

Technische Universität Wien

CI/CD Tutorial for Arrowhead Framework

Components, procedures for establishing and using CI/CD pipeline for the
Arrowhead Framework

Haris Isakovic, Peter Ketcher
8-21-2018

Contents

Motivation	3
Goals:	3
Jenkins	3
What is Jenkins?	3
Jenkins Master	4
Jenkins Master Requirements	4
Installation	5
Docker	7
Jenkins Plugins	8
Setup and Configuration	8
Access Control Mechanism	10
Jenkins Slave.....	13
Requirements.....	13
Hardware	13
Software.....	13
Installation	14
Docker Registry.....	17
Registry Deployment in the Lab	17
Portus Requirements	17
Software	18
Installation.....	18
Setup	20
Jenkins Pipelines	21
What is a Jenkins-Pipeline ?	21
Demo Jenkins-pipelines.....	22
Concept.....	23
Setup of the Jenkins-Master	23
Pull and Push of Docker-build-Images	24
Pipeline Build Arrowhead Server	26

Pipeline Build Arrowhead Client Demo.....	30
Pipeline Deploy Server	31
Pipeline Deploy Client	33
Pipeline Stop Server	34
Pipeline Stop Client	34
Shared Libraries	35
What are shared Libraries ?	35
Lab-Demo Shared Library.....	35
Dynamic Project-Parameters	38
Pipeline AddBuildTool	39
Pipeline Build-ArrowHead-ServerStack-Demo.....	42
Pipeline Deploy-Stack.....	44
Conclusion.....	46

Motivation

This Document is intended as:

- Protocol of the CD-demo created for the CPS-IoT Lab
- Guide through the installation and setup process of the CD Infrastructure
- Help in implementing basic Jenkins-Pipelines

Section denoted as *Protocol* records the actual implementation steps/actions/decisions,

Guide provides information and recommendations to reproduce/extend the process and

Help provides some background information to help clarify the motivation of the implementation decisions.

Goals:

- Install and Setup Jenkins-Master
- Install and Setup Jenkins-Slave
- Install and Setup Portus and the Docker-Registry
- Create shared maven cache on the Jenkins Master
- Script Jenkins-Pipeline to compile ArrowHead-Services, package ArrowHead-Services as docker-images and commit them to the docker-registry
- Script Jenkins-Pipeline to manage the deployment of Arrowhead-Services as docker-containers

Jenkins

What is Jenkins?

Jenkins is an open source automation server written in Java (source:wikipedia). It provides a variety of features for different SW integration, and deployment tasks. It is a highly customizable environment with a large selection of 3rd party plugins designed to accommodate needs of different development environments and projects. It is usually setup with a single master node which coordinates different CI/CD pipelines and number of slave nodes controlled by the master. In case of Arrowhead pipeline we have single Jenkins master that operates multiple stages of integration and deployment. The slaves can be added depending on the needs of a specific application.

Jenkins Master

It is the machine which runs the Jenkins-Server Software; the central, coordinating process which stores configuration, loads plug-ins, and renders the various user interfaces for Jenkins (source: <https://jenkins.io/doc/book/glossary/>).

Jenkins Master Requirements

Hardware

Minimum requirements: 256 MB of RAM and 1 GB of drive space.

Recommended for a small team: 1 GB+ of RAM 50 GB+ of drive space.

(source: <https://jenkins.io/doc/book/installing/>)

Protocol

Used Hardware:

- AWS-EC2 t2.micro instance, with 1 GB of RAM and 1 logical Core
- CPU (from lshw - <https://linux.die.net/man/1/lshw>)
- Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz
- EBS-Volume type gp2 (General purpose SSD) , Disk size 20 GiB

Guide

The requirements apply to the Jenkins-Software which by itself only provides the orchestration of tools. Individual tool requirements should be considered in addition to the above.

Software

- Java 8 runtime environments, both 32-bit and 64-bit versions are supported
- Since Jenkins 2.164 and 2.164.1, Java 11 runtime environments are supported.
- Older versions of Java are not supported, Java 9 and Java 10 are not supported, Java 12 is not supported.
- Jenkins project performs a full test flow with the following JDK/JREs:
OpenJDK JDK / JRE 8 - 64 bits

OpenJDK JDK / JRE 11 - 64 bits

(source: <https://jenkins.io/doc/administration/requirements/java/>)

Protocol

1. OS: Ubuntu 16.04.6 LTS, amd64 Xenial
2. Java version 1.8.0_201, Java RunTime version 1.8.0_201-b09, provided through the paket "openjdk-8-jdk-headless".

Help

The java requirements stated above apply to all components of the Jenkins system including Jenkins master, all types of agents, cli clients, and other components. Third party Jenkins-plugins,

Which implement most of the external tool integration, may have different java requirements as they not part of the jenkins-system. Java 8 was chosen to prevent incompatibilities with

Plug-ins written/last updated prior to the advent of Java 11.

Installation

Binaries of the Jenkins-Software LTS are available as packages for the Linux-distributions Debian,

OpenSuse and RedHat under <https://pkg.jenkins.io>. Other releases may be found at <https://jenkins.io/download/> .

Protocol

1. the shell used was "bash" (<http://manpages.ubuntu.com/manpages/xenial/man1/bash.1.html>)
2. commands that require root permissions where called using "sudo", a user account with unrestricted sudo-access (can call any command as root) was used
3. the packages were installed using the command-line tool "apt" (<http://manpages.ubuntu.com/manpages/xenial/man8/apt.8.html>)

```
#!/bin/bash

#first an Update of the Package Index was done:
sudo apt update

#OpenJDK 8 was installed
sudo apt install openjdk-8-jdk-headless

#the repository key for the Jenkins debian-repo was downloaded and added
as trusted key
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key
add -

#new apt source-list containing the reference to the Jenkins-debian-repo
was added
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \
/etc/apt/sources.list.d/jenkins.list'

#the package index was updated again
sudo apt update

#the jenkins-software was installed
sudo apt-get install jenkins

#the status of the jenkins-server was queried
systemctl status jenkins
```

Docker

Docker is a platform for developers and sysadmins to develop, deploy, and run applications with containers (source: <https://docs.docker.com/get-started/>), build on top of containerd (<https://containerd.io/>). The platform is not part of Jenkins-Software, therefore must be installed on each node where containers are needed. For the demo the “Docker Engine – Community” edition and the “Docker CLI” were used, both are freely available under a Apache License, Version 2.0 (<https://github.com/moby/moby/blob/master/LICENSE>).

Protocol

1. The docker-binaries are available as packages from the repository <https://download.docker.com/linux/ubuntu/> .
2. The shell used was “bash”
3. Commands that require root permissions where called using “sudo”, a user account with unrestricted sudo-access (can call any command as root) was used

```
#!/bin/bash

#install/check needed dependencies
sudo apt-get install \apt-transport-https \ca-
certificates \curl \gnupg-agent \software-properties-common

#fetchd and add the repository-key to the trusted keys
curl -fsSL
https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

#stable branch repository reference was added
sudo add-apt-repository \"deb
[arch=amd64] \
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable\"

#updated the package index
sudo apt-get update

#installed docker-ce engine, docker cli and the containerd-runtimes
sudo apt-get
install docker-ce docker-ce-cli containerd.io

#per default only the root user can use docker,
#in order for other user to have access
#they must be added to the group docker,
#on the Jenkins-Master that is the user jenkins.
sudo usermod -aG docker jenkins
```


Guide

The “bare” (without plug-ins) Jenkins-Software provides only one Project called “Free Style”. The project allows the definition of an automated sequence of one or more build-steps: execution of maven goal(s), run shell commands/scripts and run Windows Batch-File(s). This approach is not per se restrictive although all the functionality needed for the build process has to be implemented from scratch. Plug-ins extend Jenkins by providing additional Project-types, ready to use functions, etc. Note that plug-ins are not universally usable, some are tied to specific types of projects, extend other plug-ins, etc. Plug-ins dependencies are declared in files which are delivered with the plug-in. The Jenkins plug-in installer resolves the dependencies.

The following list shows the plug-ins needed for the Demo-Setup. Dependencies are not shown here.

Protocol

List of installed plugins:

- Locale (<https://plugins.jenkins.io/locale>)
- Folders (<https://plugins.jenkins.io/cloudbees-folder>)
- Folder Properties (<https://plugins.jenkins.io/folder-properties>)
- Pipeline (<https://plugins.jenkins.io/workflow-aggregator>)
- Git (<https://plugins.jenkins.io/git>)
- Role-based Authorization Strategy (<https://plugins.jenkins.io/role-strategy>)
- Job and Node ownership (<https://plugins.jenkins.io/ownership>)
- Authorize Project (<https://plugins.jenkins.io/authorize-project>)
- Docker (<https://plugins.jenkins.io/docker-plugin>)
- GitHub (<https://plugins.jenkins.io/github>)
- Build Timestamp (<https://plugins.jenkins.io/build-timestamp>)
- Node and Label parameter (<https://plugins.jenkins.io/nodelabelparameter>)
- Extended Choice Parameter (<https://plugins.jenkins.io/extended-choice-parameter>)

Setup and Configuration

After the initial install the Jenkins-Software starts its built-in web-server that listens per default on port 8080. All setup and configuration was done through the provided Web-UI.

