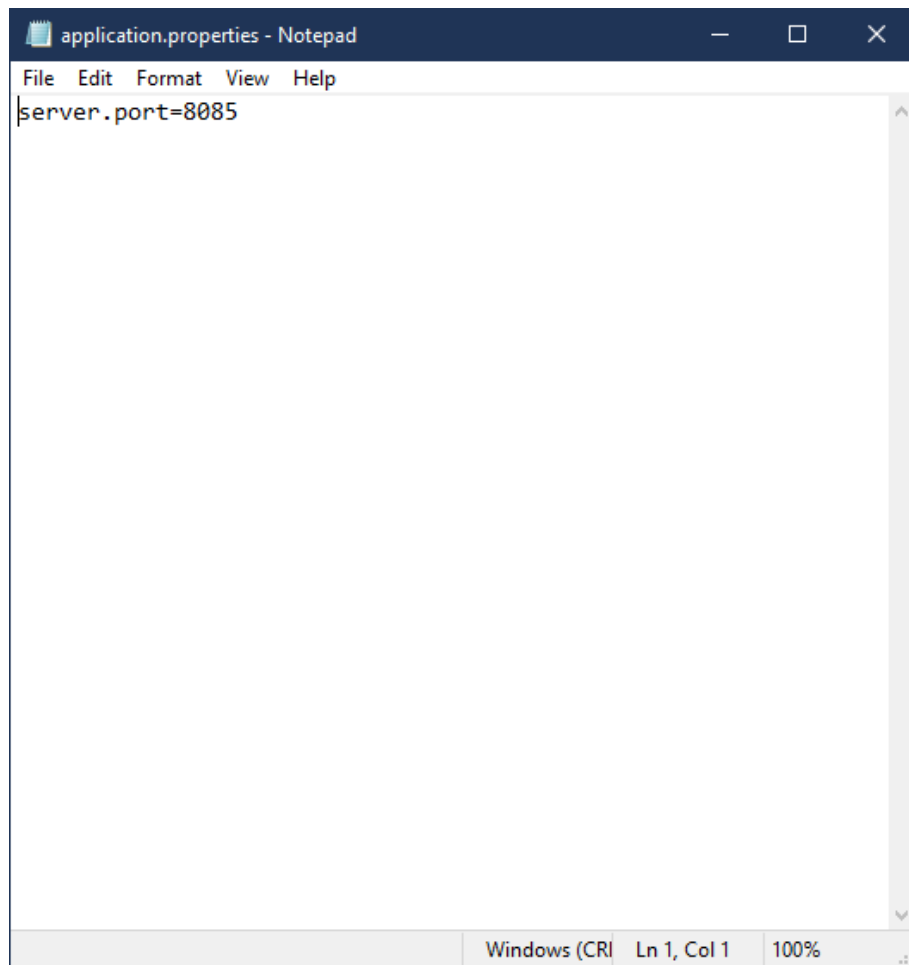
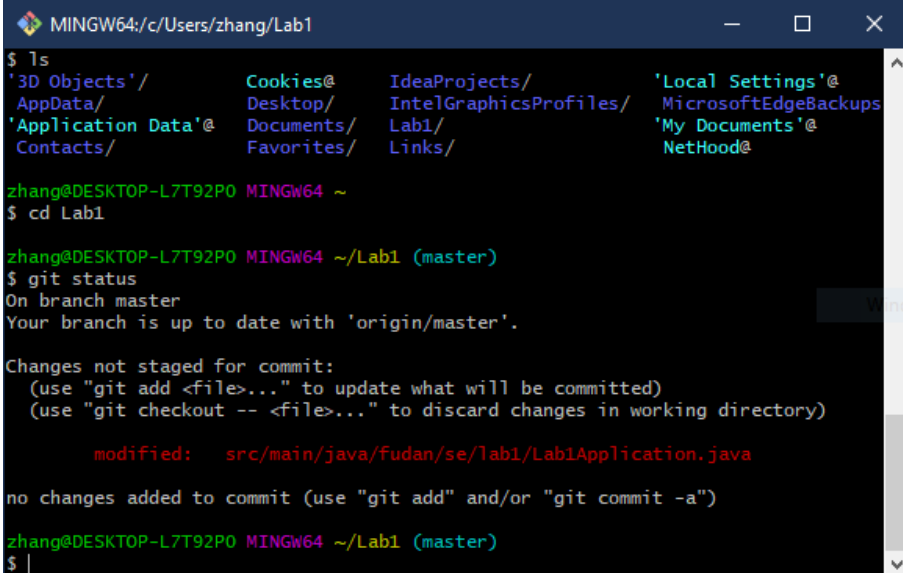


