DADock - Linux Build Server Configuration Guide

V2_1806, Thu Jun 28 2018 04:38:56 GMT+0000 (UTC)

Table of Contents

Introduction

This document is a guideline for the configuration of the Build service used by DADock in Linux. The configurations of the Linux build service of DADock will be automatically executed by Linux build service installer.

Caution

Before executing the configurations of the build service, it is a must that the installation of DADock is already complete. Refer to a separate manual [DADock – Installation Guide].

Adjust the time zone of the build service with the timezone of the platform server.
Refer to the manual below on how to set the timezone of the servers.

- Separate Manual [DADock - Installation Guide] > [Timezone settings of the server]

Operating environment

Supported Operating Environment of the Build Service

Below is the table of the supporting operating environment for the build service.

- Red Hat

OS	Red Hat Enterprise Linux 7.3
CPU	Core 2 ~
MEM	4GB ~

HDD	30GB ~

- CentOS

OS	CentOS 7.4.1708
CPU	Core 2 ~
MEM	4GB ~
HDD	30GB ~

1. Linux build service installer

1.1. Automatically installed tools by the Build Server installer

By executing Linux build service installer, the following tools are automatically installed.

Tool	Description
GitLab Runner	Tool for executing GitLab CI job.
Docker	An execution environment of container type for applications.
Ansible	Tool for executing automatic deployment.

Information

With the use of Linux build service installer, the hosts settings needed to access DADock by the build server is also automatically set.

1.2. Execution of build server installer

1.2.1. Stop security softwares

Stop the operations if there is running security softwares such as McAfee, etc.

1.2.2. Login to DADock platform server as an administrator

Use applications such as Tera Term and login as an administrator to the server where DADock is installed (referred as platform server below). When logged in as a user aside from administrator, execute the following command to switch to administrator.

```
$ sudo su -
```

1.2.3. Get the archived file

In the platform server of the directory listed below, get the Linux build service installer (linux-dadock-buildservice-installer.tar.gz).

```
/opt/dadock/buildservice/linux/linux-dadock-buildservice-installer.tar.gz
```

1.2.4. Save the archived file

Save the archived file of the linux build service installer [linux-dadock-buildservice-installer.tar.gz] in the server where Linux build service is installed (referred to as build server) .

Information

- Use applications such as WinSCP and Tera Term when saving the linux build service installer archived file.
- In the succeeding procedures, the following description is used to describe the directory where the archived file is saved.

```
~/linux-dadock-buildservice-installer.tar.gz
```

1.2.5. Login to Build Service

Use applications such as Tera Term to log in to build server as an administrator.

1.2.6. Extract the archived file

Execute the following commands to extract the archived file of the build server installer.

```
# cd ~
# tar zxvf linux-dadock-buildservice-installer.tar.gz
```

1.2.7. Execute the installer

When installing DADock, specify whether the installation would be in online mode or offline mode. The behavior of installation changes as follows depending on the specified mode of installation.

Online Mode(CentOS, Red Hat Enterprise Linux)

- The needed libraries, tools are obtained from the Internet and installed in the server.
- For the environments that can be connected to the internet, use the online mode of installation depending on the used OS.

Offline Mode

- The libraries and tools for CentOS7.4 bundled with DADock will be used and execute installation. *1 *2
- Since the included libraries and tools will be used, installation is faster compared to the online mode.
- For environments that cannot connect to the internet, prepare a CentOS7.4 environment and use the offline mode.
- Even in environments that can connect to the internet, offline mode installation can be used if the environment is CentOS7.4.

*1: For Red Hat Enterprise Linux and using offline mode, the libraries for CentOS are installed and there is possibility that there will be a violation of license in Red Hat Enterprise Linux.
*2: In the minimal version of CentOS, offline mode installation cannot be executed. It is recommended to use Everything version of CentOS.

Proxy Settings

In the online mode, the necessary libraries are obtained from the internet by using 'yum' and 'curl'. Consequently, the proxy settings of the previously stated 2 commands should be set for a proxy environment.

Refer to the following command, specify the installation mode and execute the installation.