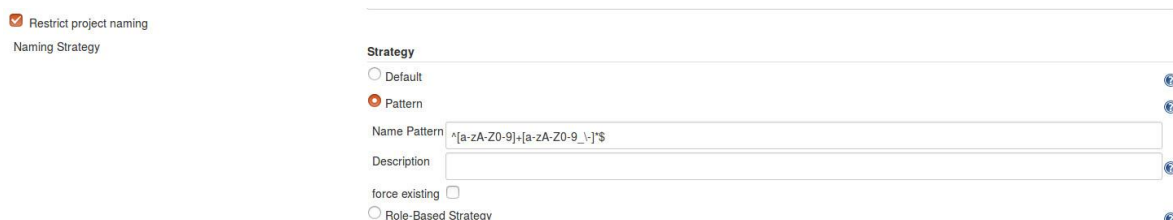
```
#!/bin/bash  
sudo systemctl restart jenkins
```

Protocol

1. The initial page of the Web-UI shows the locked jenkins-setup-wizard, the initial admin password is stored as plain-text in a file readable only by the jenkins-user and jenkins-group. The password was retrieved as sudoer.
2. After unlocking the setup-wizard, the wizard recommends plugins to be installed, this option was chosen and the first user (with admin rights) was created.
3. Jenkins was restarted
4. The plug-ins from the list (if missing) were installed as the newly created user.
5. Jenkins was restarted

Jenkins configuration:

1. The Jenkins Locale was set to “english”.
2. “Restrict project naming” option was chosen, the “Naming Strategy” set to “Pattern” with the “Name Pattern” set to “^[a-zA-Z0-9]+[a-zA-Z0-9_\-]*\$” (Java RegEx).



☒ Restrict project naming

Naming Strategy

Strategy

☐ Default

☒ Pattern

Name Pattern

Description

force existing ☐

☐ Role-Based Strategy

Figure 1 New Project

Help

The Jenkins-Software uses projects names as folder names, the names are used as arguments in command-line tools. Excluding characters like white-spaces prevents errors in scripts due to incorrect escaping of characters.

Guide

Permissions are sub-divided into categories: “Overall”, “Credentials”, “Manage ownership”, “Agent”, “Job” (Job and Folders), “Run”, “View” (dashboard / portal-like view), “SCM”, “Lockable Resources”.

Jenkins users have one or more roles which define their access permissions. Roles are defined for the scopes “Global”, “Project” and “Slave”. The “Global” scope defines the fundamental access policy, e.g. the “global build job” grants access to build any job. The “Project” scope adds more fine grained access control over individual jobs. The “Slave” scope works analogously for jenkins-slaves/nodes . The “Folders” plug-in adds the possibility to define folders which may contain jobs and/or folders, the hierarchy is a tree with a folder at its root and jobs as leafs. In addition jobs/folders inherit properties from their direct and indirect parents. The “Project” scope applies to both folders and jobs.

“Job and Node ownership” plug-in adds the possibility to define one primary owner and one or more secondary owners per folder/job and nodes.

Protocol

1. The security realm chosen was “Jenkins’ own user database”, user accounts were created through the Web-UI by the “admin” user and are managed by the Jenkins-Software. The Authorization was set to be “Role-Based Strategy”, together with the “Job and Node ownership” plug-in. Global Roles are “admin”, “superuser” and “user”.
2. The “admin” has unrestricted access.
3. The “superuser” has overall read access, can build/cancel/read any job. This role is geared towards the project manager who needs read/build access to all projects but may not alter their configurations.
4. The “user” has overall read access, can create agents. This role is intended for the developers who need to define, configure and build jobs.
5. The “Job and Node ownership” plug-in adds the possibility for user to become primary or secondary owners of projects (Jobs and Folders), this is used for the “Project” scope access control. The following macros are used : “@CurrentUserIsPrimaryOwner”, “@CurrentUserIsPrimaryOwner:1” (reuse of the first macro as macro-names per scope have to be unique), which select folders/jobs based on ownership. The other selection-criteria used are patterns (Java Regex) of job/folder names. The combination of both is used to implement the following policy:
“@CurrentUserIsPrimaryOwner” with pattern “^[^/]+”, matches a root folder/job of the current user, permission are create/delete/manageDomains/update/view Credentials, create/read Job (Job and Folder). The folder itself must be created by the “admin” user as only he has the overall right to create a root folder, the “admin” then sets the intended “user” as primary owner. Once the “user” is owner he can freely create Projects within this folder but may not delete or configure the folder itself, this prevents the “user” from locking-out himself as he cannot create

root folders.

"@CurrentUserIsPrimaryOwner:1" with pattern "^[/]+/.*", matches any folder/job that is not at the root. Here the primary owner has unrestricted access.

- The option "Ownership management policy on job modifications" was set to "Assign job creator as owner". When a "user" creates a job/folder he is automatically primary owner, as owner he has unrestricted access.

- For the "Slave" scope the following combinations are used:

"@CurrentUserIsOwner" with pattern "\. *", matches any slave the current user is owner of (primary or secondary). Permissions are Build/Connect Agent. This enables the sharing of build agents between users, an agent with several secondary owners.

"@CurrentUserIsPrimaryOwner" with pattern "\. *", matches any slave the current user is primary owner of. Permissions are unrestricted "Agent" and "Lockable Resources" access. This enables a developer to configure a build/test/deploy slave to his needs.

"@NoOwner" with pattern "\. *" matches any agent that has no owner. Permissions are Manage Ownership of Nodes. Together with the "Global" scope permission "create agents" this enables a "user" to create an agent and claim its ownership. Note that this is a work-around as the author has found no way to assign ownership of a newly created agent automatically.

Manage Roles

Global roles

Role	Overall	Credentials	Manage ownership	Agent	Job	Run	View	SCM	Lockable Resources																										
	Administer	Read	Create	Delete	ManageDomains	Update	View	Jobs	Nodes	Build	Cancel	Configure	Create	Delete	Disconnect	Provision	Build	Cancel	Configure	Create	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Configure	Create	Delete	Read	Tag	Reserve	Unlock
admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
supenuser	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
user	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

Role to add:

Add

Project roles

Role	Pattern	Credentials	Manage ownership	Agent	Job	Run	View	SCM	Lockable Resources																									
		Create	Delete	ManageDomains	Update	View	Jobs	Nodes	Build	Cancel	Configure	Create	Delete	Disconnect	Provision	Build	Cancel	Configure	Create	Delete	Discover	Move	Read	Workspace	Delete	Replay	Update	Configure	Create	Delete	Read	Tag	Reserve	Unlock
@CurrentUserIsPrimaryOwner	"^[/]+/.*"	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
@CurrentUserIsPrimaryOwner:1"	"^[/]+/.*"	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		

Role to add:

Pattern:

Add

Slave roles

Role	Pattern	Credentials	Manage ownership	Agent	Lockable Resources												
		Create	Delete	ManageDomains	Update	View	Jobs	Nodes	Build	Cancel	Configure	Connect	Delete	Disconnect	Provision	Reserve	Unlock
@CurrentUserIsOwner	"^[/]+/.*"	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
@CurrentUserIsPrimaryOwner	"^[/]+/.*"	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
@NoOwner	"^[/]+/.*"	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 2 Role Management interface Jenkins

Assign Roles

Global roles



























User/group	admin	superuser	user
 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 blue 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 Haris 	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 johannes 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 Leo 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 Lukas 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 Michael 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 Peter 	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
 testa 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
 Anonymous 	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 3 Global User-Roles

Item roles

User/group	@CurrentUserIsPrimaryOwner	@CurrentUserIsPrimaryOwner:1
 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 blue 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Haris 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 johannes 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Leo 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Lukas 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Matthias 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Michael 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 testa 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Anonymous 	<input type="checkbox"/>	<input type="checkbox"/>

User/group to add

Node roles

User/group	@CurrentUserIsOwner	@CurrentUserIsPrimaryOwner	@NoOwner
 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 blue 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Haris 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 johannes 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Leo 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Lukas 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Matthias 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Michael 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 testa 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
 Anonymous 	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 4 User-Item/Node Roles

Help

Although misleading the “Global” scope overall read permission does not imply unrestricted read access to all projects as more fine-grained permissions are defined in other scopes. Note that it is needed as basis for the other scopes, without overall read the user has no read access at all.

Jenkins Slave

A Jenkins slave node is a machine which is part of the Jenkins environment and capable of executing Pipelines or Projects (<https://jenkins.io/doc/book/glossary/>), a slave is a node that is not master.

Requirements

Hardware

Jenkins nodes are typically hardware agnostic as long as they can run necessary software to accommodate Jenkins communication and control protocols. The agent’s configuration depends on the builds it will be used for and on the tools required by the same builds (<https://jenkins.io/doc/book/hardware-recommendations/>).