Figure 2.3: Changes of the port number



```
MINGW64:/c/Users/zhang/Lab1
$ ls
'3D Objects'/'Cookies@'IdeaProjects/'Local Settings'@
AppData/'Desktop/'IntelGraphicsProfiles/'MicrosoftEdgeBackups
'Application Data'@Documents/'Lab1/'My Documents'@
Contacts/'Favorites/'Links/'NetHood@

zhang@DESKTOP-L7T92P0 MINGW64 ~
$ cd Lab1

zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

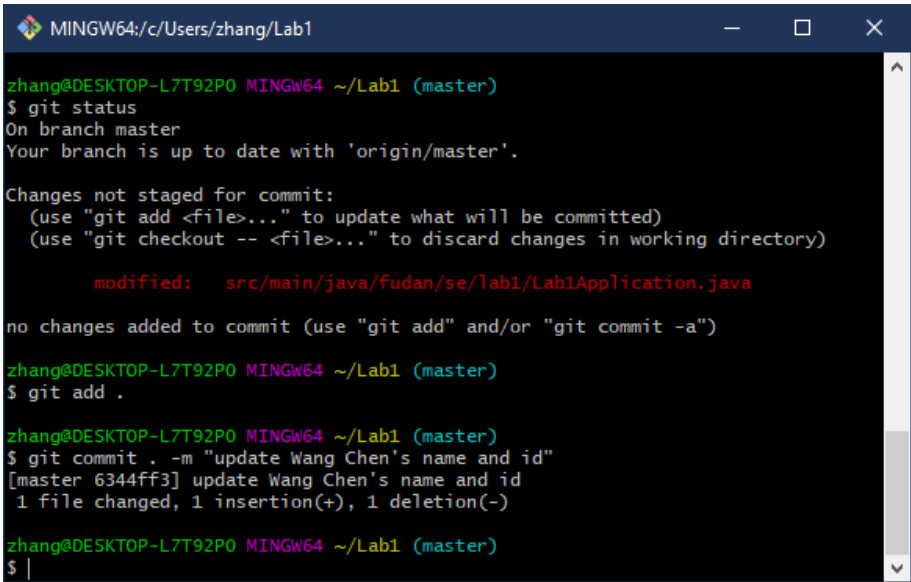
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   src/main/java/fudan/se/lab1/Lab1Application.java

no changes added to commit (use "git add" and/or "git commit -a")

zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master)
$ |
```

Figure 2.4: Status of the git folder



```
MINGW64:/c/Users/zhang/Lab1

zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   src/main/java/fudan/se/lab1/Lab1Application.java

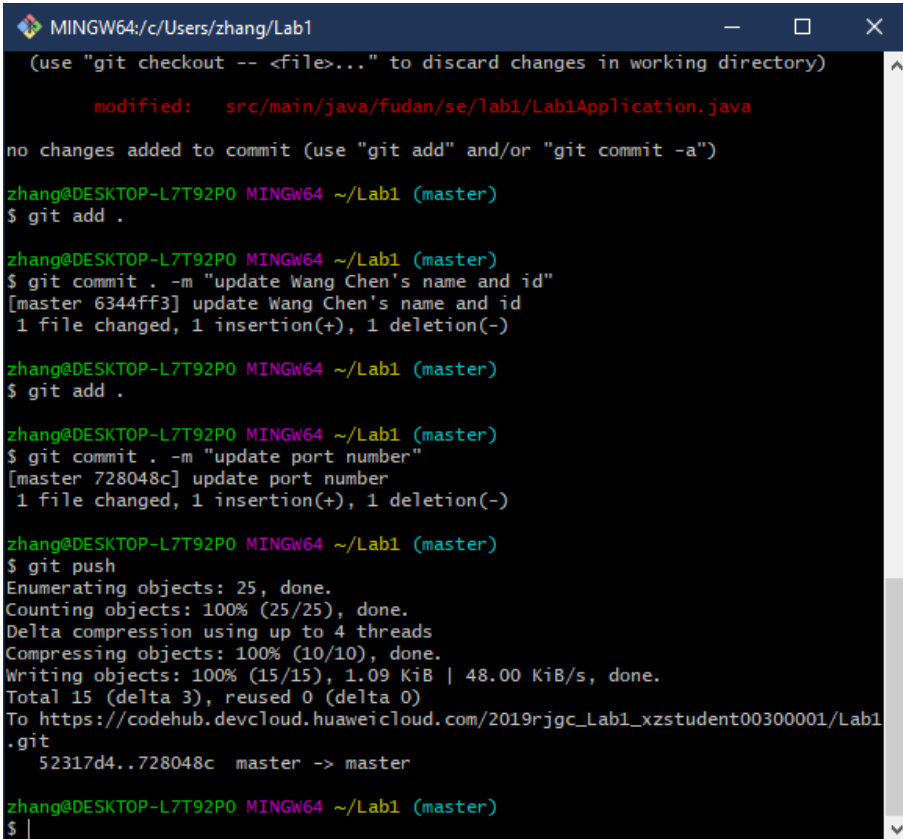
no changes added to commit (use "git add" and/or "git commit -a")

zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master)
$ git add .

zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master)
$ git commit -m "update Wang Chen's name and id"
[master 6344ff3] update Wang Chen's name and id
1 file changed, 1 insertion(+), 1 deletion(-)

zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master)
$ |
```