Example 1. Specification of the installation mode

```
./install.sh [InstallationMode]
```

Online mode(CentOS)	online_centos
Online mode(Red Hat Enterprise Linux)	online_rhel
Offline mode	offline

List 1. When offline mode is specified

```
# cd linux-dadock-buildservice-installer
# ./install.sh offline
```

When the following message is displayed, the installation is complete. To continue, register a GitLab Runner.

```
Installation completed!
To register a runner, execute the runner registration (register.sh) file.
```

1.2.8. Register GitLab Runner

Execute the following commands to register a GitLab Runner.

```
# cd ~/linux-dadock-buildservice-installer
# ./register.sh
```

Input the necessary information in each field that is being asked in the interactive type of installation.

Questions	Description	Sample Inputs
1) Runner type to be installed. Available input runner types are 'shell' and 'docker'.	Runner type	shell
2) Tag type of the runner. Multiple inputs are possible by using comma(,) as a separator.	Tag of the runner that processes the job	shared
3) Description of the runner.	Runner description	shared
4) Docker image to be used during build up.	Default docker image	gradle:3.5-jdk8

Information

Question no. 4) will only be displayed when 'docker' type was selected in question no.1).

In case of wrong inputs

In case of wrong input, press [Ctrl+C] keys to cancel and execute the [./register.sh] again.

When the following message is displayed, the registration of GitLab Runner is complete.

```
Registration completed!
```

Advanced usage

Refer to the document below for usage of multiple runner or controlling of the execution of job runner. 4.3. Advanced usage of GitLab Runner

1.2.9. Start security softwares

Start again the stopped security softwares.

1.3. Installation check

Caution

The following procedures executes operational check by checking the version of each tool. There is a possibility that there is a change in the displayed version. Refer to the site below for the latest version.
Separate Manual [DADock - Introduction] > [Operating Requirement]

1.3.1. GitLab Runner installation check

Execute the following command to check if GitLab Runner is properly installed.

```
# gitlab-runner -v
Version:      9.4.0
Git revision: ef0b1a6
Git branch:
GO version:   go1.8.3
Built:        Sat, 22 Jul 2017 08:21:04 +0000
OS/Arch:      linux/amd64
```

1.3.2. Docker installation check

Execute the following command to check if Docker is properly installed.

```
# docker -v
Docker version 17.06.0-ce, build 02c1d87
```

1.3.3. Ansible installation check

Execute the following command to check if Ansible is properly installed.

```
# ansible --version
ansible 2.3.1.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = Default w/o overrides
  python version = 2.7.5 (default, Nov  6 2016, 00:28:07) [GCC 4.8.5 20150623 (Red Hat 4.8.5-11)]
```

2. Using shell type runner to execute jobs

Shell type runner is used by the build server to execute jobs.
In that case, it is a must to configure an environment in the build server that can execute jobs.
In this section, configuration of an environment that can execute a job and
2 examples of job execution using a shell type runner are introduced.

2.1. Using Gradle to build a Java project

This introduces the usage of Gradle to build a Java project. As a sample project, the Java Library project avalaible in the creation of new repository page of DADock Portal will be used.

2.1.1. Configuring a build server environment

1. Install OpenJDK 8

The defined default version of JDK in the source and class file of [Java Library] project is version 1.8. To be able to compile with the use of the default settings, openJDK 8 should be installed

```
# sudo yum install -y java-1.8.0-openjdk-devel.x86_64
```

In case different versions of JDK is to be installed

If in case other versions of JDK (1.7 etc) is to be installaed, change the defined JDK version in the yum command.

```
# sudo yum install -y java-1.7.0-openjdk-devel.x86_64
```

In addition to that, change the build settings file(build.gradle) of Gradle.

```
(omitted)
sourceCompatibility = 1.7
targetCompatibility = 1.7
(omitted)
```

If in case an error occurs

When the execution of the command failed and the following failed message appears, it is possible that the cause of this is the proxy settings of yum. Set the proxy settings of yum properly and execute the same command again.

```
[Errno 14] curl#6 - "Could not resolve host:
```

2. Proxy Settings of Gradle

When gradle runs in a proxy environment, it is a must to set the proxy settings of Gradle. Refer to 4.1. Proxy settings - 4.1.1. Gradle and set the proxy settings of Gradle in the build server.