Protocol

In the Lab-Demo two slaves nodes where used:

- A build-slave, a AWS-EC2 t2.micro instance, with 1 GB of RAM and 1 logical Core and EBS-Volume type gp2 (General purpose SSD) , Disk size 15 GB.
- A deploy-slave, a Raspberry Pi 3 Model B Rev 1.2 with a 16 GB SD card.

Software

Protocol

Build-Slave:

- OS : Ubuntu 16.04.6 LTS, amd64 xenial
- Software-Packages: OpenJDK 8 and Docker-CE (same as on the Master).

Deploy-Slave:

- OS: Raspbian GNU/Linux 9.9 (stretch)

- Software-Packages: OpenJDK 8 and Docker-CE

Installation

Protocol

1. Installing Java and Docker on the Build-Slave was done in the same way as on the Master
2. Deploy-Slave:

```
#!/bin/bash

#Docker was installed using a script
#found at https://github.com/docker/docker-install .
curl -fsSL get.docker.com -o get-docker.sh && sh get-docker.sh

#Java was installed with apt.
sudo apt install openjdk-8-jdk-headless
```

Help

The distributed mode of Jenkins is fundamentally a "Master+Agent" architecture. The master is designed to do co-ordination and provide the GUI and API endpoints, and the Agents are designed to perform the work, agent and Jenkins master need to establish a bi-directional communication link (e.g. TCP/IP socket) in order to operate (<https://wiki.jenkins.io/display/JENKINS/Distributed+builds>).

Protocol

The communication link between Master and Slave is established through the Jenkins Remoting (<https://jenkins.io/projects/remoting/>). The link was established Slave → Master via the Agent executable.

1. - "Configure Global Security" → "Agents" on the Jenkins Master was configured with a fixed "TCP port for inbound agents".



Screenshot 2: Agent Port

2. in "Manage Jenkins" → "Manage Nodes" a new Node (Slave/Agent) was added with:
 - "Remote root directory" set to ".", "Launch method" set to "Launch agent by connecting it to the master".

Remote root directory

⚠ Are you sure you want to use a relative path for the FS root? Note that relative paths require that you can assure that the selected launcher provides a consistent current working directory. Using an absolute path is highly recommended.

Labels

Usage

Launch method

Disable WorkDir ☐

Custom WorkDir path

Internal data directory

Fail if workspace is missing ☐

Availability

Figure 5 Build-Slave

- After creation the Node is listed in “Manage Jenkins” → “Manage Nodes”, on its page the agent executable was downloaded under “Run from agent command line”, “agent.jar”. The agent executable was deployed to all Slaves.



Agent testa-deploy-x86

Connect agent to Jenkins one of these ways:

- Launch agent from browser
- Run from agent command line:

```
java -jar agent.jar -jnlpUrl https://jenkins-cpsiot-2018.pii.at/computer/testa-deploy-x86/slave-agent.jnlp -secret
-workDir "."
```

Figure 6 Agent connection interface

- For the connection to be re-established upon reboots, a systemd-service (<https://wiki.ubuntu.com/SystemdForUpstartUsers>, <https://wiki.debian.org/systemd>) was added:

```
# /etc/systemd/system/jenkins-slave-agent.service

[Unit]
Description=My service
After=network.target
[Service]
ExecStart=/bin/bash launch-jenkins-slave-agent.sh
WorkingDirectory=path-to-script
StandardOutput=inheritStandardError=inherit
Restart=always
User= username-that-runs-the-agent
[Install]
WantedBy=multi-user.target
```


the script "path-to-script/launch-jenkins-slave-agent.sh":

```
#!/bin/bash
#launch-jenkins-slave-agent.sh

java -jar path-to-executable/agent.jar \
-jnlpUrl jenkins-url/computer/slave-name/slave-agent.jnlp \
-secret secretkey \
-workDir "."
```

5. Started the service:

```
#!/bin/bash
#reload daemon
sudo systemctl daemon-reload
#activate service to start at boot
sudo systemctl enable jenkins-slave-agent.service
#start service
sudo systemctl start jenkins-slave-agent.service
```

1. Finally on both slaves :

```
#!/bin/bash
sudo usermod -aG docker username-that-runs-the-agent
```

Help

Setting the working directory to the current directory (".") delegates the path definition to the directory in which the agent executable is effectively started.

Docker Registry

The Registry is a stateless, highly scalable server side application that stores and lets you distribute Docker images. The Registry is open-source, under the permissive Apache license (source: <https://docs.docker.com/registry/>).

Registry Deployment in the Lab

The registry does only provide minimal access control out-of-the-box through `htpasswd`, this is hardly adequate for a private registry shared across teams/multiple users. The chosen solution for the Lab was Portus: an open source authorization service and user interface for the next generation Docker Registry (source: <http://port.us.org/>), created and maintained by the SUSE team. The recommended and used way to deploy Portus is with its official Docker-Image found at <https://hub.docker.com/r/opensuse/portus/> (source: <http://port.us.org/docs/deploy.html>).

Portus Requirements

Hardware

To the best knowledge of the author the official Docker-Documentation does not specify general hardware requirements for running containers, rather the requirements for containers depend primarily on the applications running inside. The applications are distributed as images which are made for a specific kernel and architecture (e.g., Linux-kernel for amd64), docker-images do not include a kernel. The kernel is provided by docker-host-machine where the image will actually be used to create a container.

Protocol

Image kernel-architecture: "Linux-amd64".

The disk space requirements for the docker-images used:

- portus-image ~ 250 MB,
- docker-registry-image ~ 30 MB,
- mariadb-image ~ 310 MB,
- nginx-image ~ 20 MB.

Selected hardware:

AWS-EC2 t2.micro instance, with 1 GB of RAM and 1 logical Core and EBS-Volume type gp2 (General purpose SSD) , Disk size 20 GB.

With the 20 GB Disk, about 14 GB where available for storing the images needed for the Lab-Deployment, with a Java-base-image for ARM of 70 MB and a mysql-server-image for ARM of 110 MB in size. The hardware could handle requests from 1-2 users simultaneously without problems.

Help

Application data is not considered in Docker-Image sizes, e.g. for the Java-base-image the size of the Java-byte-code must be considered in addition.

Software

Protocol

1. OS: Ubuntu 16.04.6 LTS, amd64 xenial
2. Software-Packages: Docker-CE (same as on the Master)
3. Additional binaries: docker-compose (<https://github.com/docker/compose>)
4. Docker-Images: portus-image (<https://hub.docker.com/r/opensuse/portus>), nginx:alpine (https://hub.docker.com/_/nginx), docker-registry (https://hub.docker.com/_/registry), mariadb (https://hub.docker.com/_/mariadb).

Installation

Guide

(Docker) Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services (source: <https://docs.docker.com/compose/>).

Protocol

1. installed Docker-CE
2. installed docker-compose (<https://docs.docker.com/compose/install/#install-compose>) :

```
#!/bin/bash

#downloaded the binarysudo curl -L \
"https://github.com/docker/compose/releases/download/1.24.1/docker-compose-\
$(uname -s)-$(uname -m)" \
-o /usr/local/bin/docker-compose

#added execution permission to the binary
sudo chmod +x /usr/local/bin/docker-compose

#added link to the binary
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

The definition for Portus-deployment was taken from an example found in the suse-portus github repository (<https://github.com/SUSE/Portus/blob/master/examples/compose/docker-compose.yml>).
The setup consists of

- two instances (docker-containers) of the portus-image,
- one instance of nginx:alpine,
- one instance of the image of the docker-registry,
- one instance of mariadb.

In this setup communication is always encrypted between Portus and the Registry containers. The TLS/SSL certificate used was certified by “Let’s Encrypt” (<https://letsencrypt.org/>).

Steps:

1. Checkout of the repository:

```
#!/bin/bash

git clone https://github.com/SUSE/Portus.git
```

2. in the environement-file “path-to-checkout/examples/compose/.env”:
the property “MACHINE_FQDN=your-fqdn” was set to the fully qualified domain name (FQDN) the certificate was issued for.
3. in the nginx configuration-file:
“path-to-checkout/examples/compose/nginx/nginx.conf” the “server_name” (existing value was “172.17.0.1”) was set to the FQDN.
4. the “Let’s Encrypt” key (“privkey.pem”) and certificate (“fullchain.pem”) files where copied to and renamed in “path-to-checkout/examples/compose/secrets”:

```
#!/bin/bash

mv privkey.pem portus.key && mv fullchain.pem portus.crt
```

5. the containers where started using “Detached mode” (Run containers in the background):
in the directory “path-to-checkout/examples/compose”:

```
#!/bin/bash
docker-compose up -d
```

Help

The service definition YAML-file is “path-to-checkout/examples/compose/docker-compose.yml” and was not changed.

Setup

Protocol

1. The setup was done through the Portus-Web-UI. In the example-configuration self-sign-up for users is activated, the first user to sign-up has “Admin” permissions.
2. signed-up as first user
3. registry setup:
“Name”: “registry”, “Hostname”: FQDN, “Use SSL”: checked.



Registries				
Name	Hostname	External hostname	SSL	Reachable
registry	docker-registry-cpsiot-2018.pii.at			 Edit

Figure 7 Registry

4. To disable self-sign-up for users, a switch to the host-machine shell was needed, stopped and removed the containers:

```
#!/bin/bash

#directory "path-to-checkout/examples/compose"
docker-compose down
```

5. Added the property “- PORTUS_SIGNUP_ENABLED=false”:

```
#file: path-to-checkout/examples/compose/docker-compose.yml
services:
    (...)
    portus:
        (...)
        environment:
            (...)
            - PORTUS_SIGNUP_ENABLED=false
```

started the containers:

```
#!/bin/bash

#directory "path-to-checkout/examples/compose"
docker-compose up -d
```

6. Created non “Admin” user accounts for the developers.

Help

As in the example service-definition Application-data is persisted as docker-volumes, which are independent from containers, the “docker-compose down” command only removes the containers not the volumes.

Jenkins Pipelines

The Jenkins Pipeline-plug-in adds, among others, a new Project-type called “Pipeline”.

What is a Jenkins-Pipeline ?

In General a (continuous delivery/integration CD/CI) pipeline is an automated expression of your process for getting software from version control right through to your users and customers (source: <https://jenkins.io/doc/book/pipeline/>). A Jenkins-Pipeline is an implementation as framework of that concept. Jenkins-Pipelines are written in a Pipeline domain-specific language (DSL) syntax (<https://jenkins.io/doc/book/pipeline/syntax/>). The definition of a Jenkins Pipeline is written into a text file (called a `Jenkinsfile`) which in turn can be committed to a project’s source

control repository (source: <https://jenkins.io/doc/book/pipeline/>).

The Syntax comes in two flavors:

- The Declarative Pipeline is a relatively recent addition to Jenkins Pipeline which presents a more simplified and opinionated syntax on top of the Pipeline sub-systems,
- The Scripted Pipeline is effectively a general-purpose DSL built with Groovy (<http://groovy-lang.org/syntax.html>) (source: <https://jenkins.io/doc/book/pipeline/syntax/>).

Demo Jenkins-pipelines

Help

Jenkins-Pipelines are composed of Stages. Stages are composed of steps, they tell Jenkins what to do and serve as the basic building block. Stages of the same Pipeline may be executed on different machines (Jenkins nodes). This distribution of stages may be defined in the Jenkinsfile or the Jenkins-Master decides at runtime of the Pipeline. Note because every node has its own copy of the workspace (working directory of the Pipeline run) files that are created during some step are not available automatically on a different node.

The Pipeline-step “stash” saves a set of files for later use in the same build (Pipeline run), “unstash” restores the files on another node/workspace. Environment variables can be set globally (whole Jenkins every Pipeline), globally per Pipeline (available to every stage) or per stage.

Pipelines can be parametrized, these parameters are not defined in the Jenkinsfile but in the Project-definition. When starting a Pipeline through the Jenkins-WebUI the parameter values are selected and set for this specific build (Pipeline run), the selection is part of the build-log.

Concept

Automate repetitive tasks in the development/deployment cycle of CPS-IoT Lab Projects. These tasks are building the Arrowhead-Server/Client stack from source, generating the necessary configuration files, packaging binaries and configuration files as Docker-Images, store the images and deploy/stop a stack of container(s).

Setup of the Jenkins-Master

Protocol

→ Global environment-variables:

“Manage Jenkins” → “Configure System” in “Global properties” / “Environment variables”

“DOCKER_REPO” = “*FQDN of the private docker-registry*”,

“DOCKER_USER_NAMESPACE” = “*docker name-space in the private registry*”.

The screenshot shows the 'Global properties' configuration page in Jenkins, specifically the 'Environment variables' tab. It lists several environment variables with their names and values, and a 'Delete' button for each. The variables are:

Name	Value
ARCH_AMD64	amd64
ARCH_ARM	arm
DOCKER_BUILD_USER_NAMESPACE	build
DOCKER_REPO	docker-registry-cpsiot-2018.pil.at
DOCKER_USER_NAMESPACE	demo_user

'Screenshot 3: Global Environment Variables

→ “Manage Jenkins” → “Manage Nodes” → “slave name to configure” → “Configure” under “Labels” = “arm build x86 cpsiot”

The screenshot shows the 'Manage Nodes' configuration page in Jenkins, specifically the 'Configure' tab for a node named 'jenkins-slave1-cpsiot-2018'. The fields are filled with the following values:

Field	Value
Name	jenkins-slave1-cpsiot-2018
Description	
# of executors	1
Remote root directory	./
Labels	arm build x86 cpsiot

Screenshot 4: Labels Build-Slave

➔ Set environment-variables for Build-Slave

`"JAVA_IMAGE_BUILD_ORACLE" = "my-java-build-oracle",`

`"MY_SQL_IMAGE_BUILD" = "my-sql-server-build"`



The screenshot shows the 'Node Properties' configuration page for a Jenkins slave node. At the top, there are two checkboxes: 'Disable deferred wipeout on this node' (unchecked) and 'Environment variables' (checked). Below these, a section titled 'List of variables' contains a table of environment variables. Each row has a 'Name' field, a 'Value' field, and a 'Delete' button. The variables listed are: ARCHITECTURE (value: arm), JAVA_IMAGE_BUILD (value: my-java-build), JAVA_IMAGE_BUILD_ORACLE (value: my-java-build-oracle), and MY_SQL_IMAGE_BUILD (value: my-sql-server-build). There is a small blue information icon to the right of the 'Delete' button for the third variable.

Name	Value	Delete
ARCHITECTURE	arm	Delete
JAVA_IMAGE_BUILD	my-java-build	Delete
JAVA_IMAGE_BUILD_ORACLE	my-java-build-oracle	Delete
MY_SQL_IMAGE_BUILD	my-sql-server-build	Delete

Screenshot 5: Node Properties Build Slave

➔ Set labels for Deploy-Slave

`"Manage Jenkins" → "Manage Nodes" → "slave name to configure" → "Configure" under "Labels" = "raspberrypi arm deploy"`

➔ added Developer-Slave, a Jenkins-Node that a developer may connect his machine to, typically this machine will have a x86 architecture

➔ Set labels for Developer-Slave

`"Manage Jenkins" → "Manage Nodes" → "slave name to configure" → "Configure" under "Labels" = "x86 deploy"`

Help

The node-name is unique, labels can be used to group nodes, e.g. nodes having a `"build"` label are in the same group or nodes having a `"arm"` label indicate that the connected machine has an ARM architecture. Jenkins does not check or enforce this, its a convenient way to distribute build-stages to eligible nodes: Pipelines have an agent section

(<https://jenkins.io/doc/book/pipeline/syntax/#agent>) which specifies where the entire Pipeline, or a specific stage, will execute in the Jenkins environment. The agent may be defined by label, e.g.

`"agent{ label "master"}"` is the Jenkins-Master or

`"agent{ label "build && cpsiot"}"` is an agent with labels `"build"` and `"cpsiot"`.

Pull and Push of Docker-build-Images

The private registry is used also to store docker-images that are the basis for packaging, e.g. when packaging the Arrowhead-Service-Registry the image that will contain the Java-byte-code needs a JRE. These base-Images can be build from scratch

(<https://docs.docker.com/develop/develop-images/baseimages/>) or downloaded from DockerHub

(<https://hub.docker.com/>). As the primary image-source should be the private docker-registry, downloaded images can be tagged ("renamed") (<https://docs.docker.com/engine/reference/commandline/tag/>) and pushed (stored) to the private registry. This brings also the advantage of a custom naming-scheme for images vs a given from DockerHub. In Addition DockerHub might not be available at times or simply some images might not be available anymore at some point.

Protocol

- ➔ on the Jenkins-Master shell with an user-account that is a member of the group "docker": tag and push the build-images:

```
#!/bin/bash

#tag and push an image with OpenJDK 8 for Linux-arm
docker pull arm32v7/openjdk:8-jdk-alpine
docker tag arm32v7/openjdk:8-jdk-alpine \registry-fqdn/namespace/my-java-build-oracle-arm:latest
docker login https://registry-fqdn
docker push registry-fqdn/namespace/my-java-build-oracle-arm:latest
docker logout https://registry-fqdn

#tag and push an image with MySQL Server for Linux-arm
docker pull beercan1989/arm-mysql:5.7.22
docker tag arm32v7/openjdk:8-jdk-alpine \
registry-fqdn/namespace/my-sql-server-build-arm:latest
docker login https://registry-fqdn
docker push registry-fqdn/namespace/my-sql-server-build-arm:latest
docker logout https://registry-fqdn

#tag and push an image with OpenJDK 8 for for Linux-amd64
docker pull openjdk:8u191-jdk-alpine
docker tag openjdk:8u191-jdk-alpine \registry-fqdn/namespace/my-java-build-oracle-x86:latest
docker login https://registry-fqdn
docker push registry-fqdn/namespace/my-java-build-oracle-x86:latest
docker logout https://registry-fqdn