Figure 2.5: Result of the *git commit* command

A screenshot of a Windows terminal window titled 'MINGW64:/c/Users/zhang/Lab1'. The terminal shows the following sequence of commands and output:
- Initial state: A message '(use "git checkout -- <file>..." to discard changes in working directory)' is shown, followed by a red line indicating a file was modified: 'src/main/java/fudan/se/lab1/Lab1Application.java'.
- Command: 'no changes added to commit (use "git add" and/or "git commit -a")'
- Command: 'zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master) \$ git add .'
- Command: 'zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master) \$ git commit . -m "update Wang Chen's name and id"'
- Output: '[master 6344ff3] update Wang Chen's name and id', '1 file changed, 1 insertion(+), 1 deletion(-)'
- Command: 'zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master) \$ git add .'
- Command: 'zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master) \$ git commit . -m "update port number"'
- Output: '[master 728048c] update port number', '1 file changed, 1 insertion(+), 1 deletion(-)'
- Command: 'zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master) \$ git push'
- Output: 'Enumerating objects: 25, done.', 'Counting objects: 100% (25/25), done.', 'Delta compression using up to 4 threads', 'Compressing objects: 100% (10/10), done.', 'Writing objects: 100% (15/15), 1.09 KiB | 48.00 KiB/s, done.', 'Total 15 (delta 3), reused 0 (delta 0)', 'To https://codehub.devcloud.huaweicloud.com/2019rjgc_Lab1_xzstudent00300001/Lab1.git', '52317d4..728048c master -> master'
- Final prompt: 'zhang@DESKTOP-L7T92P0 MINGW64 ~/Lab1 (master) \$ |'Figure 2.6: Result of the *git push* command