2.1.2. Execute build

1. Create a [Java Library] repository

At the creation of new repository page of DADock Portal, create a [Java Library] repository.

2. Check if the build process succeeds

Access GitLab, open the created repository [Java Library] and check the status of the build job if it is success.

2.2. Using Maven to build a Java project

This introduces the usage of Maven to build a Java project. As a sample project, the Java Library project avalaible in the creation of new repository page of DADock Portal will be used.

2.2.1. Configuring a build server environment

1. Install OpenJDK 8

Same with the procedure in Using Gradle to build a Java project, install OpenJDK 8.

2. Download and extract Maven

Execute the following commands to download and extract Maven.

```
# curl -O https://apache.osuosl.org/maven/maven-3/3.5.2/binaries/apache-maven-3.5.2-bin.tar.gz
# mkdir /opt/maven
# tar xzvf apache-maven-3.5.2-bin.tar.gz -C /opt/maven
# rm -f apache-maven-3.5.2-bin.tar.gz
```

If in case an error occurs

When the execution of the command failed, it is possible that the cause of this is the proxy settings of curl. Set the proxy settings of yum properly and execute the same command again.

3. Set the environment variables

Add the following commands at the end of the file settings `/etc/profile` to set the environment variable.

```
export MAVEN_HOME=/opt/maven/apache-maven-3.5.2
export PATH=$PATH:$MAVEN_HOME/bin
```

4. Reflect the changes made in the environment variables

Execute the following commands to reflect the changes made in the environment variables.

```
# source /etc/profile
```

5. Check the installation of Maven

Execute the following commands to check if Maven is properly installed.

```
# mvn -v
Apache Maven 3.5.2 (138edd61fd100ec658bfa2d307c43b76940a5d7d; 2017-10-18T07:58:13Z)
Maven home: /opt/maven/apache-maven-3.5.2
Java version: 1.8.0_151, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.151-1.b12.el7_4.x86_64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "3.10.0-514.10.2.el7.x86_64", arch: "amd64", family: "unix"
```

6. Repository settings of Maven

It is a must that authentication information of Maven is defined in the settings file during the pull and push in the Maven repository. Create a directory `/home/gitlab-runner/.m2` and change the owner to [gitlab-runner]. Then, under that directory create a file named `settings.xml`.

```
# mkdir /home/gitlab-runner/.m2
# chown gitlab-runner:gitlab-runner /home/gitlab-runner/.m2
# vi /home/gitlab-runner/.m2/settings.xml
```