#tag and push an image with MySQL Server for Linux-AMD64
docker pull mysql:5.7.27
docker tag mysql:5.7.27 \registry-fqdn/namespace/my-sql-server-build-x86:latest
docker login https://registry-fqdn
docker push registry-fqdn/namespace/my-sql-server-build-x86:latest
docker logout https://registry-fqdn
```

Help

The full naming-scheme of a docker image is :

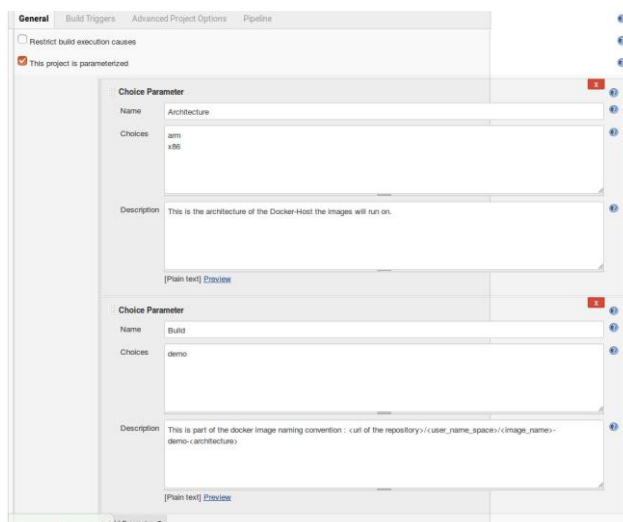
fqdn-of-the-registry/repository-name/image-name:tag.

Note when the *fqdn-of-the-registry* is omitted, the docker-cli automatically assumes DockerHub as registry. Also repositories might have the same image for several different *Kernel-Architecture* (e.g. Linux-amd64), in that case the docker-cli pulls automatically an image matching the docker-host-machine, e.g. pulling “openjdk:8u191-jdk-alpine” on the Jenkins-Master fetches a Linux-am64 image, on a raspberry pi the same command would fetch a Linux-arm image. Note when pulling from the private registry the image architecture is identified through the image-name, e.g. “x86”.

Pipeline Build Arrowhead Server

Protocol

- Environment-variables (Master):
“DOCKER_REPO” = “*FQDN of the private docker-registry*”,
“DOCKER_USER_NAMESPACE” = “*docker name-space in the private registry*”.
- Environment-variables (Build-Slave):
“JAVA_IMAGE_BUILD_ORACLE” = “my-java-build-oracle”,
“MY_SQL_IMAGE_BUILD” = “my-sql-server-build”
- Jenkins Node Labels (Build-Slave):
“build *Architecture-to-build* cpsiot”
- Credentials used: “demo-user” is username and password of a portus non “Admin” account with push and pull access to the “DOCKER_USER_NAMESPACE”
- Pipeline Parameters:
“Architecture” = “arm” or “x86”, specifies the docker-image architecture to use for packaging, used in the docker-image naming scheme, matches labels (Jenkins Slave),
“Build” = “demo”, is used in the docker-image naming scheme



Screenshot 6: Simple Static Choice Parameters

- **Repositories used:**
https://github.com/tuw-cpsg/arrowhead_devops.git /
branch:arrowhead_server_stack_demo,
it contains the Arrowhead-sources (Arrowhead Framework G4.0),
Jenkinsfile and scripts to generate the Docker-Images and configuration files.
- **Jenkins nodes:** Master, jenkins-slave1-cpsiot-2018
- **Docker-Images used:**
'maven:3-alpine' (https://hub.docker.com/_/maven) Linux-x86-64,
'beercan1989/arm-mysql:5.7.22' Linux-ARM
(<https://hub.docker.com/r/beercan1989/arm-mysql>),
'arm32v7/openjdk:8-jdk-alpine' Linux-ARM
(<https://hub.docker.com/r/arm32v7/openjdk/>),
'mysql:5.7.27' Linux-x86-64 (https://hub.docker.com/_/mysql),
'openjdk:8u191-jdk-alpine' Linux-x86-64 (https://hub.docker.com/_/openjdk).
- **Pipeline Syntax Style:** Declarative Pipeline.

Pipeline Stages:

1. "Create Maven Cache" (Master):
creates a docker-volume on the first run, this volume is used as maven cache, subsequent runs will skip the creation as the volume already exists,
2. "Build" (Master):
builds the Java-byte-code using maven (from the root pom.xml), the build runs inside a docker-container ("agent{ docker{...} }") from the image 'maven:3-alpine', the volume from Stage 1 is mounted as maven-cache, the Java-byte-code is stored in the workspace,
3. "Stash Artifacts" (Master):
stash the Java-byte-code from the local workspace for later use,
4. "Generate My-Sql Startup SQL" (jenkins-slave1-cpsiot-2018):
generates a SQL-script from environment-variables,
5. "Dockerize My-Sql-Server" (jenkins-slave1-cpsiot-2018):
 1. creates a Dockerfile which contains the definition of the the Mysql-Server image from
"*DOCKER_REPO/DOCKER_USER_NAMESPACE/MY_SQL_IMAGE_BUILD-Architecture:latest*" base-image, the SQL-script from Stage 4 is packaged as init-script (will be executed when a container from the image will be created)
"*COPY ./init_db.sql /docker-entrypoint-initdb.d/*" (Dockerfile),
 2. the new image is created and pushed to the private registry as
"*DOCKER_REPO/DOCKER_USER_NAMESPACE/my-sql-server-Build-Architecture:latest*" and
"*DOCKER_REPO/DOCKER_USER_NAMESPACE/my-sql-server-Build-Architecture:build-nr*"
6. "Generate Application Properties" (jenkins-slave1-cpsiot-2018):
the Arrowhead "app.properties" files are created from environment-variables,

7. "Dockerize Registry SQL" (jenkins-slave1-cpsiot-2018):

1. creates a Dockerfile which contains the definition of the the Arrowhead Service-registry image from

```
"DOCKER_REPO/DOCKER_USER_NAMESPACE/JAVA_IMAGE_BUILD_ORACLE-
Architecture:latest" base-image, the Java-byte-code and "app.properties" are
copied into the image on creation (directories are relative to
"$WORKSPACE/serviceregistry_sql"), "ENTRYPOINT" sets the command to execute on
container-startup
" COPY ./target/ /arrowhead/serviceregistry_sql/
COPY ./config/ /arrowhead/serviceregistry_sql/config/
WORKDIR/arrowhead/serviceregistry_sql
ENTRYPOINT ["java", "-cp", "lib/*:*", "-jar", "serviceregistry_sql-
4.0.jar", "-d", "-daemon" ]" (Dockerfile)
```

2. the new image is created and pushed to the private registry as

```
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-registry-sql-Build-
Architecture:latest" and
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-registry-sql-Build-
Architecture:build-nr"
```

8. "Dockerize Authorization" (jenkins-slave1-cpsiot-2018):

same steps as 7 for the Arrowhead Authorization Service

```
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-authorization-Build-
Architecture:latest"
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-authorization-Build-
Architecture:build-nr",
```

9. "Dockerize Gateway" (jenkins-slave1-cpsiot-2018):

same steps as 7 for the Arrowhead Gateway Service

```
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-gateway-Build-Architecture:latest"
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-gateway-Build-Architecture:build-nr",
```

10. "Dockerize Event-Handler" (jenkins-slave1-cpsiot-2018),

same steps as 7 for the Arrowhead Event-Handler Service

```
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-eventhandler-Build-
Architecture:latest"
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-eventhandler-Build-
Architecture:build-nr",
```

11. "Dockerize Gate-Keeper" (jenkins-slave1-cpsiot-2018),

same steps as 7 for the Arrowhead Gate-Keeper Service

```
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-gatekeeper-Build-Architecture:latest"
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-gatekeeper-Build-Architecture:build-
nr",
```

12. "Dockerize Orchestrator" (jenkins-slave1-cpsiot-2018).

same steps as 7 for the Arrowhead Orchestrator Service

```
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-orchestrator-Build-
```