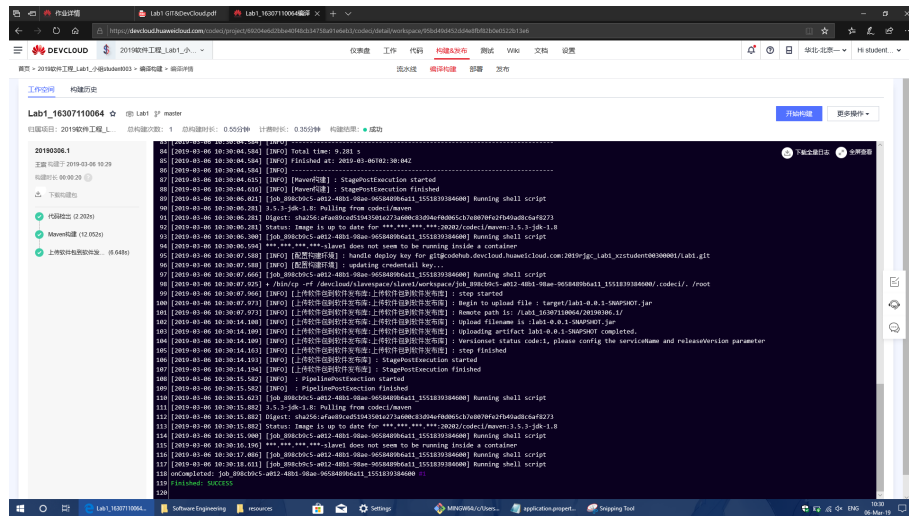


Figure 2.7: Result of the first deploy process

2.2 Spring Boot framework construction

Going back to the Dev Cloud workstation console, we can deploy the project we have just finished editing. Selecting the Build release section and the compile construction subsection, we can create a new task with our student ID and name being the name of the task. Then we need to select the source code from the code management repository and the master branch. After selecting the *Maven Template* option, we can create a new task. As we finish the deploy process, we can see a *Finished Success* prompt, indicating that the deploy process is finished. The result of my deploy is shown in the Figure 2.7 above.

2.3 Executions on the Spring Boot framework

2.3.1 Create a new task

We should create a new task and name the task with our student ID and name. In the next step, we should choose the Spring Boot deployment and go down to the next step.

2.3.2 Adjust the deploy sequence

We should drag the “Stop Spring Boot service” to the end of the list and remove the “URL health test” method. After removing, we ought to add a new service called “Pending execution” between “Start Spring Boot Service” and “Stop Spring Boot Service”. The result of the adjusted deploy sequence is shown as the Figure 2.8 below.

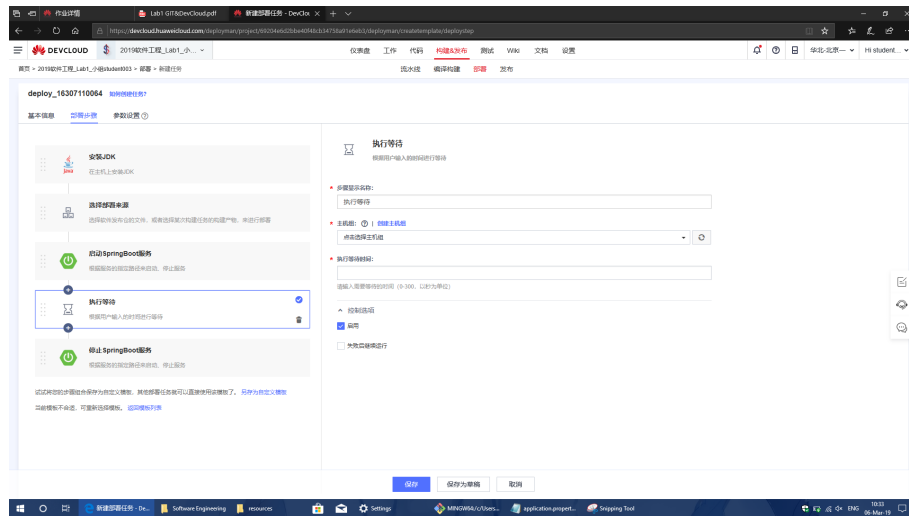


Figure 2.8: Result of the adjusted deploy sequence

2.3.3 Configure and install the JDK

We should create a machine group in the configuring JDK part and create a new host according to the given information. Then we can install the JDK at `/usr/local/jdk`.

2.3.4 Choose deployment origin

In the choosing deployment origin section, we can select the host we have just created and the task we have deployed in the previous part, downloading it to the `/usr/local` path. The result of the configured deployment origin is shown as the Figure 2.9 below.

2.3.5 Configure startup service

In the “Start Spring Boot Service” section, we should configure the start-up service at the absolute path `/usr/local/lab1-0.0.1-SNAPSHOT.jar` and configure as the information given. The host should be the one we have created in the previous subsection. The result of the configured startup service is shown as the Figure 2.10 below.

2.3.6 Configure pending execution

In the “Configure pending execution” section, we should create a 200 second pending time for us to visit our service. The result of the configured pending execution is shown as the Figure 2.11 below.