List 2. settings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0 http://maven.apache.org/xsd/settings-1.1.0.xsd">
  <servers>
   <server>
      <id>central</id>
      <username>developer</username>
      <password>developer</password>
   </server>
   <server>
      <id>libs-release-local</id>
      <username>developer</username>
      <password>developer</password>
   </server>
  </servers>
</settings>
```

2.2.2. Add the remote repository in the Artifactory of the Platform server

In the registered by default remote repository of jcenter in the Artifactory, a part of the library in order to use Maven plugin is not registered. Therefore, a central repository of Maven will be added as a remote repository.

1. Log in Artifactory as an administrator.

2. Add a remote repository

At the left part of the page, select [Admin] → [Remote] and the Remote repository page will be displayed. Then, at the right upper part of the page, click [New] and select [Maven] for the Package Type. After that, input the following information and click the [TEST] button.
Check if the connection is successful then click [Save & Finish] button.

Item	**Input Value**
Repository Key	central
URL	http://central.maven.org/maven2

3. Register the remote repository in the virtual repository

At the left part of the page, select [Admin] → [Virtual] and the Virtual repository page will be displayed. Select [libs-release], from the [Available Repositories] selection, move "central" to [Selected Repositories] and click [Save & Finish] button.

2.2.3. Execute build

1. Create a [Java Library] repository

At the [Create New Repository] Page of DADock Portal, create a [Java Library] repository.

2. Clone the repository in the local machine

Access GitLab, open the created Java Library repository and check for the repository pass, clone the repository in the local machine.

3. Delete the file for Gradle

Delete the following files for these are used for gradle.

File Name
build.gradle
gradle/wrapper/gradle-wrapper.jar
gradle/wrapper/gradle-wrapper.properties
gradlew
gradlew.bat
settings.gradle

4．Create a file pom.xml

Create pom.xml file directly below the project with the following content.
(Edit the groupId and artifactId according to each repository)

pom.xml::Example：If in case the host name of DADock Portal is [portal.dadock.fujitsu.local]

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>samplelib.project.lib</groupId>
    <artifactId>samplelib</artifactId>
    <version>0.0.1</version>

    <repositories>
        <repository>
            <id>central</id>
            <url>http://artifactory.dadock.fujitsu.local/artifactory/libs-release</url>
        </repository>
    </repositories>

    <pluginRepositories>
        <pluginRepository>
            <id>central</id>
            <url>http://artifactory.dadock.fujitsu.local/artifactory/libs-release</url>
        </pluginRepository>
    </pluginRepositories>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>1.8</java.version>
        <sonar.host.url>http://sonarqube.dadock.fujitsu.local</sonar.host.url>
    </properties>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
```

```
                <groupId>org.sonarsource.scanner.maven</groupId>
                <artifactId>sonar-maven-plugin</artifactId>
                <version>3.4.0.905</version>
            </plugin>
            <plugin>
                <groupId>org.jacoco</groupId>
                <artifactId>jacoco-maven-plugin</artifactId>
                <version>0.7.9</version>
                <executions>
                    <execution>
                        <id>default-prepare-agent</id>
                        <goals>
                            <goal>prepare-agent</goal>
                        </goals>
                    </execution>
                    <execution>
                        <id>default-report</id>
                        <goals>
                            <goal>report</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <distributionManagement>
        <repository>
            <id>libs-release-local</id>
            <url>http://artifactory.dadock.fujitsu.local/artifactory/libs-release-local</url>
        </repository>
```

```
    </distributionManagement>

</project>
```

5. Edit .gitlab-ci.yml file

Edit .gitlab-ci.yml file with the following content.

```
stages:
  - build
  - test
  - publish

job_build:
  stage: build
  script:
    - mvn jar:jar
  artifacts:
    name: "${CI_PROJECT_NAME}_${CI_BUILD_REF_NAME}"
    paths:
      - target/*.jar

job_test:
  stage: test
  script:
    - mvn test jacoco:report sonar:sonar -Dsonar.branch=$CI_BUILD_REF_NAME
  artifacts:
    name: "${CI_PROJECT_NAME}_${CI_BUILD_REF_NAME}_testreports"
    paths:
      - target/site/jacoco/*

job_publish:
  stage: publish
  script:
    - mvn deploy
  only:
    - master
  when: manual
```

6. Commit the changes and reflect the changes to the remote repository

Commit all the changes and push it to the remote repository

7. Check if the build process is successful

Access GitLab, open the created Java Library repository and check if the build job is successful.

3. Using docker type runner to execute jobs

Docker type runner uses the Docker of the build server and configures the environment of job execution in a docker container. The image created internally can be used by registering an image in an external registry such as DockerHub or in the Container Registry of GitLab of DADock. In this section, registration of image in the Container Registry of GitLab in DADock and using that image as an execution environment of jobs for the build processes are introduced.

3.1. Building Java project with the use of the Docker image with Gradle proxy settings

This introduces an example of building Java project with the use of the Docker image with Gradle proxy settings As a sample project, the Java Library project avalaible in the creation of new project page in DADock Portal will be used.

3.1.1. Configuring the build server environment

1. Proxy settings of Docker

When using a proxy environment, it is a must to set the proxy settings of Docker. Refer to 4.1. Proxy Settings - 4.1.2. Docker to set the proxy settings of the Docker in the Build server.

3.1.2. Create a Docker image with Gradle proxy settings

1. Create a project in GitLab

From the Project Settings of DADock Portal, show the Project Creation Page and then create a Blank Project.