```
Architecture:latest"
"DOCKER_REPO/DOCKER_USER_NAMESPACE/my-orchestrator-Build-
Architecture:build-nr".
```

Pipeline Build Arrowhead Client Demo

Protocol

- **Environment-variables (Master):**
`"DOCKER_REPO" = "FQDN of the private docker-registry",`
`"DOCKER_USER_NAMESPACE" = "docker name-space in the private registry".`
- **Environment-variables (Build-Slave):**
`"JAVA_IMAGE_BUILD_ORACLE" = "my-java-build-oracle",`
`"MY_SQL_IMAGE_BUILD" = "my-sql-server-build"`
- **Jenkins Node Labels (Build-Slave):**
`"build Architecture-to-build cpsiot"`
- **Credentials used:** "demo-user" is username and password of a portus non "Admin" account with push and pull access to the "DOCKER_USER_NAMESPACE"
- **Pipeline Parameters:**
`"Architecture" = "arm" or "x86", specifies the docker-image architecture to use for packaging, used in the docker-image naming scheme, matches labels (Jenkins Slave),`
`"Build" = "demo", is used in the docker-image naming scheme`
- **Repositories used:**
https://github.com/tuw-cpsg/arrowhead_devops.git / branch:arrowhead_client_demo,
it contains the Arrowhead-sources for the Publisher and Subscriber,
Jenkinsfile and scripts to generate the Docker-Images and configuration files.
- **Jenkins nodes:** Master, jenkins-slave1-cpsiot-2018
- **Docker-Images used:**
'maven:3-alpine' (https://hub.docker.com/_/maven) Linux-x86-64,
'arm32v7/openjdk:8-jdk-alpine' Linux-ARM
(<https://hub.docker.com/r/arm32v7/openjdk/>),
'mysql:5.7.27' Linux-x86-64 (https://hub.docker.com/_/mysql),
'openjdk:8u191-jdk-alpine' Linux-x86-64 (https://hub.docker.com/_/openjdk).
- **Pipeline Syntax Style:** Declarative Pipeline.

Pipeline Stages:

1. "Create Maven Cache" (Master):
same step as for Build the Arrowhead-Server stack,
2. "Build" (Master):
same step as for Build the Arrowhead-Server stack,

3. "Stash Artifacts" (Master):
essentially same step as for Build the Arrowhead-Server stack to stash the Publisher and Subscriber Java-byte-cpde,
4. "Generate Application Properties"(jenkins-slave1-cpsiot-2018):
same step as for Build the Arrowhead-Server stack, "app.properties" files for Publisher and Subscriber,
5. "Dockerize Arrowhead Publisher"(jenkins-slave1-cpsiot-2018):
same step as for Build the Arrowhead-Server stack for the Publisher
`"DOCKER_REPO/DOCKER_USER_NAMESPACE/publisher1-cloud1-Build-Architecture:latest"`
`"DOCKER_REPO/DOCKER_USER_NAMESPACE/publisher1-cloud1-Build-Architecture:build-nr",`
6. "Dockerize Arrowhead Subscriber"(jenkins-slave1-cpsiot-2018):
same step as for Build the Arrowhead-Server stack for the Publisher
`"DOCKER_REPO/DOCKER_USER_NAMESPACE/subscriber1-cloud1-Build-Architecture:latest"`
`"DOCKER_REPO/DOCKER_USER_NAMESPACE/subscriber1-cloud1-Build-Architecture:build-nr".`

Pipeline Deploy Server

Protocol

- Environment-variables (Master):
`"DOCKER_REPO" = "FQDN of the private docker-registry",`
`"DOCKER_USER_NAMESPACE" = "docker name-space in the private registry".`
- Environment-variables (Deploy-Slave):
`"ARCHITECTURE" = "architecture-of-the-slave",`
- Credentials used: "demo-user" is username and password of a portus non "Admin" account with pull access to the "DOCKER_USER_NAMESPACE"
- Pipeline Parameters:
`"My_SQL_Server_Version" = "latest"` , specifies tag of the Mysql-docker-image to deploy,
`"Registry_Version" = "latest"` , specifies tag of the Arrowhead Service-Registry docker-image to deploy,
`"Authorization_Version" = "latest"` , specifies tag of the Arrowhead Authorization docker-image to deploy,
`"Gateway_Version" = "latest"` , specifies tag of the Arrowhead Gateway docker-image to deploy,
`"Gatekeeper_Version" = "latest"` , specifies tag of the Arrowhead Gatekeeper docker-image to deploy,
`"Orchestrator_Version" = "latest"` , specifies tag of the Arrowhead Orchestrator docker-image to deploy,

"EventHandler_Version" = "latest" , specifies tag of the Arrowhead Eventhandler docker-image to deploy,

"Node" = "list-of-deploy-node", contains a list of the available Jenkins-Nodes for deployment

- Repositories used:
https://github.com/tuw-cpsg/arrowhead_devops.git / branch:deployment_demo, it contains the Jenkinsfile (Deploy_Server_Stack) for this Pipeline.
- Jenkins nodes: user-selected jenkins deploy-slave
- Pipeline Syntax Style: Declarative Pipeline.

Pipeline Stages:

1. "Deploy Server Stack" (selected deploy-slave):

The steps use "sh: Shell Script" (from the "Pipeline: Nodes and Processes" plug-in <https://plugins.jenkins.io/workflow-durable-task-step>) to execute docker-cli commands,the command options are defined in environment-variables, the image tag to use comes from the parameter value:

- a) stop and remove any existing containers (MySQL-Server, Service-Registry, Authorization, Gateway, Event-Handler, Gate-Keeper, Orchestrator)
- b) remove Arrowhead Service container network
- c) remove all unused containers, networks, images (both dangling and unreferenced)
- d) remove all unused local volumes. Unused local volumes are those which are not referenced by any containers
- e) add a new container network
- f) start the Mysql-Sever container
- g) sleep for 10 seconds (" sh "sleep 10" ")
- h) start the Arrowhead Service-Registry container,
- i) sleep for 10 seconds
- j) start the Arrowhead Authorization container
- k) sleep for 10 seconds
- l) start the Arrowhead Gateway container
- m) sleep for 10 seconds
- n) start the Arrowhead Event-Handler container
- o) sleep for 10 seconds
- p) start the Arrowhead Gate-Keeper container
- q) sleep for 10 seconds
- r) start the Arrowhead Orchestrator container

- s) sleep for 10 seconds
- t)

Pipeline Deploy Client

Protocol

- Environment-variables (Master):
`"DOCKER_REPO" = "FQDN of the private docker-registry",`
`"DOCKER_USER_NAMESPACE" = "docker name-space in the private registry".`
- Environment-variables (Deploy-Slave):
`"ARCHITECTURE" = "architecture-of-the-slave",`
- Credentials used: "demo-user" is username and password of a portus non "Admin" account with pull access to the "DOCKER_USER_NAMESPACE"
- Pipeline Parameters:
`"Publisher_Version" = "latest",` specifies tag of the Arrowhead Publisher-docker-image to deploy,
`"Subscriber_Version" = "latest",` specifies tag of the Arrowhead Subscriber docker-image to deploy,
`"Node" = "list-of-deploy-node",` contains a list of the available Jenkins-Nodes for deployment
- Repositories used:
https://github.com/tuw-cpsg/arrowhead_devops.git / branch:deployment_demo,
it contains the Jenkinsfile (Deploy_Client_Stack) for this Pipeline.
- Jenkins nodes: user-selected jenkins deploy-slave
- Pipeline Syntax Style: Declarative Pipeline.

Pipeline Stages:

1. "Deploy Client Stack" (selected deploy-slave):
Implemented with the "sh: Shell Script" step:
 - a) stop and remove any existing containers (Publisher, Subscriber)
 - b) start the Arrowhead Publisher container,
 - c) sleep for 10 seconds
 - d) start the Arrowhead Subscriber container

Pipeline Stop Server

Protocol

- Pipeline Parameters:
"Node" = "*list-of-deploy-node*", contains a list of the available Jenkins-Nodes for deployment
- Repositories used:
https://github.com/tuw-cpsg/arrowhead_devops.git / `branch:deployment_demo`, it contains the Jenkinsfile (`Stop_Server_Stack`) for this Pipeline.
- Jenkins nodes: user-selected jenkins deploy-slave
- Pipeline Syntax Style: Declarative Pipeline.

Pipeline Stages:

1. "Stop Server Stack and Clean Up" (selected deploy-slave):
Implemented with the "`sh`: Shell Script" step:
 - a) stop and remove any existing containers (MySQL-Server, Service-Registry, Authorization, Gateway, Event-Handler, Gate-Keeper, Orchestrator)
 - b) remove Arrowhead Service container network
 - c) remove all unused containers, networks, images (both dangling and unreferenced)
 - d) remove all unused local volumes (Unused local volumes are those which are not referenced by any containers).

Pipeline Stop Client

Protocol

- Pipeline Parameters:
"Node" = "*list-of-deploy-node*", contains a list of the available Jenkins-Nodes for deployment
- Repositories used:
https://github.com/tuw-cpsg/arrowhead_devops.git / `branch:deployment_demo`, it contains the Jenkinsfile (`Stop_Server_Stack`) for this Pipeline.
- Jenkins nodes: user-selected jenkins deploy-slave
- Pipeline Syntax Style: Declarative Pipeline.

Pipeline Stages:

1. "Stop Server Stack and Clean Up" (selected deploy-slave):
Implemented with the "`sh`: Shell Script" step:
 - a) stop and remove any existing containers (MySQL-Server, Service-Registry, Authorization, Gateway, Event-Handler, Gate-Keeper, Orchestrator)
 - b) remove all unused containers, networks, images (both dangling and unreferenced)

- c) remove all unused local volumes. (Unused local volumes are those which are not referenced by any containers)

Shared Libraries

What are shared Libraries?

For different projects to reuse a Pipeline or part of, the code would have to be copied to each Jenkinsfile, because it is tied to the specific project. Shared libraries provides classes, global variables and entire Pipelines that can be imported into a Jenkinsfile. The library is stored in version control, e.g. github, and loaded at Pipeline runtime. For Projects to use a library it must be defined either as Global Pipeline Libraries (available to any Project) or in a Folder (available to any direct/indirect child Project). An in-depth discussion of shared libraries is beyond the scope of the Protocol, more details can be found at <https://jenkins.io/doc/book/pipeline/shared-libraries/> .

Lab-Demo Shared Library

Protocol

In an effort to provide reusable Pipeline-code, the Build, and Deployment Pipelines have been re-written and stored in a shared library. In addition a simple client library for the Portus-Rest-API (<http://port.us.org/docs/API.html>) was written. Lastly a simple client to fetch an image-manifest (<https://docs.docker.com/registry/spec/manifest-v2-2/>) from Dockerhub was implemented.

The library is hosted on github <https://github.com/studentblue/shared-libraries>, with public access.

The library has the following structure:

```
(root)
+- src                                     # Groovy
|                                         # source files
|   +- at
|       +- pii
|           +- jenkins_cpsiot_2018
|               +- sandbox
|                   +- addBuildToolHelpers.groovy      # class
|                   +- Constants.groovy                # class
|                   +- DockerHub.groovy               # class
|                   +- PortusApi.groovy               # class
|                   +- buildArrowHeadServerStackHelpers2.groovy # class
|                   +- DeployServerStackHelpers.groovy # class
|                   +- JenkinsApi.groovy              # class
|                   +- Log.groovy                     # class
|                   +- Portus.groovy                  # methods
|                   +- Utils.groovy                   # methods
+- vars                                   # Pipelines
|                                         # and steps
|   +- addBuildToolPipeline.groovy                # Pipeline
|   +- buildArrowHeadServerStackDemoPipeline2.groovy # Pipeline
|   +- DeployServerStackPipeline.groovy            # Pipeline
|   +- pushImage.groovy                           # Pipeline-
|                                                   # Step
```

Internally, scripts in the “vars” directory are instantiated on-demand as singletons. This allows multiple methods to be defined in a single .groovy file for convenience (source: <https://jenkins.io/doc/book/pipeline/shared-libraries/#writing-libraries>).

```
@Library('SandBox') _
addBuildToolPipeline(env, currentBuild)
```

The “vars” directory contains here whole Declarative Pipelines and one step: after defining (globally or in a Folder) the library can be loaded into a Jenkinsfile.

Note that the library name was chosen in the definition.

The Pipeline can be called with parameters by its file-name without the extension
the parameters in this example are the environment-variables and the current-build-number.

```
@Library('SandBox') _
Pipeline
{
    agent any
    stages
    {
        stage('Some Stage')
        {
            Steps
            {
                Script
                {
                    pushImage name1: value, name2: value2
                }
            }
        }
    }
}
```

A Pipeline-Step from the library is called the same way.

Note here the parameter is a Map (http://www.groovy-lang.org/syntax.html#_maps), the step must be inside a script directive (<https://issues.jenkins-ci.org/browse/JENKINS-42360>) for Declarative Pipelines. Shared Pipelines allow their reuse in any project, shared steps could be used to define a more structured Domain-Specific-Language for Pipelines. (<https://jenkins.io/doc/book/pipeline/shared-libraries/#writing-libraries>).

```
// DeployServerStackPipeline.groovy line 4.  
  
Constants = new at.pii.jenkins_cpsiot_2018.sandbox.Constants()
```

The “src” directory contains a collection of methods and classes. As library classes cannot directly call steps such as “sh” or “git”, the method collections were used whenever “calling steps” was needed. The class-file-name is the same as the containing class-name. Classes may then be instantiated in a shared Pipeline.

The approach taken here was to write “lean” shared-pipelines and encapsulate all the necessary implementation-details in classes. The motivation here was to reuse as many code as possible, e.g. the “*Constants.groovy*” and “*Log.groovy*” are used in every shared Pipeline.

Dynamic Project-Parameters

Help

The Project-parameters used so far are static: they are defined once during Project-setup along with all their possible values.

The “Extended Choice Parameter”-plug-in adds a new parameter type called “Extended Choice Parameter”. This parameter can be configured as “JSON Parameter Config Groovy Script File”: the parameters and its values are generated from a Groovy-Script. The file is stored on the Jenkins-Master and can be version-controlled, the plug-in itself has no option to fetch the file from version control.

The script returns an Object of Type “`org.boon.core.value.LazyValueMap`” (<http://javadoc.com/io.fastjson/boon/0.23/org/boon/core/value/LazyValueMap.html>). The Object is passed to JSON Editor (<https://github.com/josdejong/jsoneditor>) which renders the JSON into a WebUI, where the user can set the parameter-values.

JSON Editor returns the parameter-values as JSON which is then accessible to the Pipeline as an environment-variable for the build. The actual build then starts.

To the best knowledge of the author the Groovy-Script runs in the Jenkins Script Console (<https://wiki.jenkins.io/display/JENKINS/Jenkins+Script+Console>) this allows for the parameters and parameter-values to be highly dynamic,

e.g. the Class `RemotingDiagnostics`

(<https://javadoc.jenkins-ci.org/hudson/util/RemotingDiagnostics.html>) allows the execution of shell-scripts on a connected Slave-Node, this was used to query the state of Docker-Containers and generate parameter-values accordingly (see below Pipeline Deploy-Stack).

The used scripts need the following Signatures approved (this requires admin permissions in Jenkins):

```
"method groovy.json.JsonSlurperClassic parseText java.lang.String"  
"method java.util.Date getTime"  
"new groovy.json.JsonSlurperClassic"  
"staticMethod org.boon.Boon fromJson java.lang.String"
```

```
"staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods hasProperty  
java.lang.Object java.lang.String"
```

Approval Configuration Page: <https://jenkins-adress/scriptApproval/>.

No pending script approvals.

You can also remove all previous script approvals: [Clear Approvals](#)

No pending signature approvals.

Signatures already approved:

```
method groovy.json.JsonSlurperClassic parseText java.lang.String  
method java.util.Date getTime  
new groovy.json.JsonSlurperClassic  
staticMethod org.boon.Boon fromJson java.lang.String  
staticMethod org.codehaus.groovy.runtime.DefaultGroovyMethods hasProperty java.lang.Object  
java.lang.String
```

Signatures already approved assuming permission check:

You can also remove all previous signature approvals: [Clear Approvals](#)

No pending classpath entry approvals.

Classpath entries already approved:

No approved classpath entries.

Figure 8 Approved Method Signatures

Pipeline AddBuildTool

Protocol

The DockerHub build-images have to be pushed in the private registry, this was done “manually” (see *Pull and Push of Docker-build-Images*). The AddBuildTool Pipeline is an automation of these steps, the code is stored in Lab-Demo Shared Library.

- Required Folder properties:
 - “REPO_URL” is the url of the private registry
 - “PORTUS_USER” portus user name
 - “PORTUS_CREDS_STD” credentials id for user & password of portus account
 - “PORTUS_USER_TOKEN_API” credentials id for user & token of portus account
 - “PORTUS_USER_ID” portus user-id
- Pipeline Parameter:
 - Name: “AddBuildTool”,
 - Type: “Extended Choice Parameter”

- generated from script:
https://github.com/studentblue/jenkins_scripts/blob/master/scripts_jenkins/SchemaAddBuildToolAlpha1.2.groovy,
- Settings: “JSON Parameter Config Groovy Script File” / “Advanced” → “Variable bindings”: “base=*base-folder-name*”

```
{
  "DockerHub":
  {
    "repo": "openjdk",          //DockerHub-registry repo-name
    "tag": "8u162-jre-slim"    //tag within the repo
  },
  "Namespace":
  {
    "name":                    //existing namespace in the registry
    {
      "id": 10,
      "name": "cpsiot-deploy-test-deploy1-testa",
      "description": "Deploy Namespace for User testa",
      "team": "testa-deploy1",    //team of the namespace
      "visibility": "private"
    },
    "teamFromNamespace": true    //use team associated with namespace },
    "Repo":                    //repository in the private registry
    {
      //to push the image to
      "name": true,             //pipeline generates the name
      "tag": true,              //pipeline generates the tag
      "type": "base-with-java"  //used by the pipeline to generate the
                                //tag
    }
  }
}
```

- The parameters generated are the DockerHub image-name and tag to download and the private registry image-name and tag the image will be pushed under.
 Sample JSON generated:
- Here the user selected an existing name-space to push the image to.
 The existing name spaces to choose from are fetched on each call (dynamic) from the private registry. Alternatively the user may choose to define a new name-space with an existing team or a new team, the name-space-name can be set to be automatically generated. The repository-name can be set to be generated or input by the user, same

for the tag. The generation of values (if selected) is delegated to the Pipeline. The DockerHub repository name and tag were queried outside Jenkins from (<https://hub.docker.com/>).

The screenshot shows the Jenkins configuration interface for an 'Extended Choice Parameter'. The 'General' tab is selected. At the top, a checkbox 'This project is parameterized' is checked. The parameter is named 'AddBuildTool' with the description 'Parameters for the image to push'. Under 'Basic Parameter Types', the 'JSON Parameter Type' is selected. Below this, the 'JSON Parameter Config Groovy Script File' option is selected, with the file path '/home/ubuntu/scripts_github/scripts_jenkins/SchemaAddBuildTool/Alpha1.2.' entered. The 'Variable bindings' section contains 'base=testa'. The 'Groovy Classpath' section is empty. At the bottom, the 'JSON Parameter Config Javascript File' option is also visible, with the file path '/home/ubuntu/scripts_github/scripts_jenkins/SchemaAddBuildTool/Alpha1.1.' entered. A 'Save Parameter to File' checkbox is unchecked. An 'Add Parameter' button is at the bottom left.

General Build Triggers Advanced Project Options Pipeline

☒ This project is parameterized

Extended Choice Parameter

Name: AddBuildTool

Description: Parameters for the image to push

☐ Basic Parameter Types

☐ Multi-level Parameter Types

☒ JSON Parameter Type

☐ JSON Parameter Config Groovy Script

☒ JSON Parameter Config Groovy Script File

/home/ubuntu/scripts_github/scripts_jenkins/SchemaAddBuildTool/Alpha1.2.

Variable bindings: base=testa

Groovy Classpath

☐ JSON Parameter Config Javascript

☒ JSON Parameter Config Javascript File

/home/ubuntu/scripts_github/scripts_jenkins/SchemaAddBuildTool/Alpha1.1.

Save Parameter to File ☐

Add Parameter

Figure 9 Extended Choice Parameter from Script-File

- **Repositories used:** <https://github.com/studentblue/shared-libraries>
- **Jenkinsfile:**