2.3.7 Configure stopping service

In the “Configure stopping service” section, we should configure the stop service at the absolute path `/usr/local/lab1-0.0.1-SNAPSHOT.jar` and configure as the

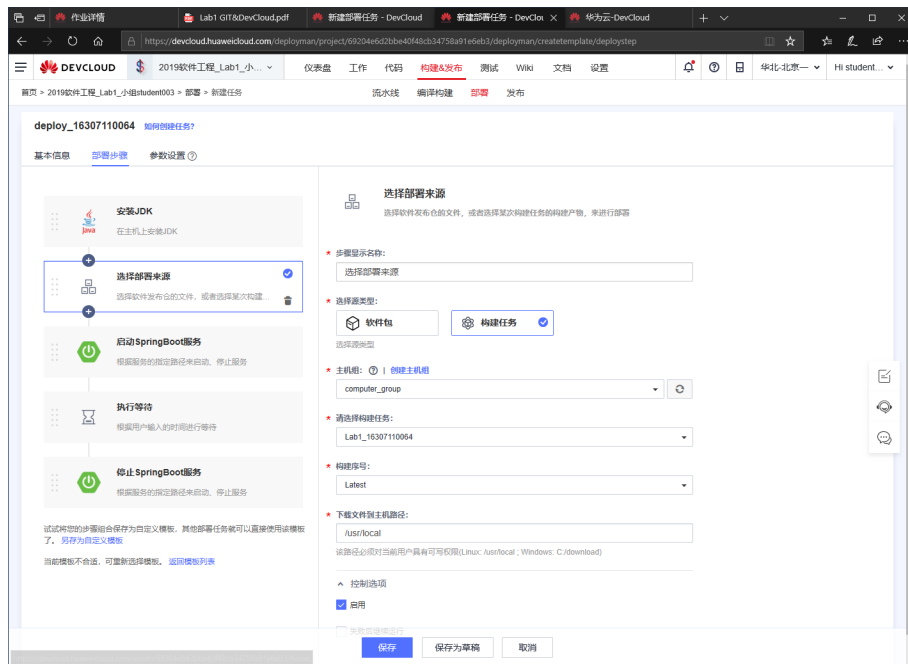


Figure 2.9: Result of the adjusted deploy sequence

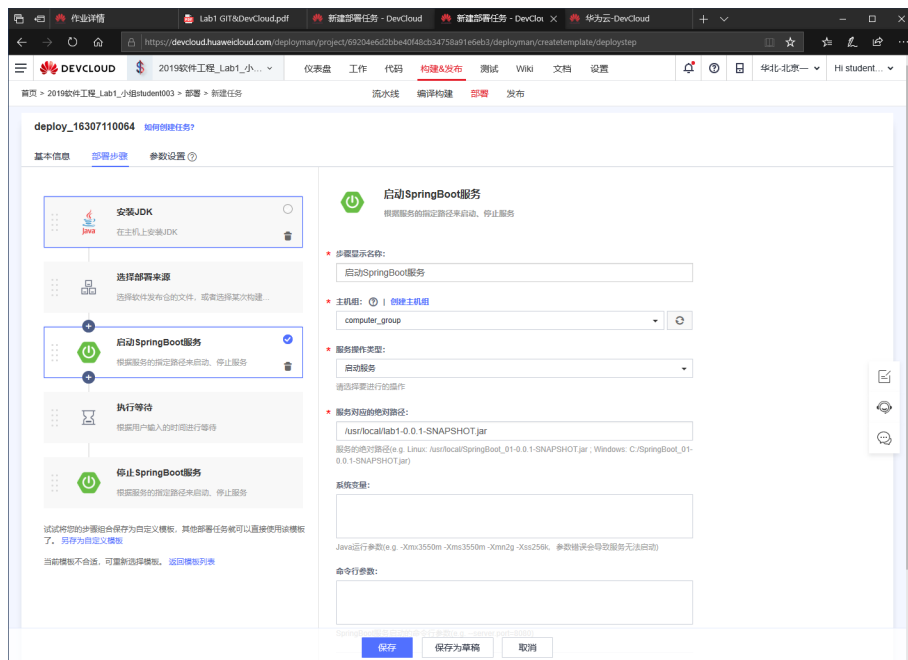


Figure 2.10: Result of the configured startup service

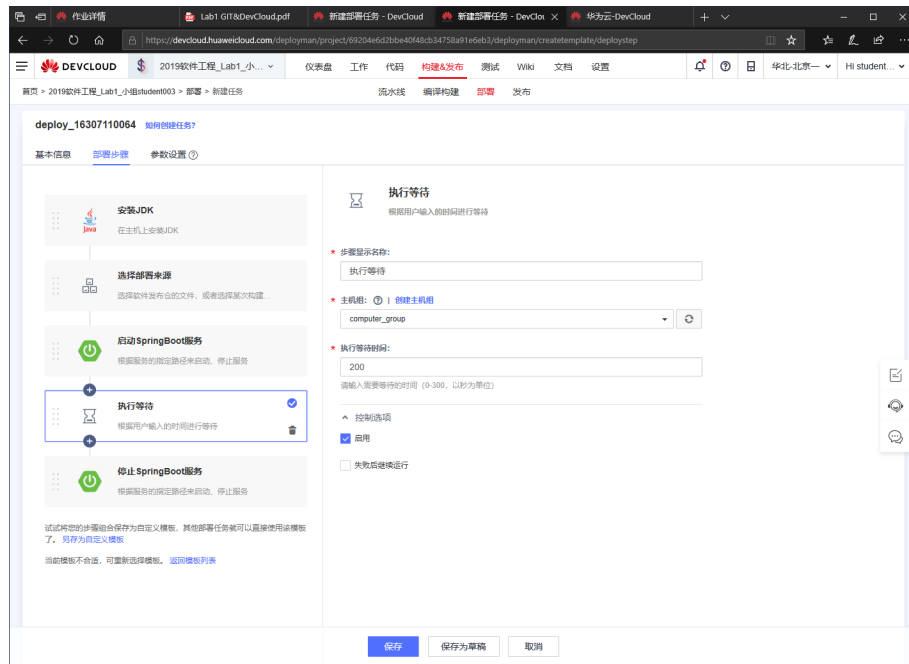


Figure 2.11: Result of the configured pending execution

information given. The host should be the one we have created in the previous subsection. The result of the configured stop service is shown as the Figure 2.12 below.

2.3.8 Start deployment

As we choose to save and start the deployment process, we can observe the process of our project being deployed on the Huawei Dev Cloud. The result of the start deployment process is shown as the Figure 2.13 below.

2.3.9 Test the deployed application

While our service is running, we can visit our service by the host IP address and the port number. The result of the test deployed application part is shown as the Figure 2.14 below.