About Project Creation Page

Refer to the Reference Guide [2.5.1. Project Settings] for the operational procedures of Project Creation Page.

2. Clone the project in the local machine

Open the newly created project and check for the repository pass, clone the repository in the local machine.

3. Create Dockerfile

Inside the cloned repository, create a Dockerfile which uses a public image of openjdk.

List 3. Dockerfile

```
FROM openjdk:8-jdk

RUN mkdir /root/.gradle
COPY gradle/gradle.properties /root/.gradle
```

4. In the same directory with [Dockerfile], create another directory [gradle], and under it create the file 'gradle.properties'.

gradle/gradle.properties

Example：If in case the host name of DADock Portal is set to [portal.dadock.fujitsu.local]

```
systemProp.http.proxyHost=<HostName>
systemProp.http.proxyPort=<Port>
systemProp.http.proxyUser=<UserName>
systemProp.http.proxyPassword=<Password>
systemProp.http.nonProxyHosts=gitlab.dadock.fujitsu.local|sonarqube.dadock.fujitsu.local|artifactory.dadock.fujitsu.local

systemProp.https.proxyHost=<HostName>
systemProp.https.proxyPort=<Port>
systemProp.https.proxyUser=<UserName>
systemProp.https.proxyPassword=<Password>
systemProp.https.nonProxyHosts=gitlab.dadock.fujitsu.local|sonarqube.dadock.fujitsu.local|artifactory.dadock.fujitsu.loca
l
```

5. In the same directory with [Dockerfile], create a file '.gitlab-ci.yml' and set a job to configure an image and to push the image in the Container Registry.

List 4. gitlab-ci.yml:: Example：If in case the host name of DADock Portal is set to [portal.dadock.fujitsu.local], and the port number is set to [8081]

```
build_image:
  script:
    - docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN registry.dadock.fujitsu.local[:8081]
    - docker build -t registry.dadock.fujitsu.local[:8081]/dadock/<ProjectName>:8-jdk .
    - docker push registry.dadock.fujitsu.local[:8081]/dadock/<ProjectName>:8-jdk
    - docker logout registry.dadock.fujitsu.local[:8081]
```

*If in case there is no change in the default port number, 80, the port number in the above file can be ommited.

Information

For docker command samples, clicking the Registry menu present in each project present in GitLab will display the sample commands.

6．Commit the created files and push the changes to the remote repository.

7．Pipeline will be executed then, the image will be registered in the Container Registry and then the image will be displayed in the [registry] tab of GitLab.

3.1.3. Execute build

1．Register a Docker type GitLab Runner

It is a must to register a new GitLab Runner that uses the image registered in the procedures stated above. Refer to 1.2.7. Registering GitLab Runner to register a Docker type Runner just as follows.

Questions	Sample Input

Questions	Sample Input
1) Runner type to be installed. Available input runner types are 'shell' and 'docker'.	docker
2) Tag type of the runner. Multiple inputs are possible by using comma(,) as a separator.	docker-runner
3) Description of the runner.	docker-type-runner
4) Docker image to be used during build up.	registry.dadock.fujitsu.local/dadock/<ProjectName>:8-jdk

2. Create a Java Library repository

In the Create Project Page of DADock Portal, create a [Java Library] repository.

3. Clone the repository in the local machine

Access GitLab, open the newly created Java Library repository and copy the repository pass, then clone the repository in the local machine.

4. Edit the file .gitlab-ci.yml

In order to make use of eth Docker type Runner, edit the file .gitlab-ci.yml and set the created iamge.

List 5. gitlab-ci.yml:: Example：If in case the host name of DADock Portal is [portal.dadock.fujitsu.local]

```
image: registry.dadock.fujitsu.local/dadock/<ProjectName>:8-jdk

stages:
  - build
  - test
  - publish

job_build:
  stage: build
  tags:
    - docker-runner
  script:
    - chmod +x gradlew
    - ./gradlew jar
    - ./gradlew javadoc
  artifacts:
    name: "${CI_PROJECT_NAME}_${CI_BUILD_REF_NAME}"
    paths:
      - build/libs/*.jar
      - build/docs/javadoc/*

job_test:
  stage: test
  tags:
    - docker-runner
  script:
    - chmod +x gradlew
    - ./gradlew test sonarqube -PsonarBranch=$CI_BUILD_REF_NAME
  artifacts:
    name: "${CI_PROJECT_NAME}_${CI_BUILD_REF_NAME}_testreports"
    paths:
      - build/reports/tests/test/*

job_publish:
  stage: publish
  tags:
    - <Tag of the Docker type runner>
  script:
    - chmod +x gradlew
```

```
    - ./gradlew publish
  only:
    - master
  when: manual
```