```
import at.pii.jenkins_cpriot_2018.sandbox.*
addBuildToolPipeline( env, currentBuild )
```

- **Jenkins nodes:** any
- **Pipeline Syntax Style:** Declarative Pipeline.

Pipeline Stages:

1. 'Init Env & Test Parameters':
Initialize Pipeline-Helpers and check parameters (e.g. the repository exists on DockerHub) and folder properties.
2. 'Select Image'
Delays Pipeline-execution by prompting the user to select the image-version (e.g. Linux-ARM or Linux-amd64), in case the image has only a single variant the prompt is skipped.
3. "Push Selected Image"
Delays Pipeline-execution by prompting the user to confirm the image-selection, after confirmation the image is tagged and pushed to the private registry

Pipeline Build-ArrowHead-ServerStack-Demo

Protocol

Builds Arrowhead-Stack from source and packages selected Services/Mysql-Server.

- **Required Folder properties:**
 - "REPO_URL" is the url of the private registry
 - "PORTUS_USER" portus user name
 - "PORTUS_CREDS_STD" credentials id for user & password of portus account
 - "PORTUS_USER_TOKEN_API" credentials id for user & token of portus account
 - "PORTUS_STD_CLOUD_TEAM"
 - "PORTUS_USER_ID" portus user-id
- **Pipeline Parameter:**
 - **Name:** "ArrowHeadServerStack",
 - **Type:** "Extended Choice Parameter"
 - **Generated from script**
https://github.com/studentblue/jenkins_scripts/blob/master/scripts_jenkins/SchemaBuildArrowHeadServerStackDemo1-1.groovy,

- Settings: “JSON Parameter Config Groovy Script File” / “Advanced” → “Variable bindings”: “base=base-folder-name”

The parameter-script generates four categories of parameters:

- *Compile*: maven-image to use (DockerHub or private registry), maven-cache to use, maven command to use (e.g. mvn)
- *NameSpace*: Existing or New, when a new name-space is chosen a team must be chosen, the teams are fetched from the private registry
- *Images*: a list of Arrowhead Service docker-images to create; an item may be added from a pre-defined set of categories: [Service-registry, Authorization, Gateway, Event-Handler, Gate-Keeper, Orchestrator];

The item can be customized further by:

- *Build-image* to use, available options are fetched the private registry,
- *Repo* (private registry repository-name),
- *Path* to byte-code to package relative to the source-code checkout directory (e.g. Eventhandler/target/),
- *Entry-point*, command to execute when the container is created (e.g. "java -cp lib/*:* -jar arrowhead-authorization-4.1.2.jar -d -daemon"),
- *Workdir*, the directory-path the byte-code will be copied to within the docker-image (e.g. /arrowhead/eventhandler) and the entry-point-command will run in.

The list must have at least one item.

- *Arrowhead*: the url of the github repo to checkout, only repositories with public access are supported at the moment
- *Repositories used*: <https://github.com/studentblue/shared-libraries>
- *Jenkinsfile*:

```
@Library('SandBox') _
import at.pii.jenkins_cpsiot_2018.sandbox.*

buildArrowHeadServerStackDemoPipeline2( env, currentBuild,
env.ArrowHeadServerStack )
```

Pipeline Stages:

1. "init" (any Jenkins Node):
init Pipeline-Helpers and check parameters and folder properties.
2. "Create Maven Cache" (Master):
Sets up the docker-volume used as shared maven cache.

3. "Build" (Master):
Generate the Java-byte-code from source.
4. "Dockerize Selected Images" (any Jenkins Node):
Generate "app.properties", sql init-script, package byte-code and configuration-files/script, push image(s).

Pipeline Deploy-Stack

Protocol

Builds Arrowhead-Stack from source and packages selected Services/Mysql-Server.

- Required Folder properties:
 - "REPO_URL" is the url of the private registry
 - "PORTUS_USER" portus user name
 - "PORTUS_CREDS_STD" credentials id for user & password of portus account
 - "PORTUS_USER_TOKEN_API" credentials id for user & token of portus account
 - "PORTUS_STD_CLOUD_TEAM"
 - "PORTUS_USER_ID" portus user-id
- Pipeline Parameter 1:
 - Name: "DeployParameters",
 - Type: "Extended Choice Parameter"
 - generated from script:


```
https://github.com/studentblue/jenkins_scripts/blob/master/scripts_jenkins/DeployServerStackAlpha1-1.groovy,
```
 - Settings:
 - "JSON Parameter Config Groovy Script File" / "Advanced" → "Variable bindings":
"base=base-folder-name"
 - The parameter-script generates five categories of parameters:
 - ♦ Docker: cloud is the name of the docker-network the containers will be part of, new and from node (existing network on the node, read-only, chosen in the Node-Category) are possible; delay sets the time-out between container starts
 - ♦ Node: Node is a list of slaves the user is owner of (primary of secondary) excluding the Master and the build-slave(s) (identified by name starting with "jenkins-slave") that are connected, the user can choose from;name, kernel, os and Docker Client and Server version on the selected node is displayed read-only; networks shows a list of docker-networks on the selected node (bridge, host and none are excluded), the selected value is linked to existing-cloud-value in Docker, changing the value here will trigger a change to the other value, the network can be flagged to be removed; if any containers exist that are connected to the selected network they are

listed here, each container can be flagged to be removed; Note removing a network removes all connected containers.

- ♦ Images: Images is the list of images that are to be deployed on the selected node, an Image is either a DB (database), SERVICE_REGISTRY, AUTHORIZATION, GATEWAY, EVENTHANDLER, GATEKEEPER or ORCHESTRATOR;
- Each image can be further customized:
 - ♦ *deployImage*, is the image to pull from the private-registry for the selected type (e.g. DB), the list of available images is fetched dynamically from the registry, if no image exists for a specific type, the whole type is excluded from selection.
 - ♦ For the DB-type an initDBScript is generated or can be input by the user (macros are provided to substitute other parameters when the Pipeline runs, e.g. the ArrowHead DB name which is set in another category).
 - ♦ For the other Image-Types the app-properties are provided according to their type as input masks; common to all Arrowhead Images is the “workdir”, this is where the generated properties-file will be copied to on container creation (e.g. “/arrowhead/orchestrator”), note that this must match the location of the byte-code already contained in the Image.
 - ♦ ArrowHead: configures the network properties of the services, ports, addresses, etc.; conf defines the properties file-convention (Arrowhead 3 “app.properties” and “log4j.properties” or 4 “app.conf”).
 - ♦ Logger: defines the Logger properties.
- Pipeline Parameter 2:
Database username and password for the Arrowhead Services application properties-files
 - Name: “ArrowHead”,
 - Type: “Credentials Parameter”
- Pipeline Parameter 3:
Root password for the DB-Server-Image
 - Name: “DBRootPassword”,
 - Type: “Credentials Parameter”
- Repositories used: <https://github.com/studentblue/shared-libraries>
- Jenkinsfile: “

```
@Library('SandBox') _  
  
import at.pii.jenkins_cpsiot_2018.sandbox.*  
  
DeployServerStackPipeline( env, currentBuild, env.DeployParameters,  
env.ArrowHead, env.DBRootPassword )
```

Pipeline Stages:

1. "init":
init Pipeline-Helpers and check parameters and folder properties
2. "Cloud Management":
the docker-networks are removed (if flagged) or created, flagged containers are removed
3. "Deploy Selected Images":
depending on the selected parameters the property-files and DB script are generated, the containers are started in sequence with respect to their defined position in the Image-List, the delay is used as time-out between starts, the directories containing the properties/script files are mounted at container start.

Conclusion

Pipelines with static parameters are simple to use and extend/develop, dynamic parameters introduce complexity for the user and Pipeline developer. The approach here was to start with many simple Pipelines with static parameters tailored to very specific Projects and building upon them to create shared Pipelines (with dynamic parameters) which cover a wider range of Projects. Another approach could be to develop a domain-specific Pipeline language as a shared library which provides easy to use building steps.