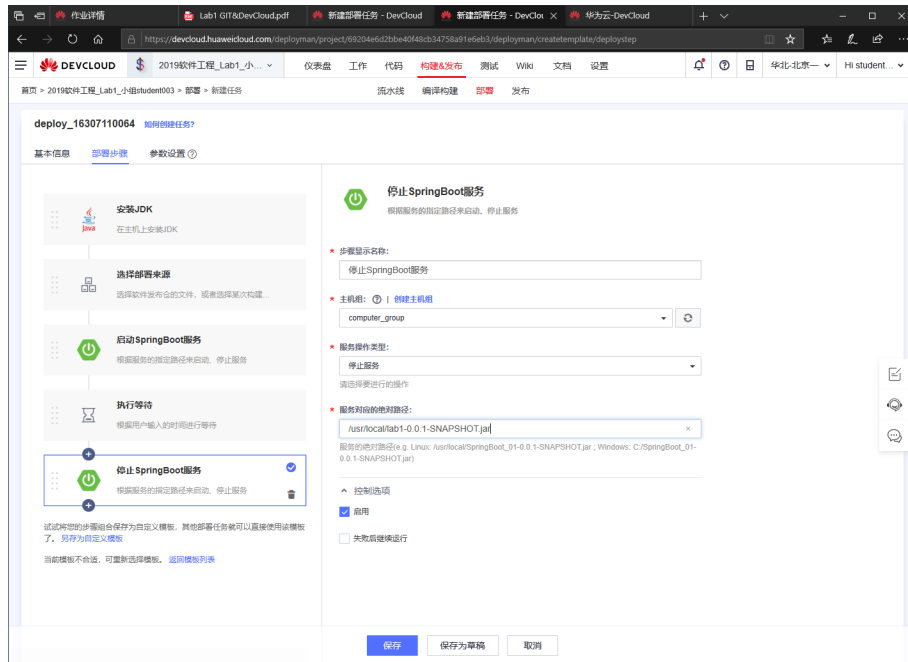


Figure 2.12: Result of the configured startup service

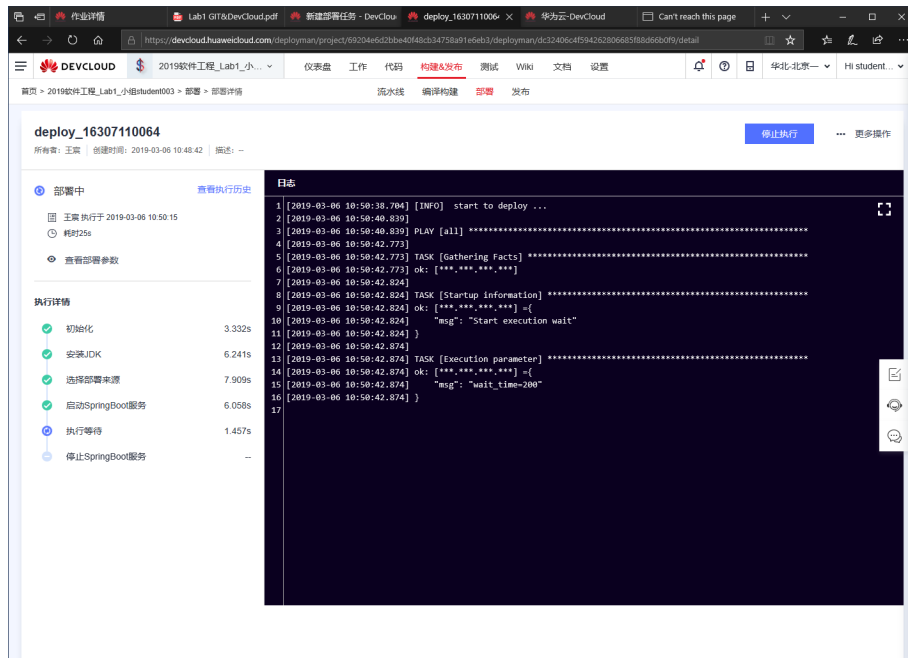


Figure 2.13: Result of the start deployment process

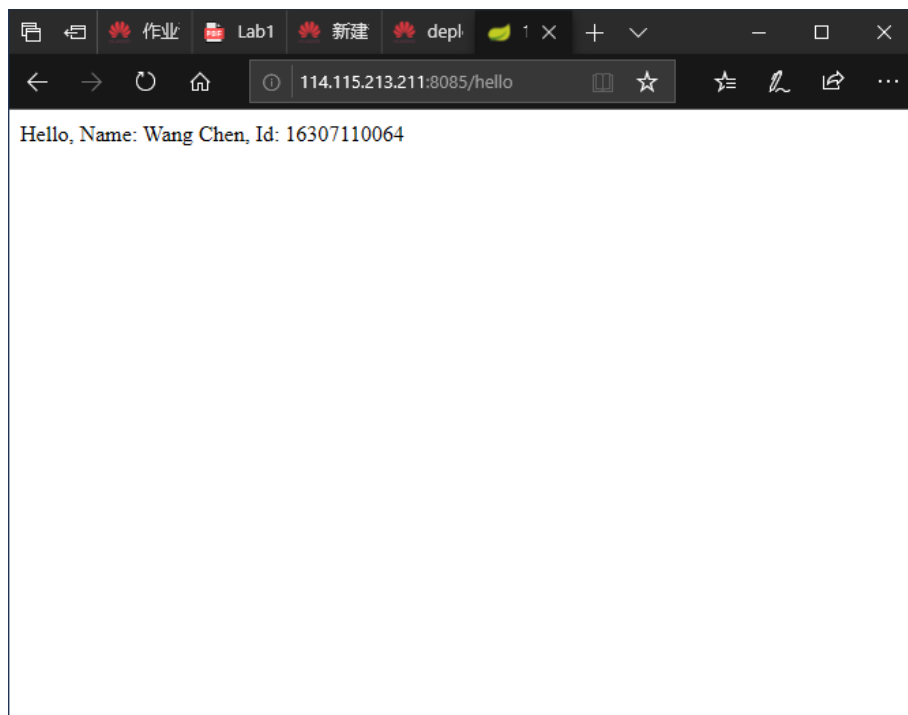


Figure 2.14: Result of the test deployed application

Chapter 3

Conclusion