5. Commit the changes and push it to the remote repository

6. Check if the build processes are successful

> *Information*
>
> There is a possibility that the build process might fail due to the following error.
>
> 
>
> ```
> Using Docker executor with image registry.dadock.fujitsu.local[:8081]/dadock/<ProjectName>:8-jdk ...
> Pulling docker image registry.dadock.fujitsu.local[:8081]/dadock/<ProjectName>:8-jdk ...
> ERROR: Preparation failed: Error response from daemon: Get
> http://registry.dadock.fujitsu.local[:8081]/dadock/<ProjectName>/8-jdk: unauthorized: HTTP Basic: Access
> denied
> ```
>
> This error occurs when docker logout is not executed during image configuration. To resolve the error, there is a need to delete the file `/home/gitlab-runner/.docker/config.json`.

Access GitLab, open the created Java Library repository and check if the build job is successful.

4. Appendix

4.1. Proxy settings

4.1.1. Gradle

Gradle Wrapper downloads from the internet the Gradle binary and uses that binary to execute Gradle. Accordingly, when using proxy environment, it is a must to set the proxy settings of the Gradle.
Make a directory [/home/gitlab-runner/.gradle] and change the owner to 'gitlab-runner', then create a file [gradle.properties] under that directory.

```
# mkdir /home/gitlab-runner/.gradle
# chown gitlab-runner:gitlab-runner /home/gitlab-runner/.gradle
# vi /home/gitlab-runner/.gradle/gradle.properties
```

```
/home/gitlab-runner/.gradle/gradle.properties
```

gradle.properties

Example：If in case the host name of DADock Portal is [portal.dadock.fujitsu.local]

```
systemProp.http.proxyHost=<HostName> +
systemProp.http.proxyPort=<Port> +
systemProp.http.proxyUser=<UserName> +
systemProp.http.proxyPassword=<Password> +
systemProp.http.nonProxyHosts=gitlab.dadock.fujitsu.local|sonarqube.dadock.fujitsu.local|artifactory.dadock.fujitsu.local
+

systemProp.https.proxyHost=<HostName> +
systemProp.https.proxyPort=<Port> +
systemProp.https.proxyUser=<UserName> +
systemProp.https.proxyPassword=<Password> +
systemProp.https.nonProxyHosts=gitlab.dadock.fujitsu.local|sonarqube.dadock.fujitsu.local|artifactory.dadock.fujitsu.loca
l
```

4.1.2. Docker

Download the image from the internet when using an image from external Docker registry such as DockerHub with the use of Docker type GitLab Runner and Dockerfile.+ Accordingly, when using proxy environment it is a must to set Gradle proxy settings. Edit the contents of the file [/etc/systemd/system/docker.service.d/docker.conf] as follows.

docker.conf

Example：If in case the host name of DADock Portal is [portal.dadock.fujitsu.local]

```
[Service]
Environment="http_proxy=http://<UserName>:<Password>@<HostName>:<Port>"
Environment="https_proxy=https://<UserName>:<Password>@<HostName>:<Port>"
Environment="no_proxy=gitlab.dadock.fujitsu.local,registry.dadock.fujitsu.local"
ExecStart=
ExecStart=/usr/bin/dockerd --insecure-registry registry.dadock.fujitsu.local
```

Execute the following command to reflect the setting.

```
# systemctl daemon-reload
# systemctl restart docker
```

4.2. When the initial settings in the Platform server are changed

After registering GitLab Runner and the initial settings(host suffix, port number or IP Address) of the platform server are changed, it is a must that the settings in the build service must also be changed.

4.2.1. hosts settings

In the hosts settings, edit the contents of the hostname suffix or the IP Address .
*Edit the hosts file with the use of Administrator privilege.

```
/etc/hosts
```

List 6. Example of hosts information settings (If in case the hostname suffix is set to [dadock.fujitsu.local], and DADock Platform server is [192.0.2.1])

```
192.0.2.1 gitlab.dadock.fujitsu.local
192.0.2.1 sonarqube.dadock.fujitsu.local
192.0.2.1 artifactory.dadock.fujitsu.local
192.0.2.1 portal.dadock.fujitsu.local
```

4.2.2. Runner Settings

Edit the settings of the GitLab Runner with the new hostname suffix, new port number, or IP Address.
*Edit the GitLab Runner with the use of Administrator privilege.
It is possible to ommit the port number when the port number is set as default, 80.

```
/etc/gitlab-runner/config.toml
```

Example of GitLab Runner settings (where the hostname suffix is set to [dadock.fujitsu.local],

the port number is set to [8081] and the IP address of DADock Platform server is [192.0.2.1])

```
[[runners]]
  name = "sample-shell"
  url = "http://gitlab.dadock.fujitsu.local:8081"
  token = "47f769d2e703c3baab514bfde5b323"
  executor = "shell"
  [runners.cache]



[[runners]]
  name = "sample-docker"
  url = "http://gitlab.dadock.fujitsu.local:8081"
  token = "47f769d2e703c3baab514bfde5b323"
  executor = "docker"
  [runners.docker]
    tls_verify = false
    image = "centos"
    privileged = false
    disable_cache = false
    volumes = ["/cache"]
    extra_hosts = ["gitlab.dadock.fujitsu.local:192.0.2.1", "registry.dadock.fujitsu.local:192.0.2.1",
"sonarqube.dadock.fujitsu.local:192.0.2.1", "artifactory.dadock.fujitsu.local:192.0.2.1"]
    shm_size = 0
  [runners.cache]
```

4.2.3. Docker

Edit the Docker settings with the new hostname suffix and the new port number.
*Edit the Docker settings file with the use of Administrator privilege.
It is possible to ommit the port number when the port number is set as default, 80.

```
/etc/systemd/system/docker.service.d/docker.conf
```

List 7. docker.conf(Example：If in case the host name of DADock Portal is [portal.dadock.fujitsu.local] and port number is [8081])

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd --insecure-registry registry.dadock.fujitsu.local:8081
```

Execute the following commands to reflect the changes in the Docker settings.

```
# systemctl daemon-reload
# systemctl restart docker
```

4.2.4. Gradle

Change the Gradle proxy settings only when the using proxy environment.
Refer to Gradle Proxy Settings to edit the file with the contents of the new hostname suffix.

4.3. Advanced usage of GitLab Runner

4.3.1. Registering multiple Runner

By repeating the procedures written in Registering GitLab Runner manual, multiple runners can be registered.
However, even if multiple runners are registered yet the settings in simultaneously executing job-count settings of Runner is not set, multiple jobs cannot be executed at the same time. (another job wil not run until the first job is finished running)

4.3.2. Simultaneously executing job count settings of Runner

Setting the count number of Runner that can simultaneously executes jobs in one build service.

1. Login to the platform server with the use of the applications such as Tera Term, and login as an Administrator.

2. Execute the following command to open the settings file of GitLab Runner.

```
# vi /etc/gitlab-runner/config.toml
```

config.toml

Example of only one registered Runner

```
concurrent = 1
check_interval = 0

[[runners]]
  name = "sample"
  url = "http://gitlab.sample.fujitsu.local"
  token = "47f769d2e703c3baab514bfde5b323"
  executor = "shell"
  [runners.cache]
```

3. In the field [concurrent = 1], input the desired maximum number of simultaneously executing jobs count and save the file.

config.toml

Example of desired maximum number of simultaneously executing job is 3

```
concurrent = 3
check_interval = 0

[[runners]]
  name = "sample"
  url = "http://gitlab.sample.fujitsu.local"
  token = "47f769d2e703c3baab514bfde5b323"
  executor = "shell"
  [runners.cache]
```

With these settings, the number of simultaneously executing job count of Runner is changed.

4.3.3. Usage of Runner tag

By setting the [tags] description of the [gitlab-ci.yml] file of the project, only the defined runner can execute the corresponding job.

gitlab-ci.yml

Example of executing the test_job only with the use of the tagged [docker] runner(Reference：Registering GitLab Runner)

```
stages:
  - test

test_job:
  type: test
  script:
    - ./test_script.sh
  tags:
    - docker
```

4.3.4. Scheduled execution of job build in GitLab

It is possible to execute a build process in the set time schedule by changing the scheduler settings of the CI/CD Schedule under the GitLab project and as well as changing the internal scheduler settings of the entire GitLab (Sidekiq).

CI/CD Schedule Settings of the GitLab project

1.In the project that needs to be executed in a schedule, click [CI / CD > Schedules].

2.Click the [New Schedule] button.

3.Refer to the table below and input the necessary fields required for the scheduling.

Item	Details
Description	Input the schedule description or explanation.

Item	Details
Interval Pattern	Set the desired schedule. If in case of custom settings, refer to [cron syntax].
Cron Timezone	Set the time zone. Example) For Japan, set [Asia/Tokyo] timezone.
Target Branch	Set the branch of the project that is set for scheduled execution.
Variables	(when needed) settings of the variable used in .gitlab-ci.yml file.
Activated	Checking this option would activate the scheduled job.

4.Click [Save pipeline schedule] button.

With these settings, the scheduling of build in GitLab is now complete.

Settings of the internal scheduler of the entire GitLab (Sidekiq)

The set time schedule in GitLab build scheduler will be processed through Sidekiq that is why it is a must that the time schedule settings will also be done in Sidekiq. (Sidekiq is executed in default every 19th minute of every hour.
For example, when the set time schedule in GitLab is 2 o'clock, the execution will occur at 2:19.)

1.Login as an administrator in DADock Platform server.

2.Add the following entry at the end of the file /opt/dadock/etc/gitlab/gitlab.rb to set that Sidekiq will be executed every minute.

```
gitlab_rails['pipeline_schedule_worker_cron'] = "* * * * *"
```

*There are no problems even if the set schedule for the execution of Sidekiq is the same only with the set schedule of the build process in Gitlab.

3.Execute the following commands to restart GitLab container and to reflect the changes made in Sidekiq.

```
# docker stop compose_gitlab_1
# docker start compose_gitlab_1
```

With these settings, the build process will now automatically execute in the set time schedule.

Setting the execution of the job only in the stated time schedule

When setting the schedule of job execution, each job automatically executes every push and also during the set time schedule.

By adding the entry [only: - schedules] in the .gitlab-ci.yml file, just like in the example below, build process will not be automatically executed whenever there is a push process from the user but will only be executed in the set time schedule.

*Each job is independent of one another so if there are multiple jobs set in the file, it is a must that the entry [only: - schedules] should be placed in every job.

List 8. gitlab-ci.yml

```
job_only-schedule:
  only:
    - schedules
  script:
    - echo "only schedule"
```

Furthermore, the job that is not executed will also not be displayed under [CI/CD > Pipelines].

For the limitations or restrictions of the automatic build execution, refer to the official documents of GitLab to know more about the details.

4.4. Resolution when the job in GitLab is not starting

When the job is in an indefinite loop or when the executed job became a runaway process, there is a chance that even if the job is cancelled from GitLab page, the next job executed will not start.

In that case, it is possible to resolve the problem by deleting the job process, that is executed by the GitLab Runner, in the build server.
This section will explain the procedure on how delete job process, that is executed by the GitLab Runner, in the build server

1. Use application such as Tera Term to login to the build server as an administrator.

2. Execute the following commands and the process whose owner is [gitlab-runner] will be displayed.

```
# ps -eo uname:32,pid,cmd | grep ^gitlab-runner
```

3. Execute the following command to delete the corresponding process that needs to be deleted.

```
# kill -9 <ProcessID>
```

V2_1806
Last Update 2018-06-28 04:38:57 UTC