JePL
(*J*enkins *P*ipeline *L*ibrary)

Speaker: Samuel Bernardo <samuel@lip.pt>
On behalf of WP3

JePL: what, why and how

What	Core component of the SQAaaS platform	● Implementation of baseline quality criteria ● Creation and execution of QA pipelines, CI and CD ● Used by SQAaaS components
Why	Facilitates adoption of DevOps practices	● Development practices improvement ● Enable automation of the QA process ● Flexible tooling adoption for python (tox), java (maven) or any other tool
How	Using human-readable YAML format instead of Jenkins Groovy-based language	● Using docker compose to load the build tools and environment setup ● Easy creation and execution of complex pipelines for QA ● Library leveraging the Jenkins PaC

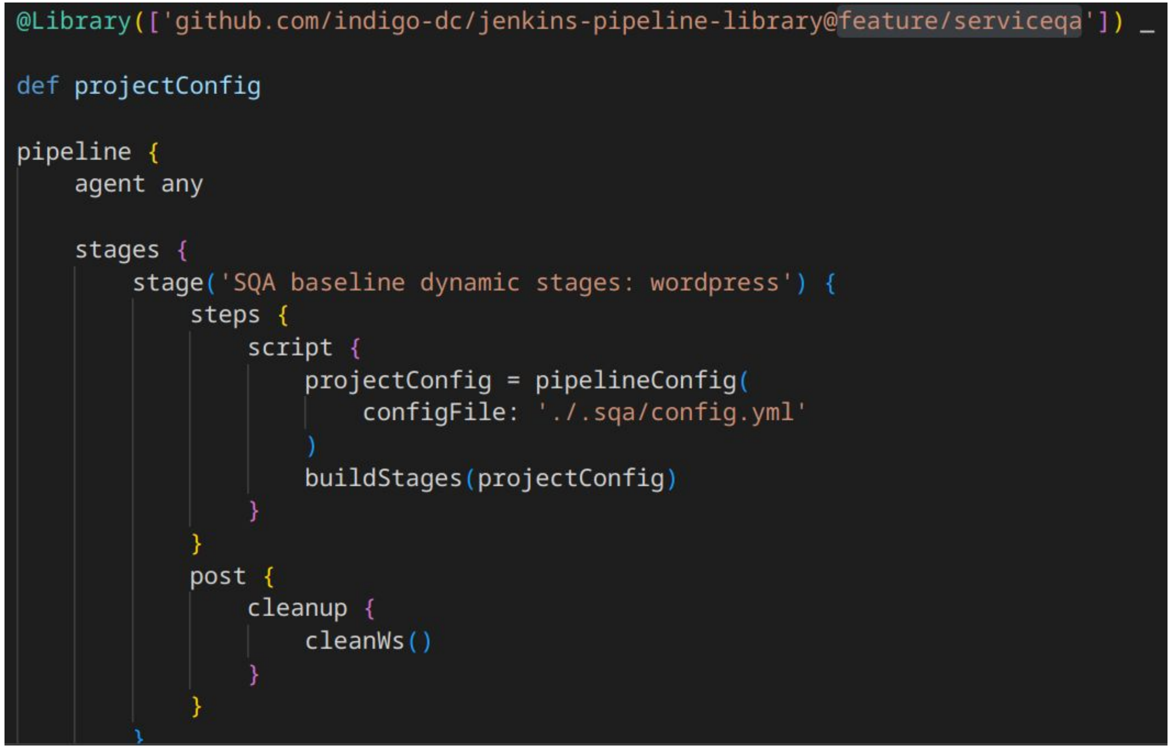
JePL: https://github.com/indigo-dc/jenkins-pipeline-library Latest: release 2.4.0

JePL: where PaC become JCasC

- **Jenkins framework** provides an implementation of **Pipeline as Code (PaC)**
    - Define pipelined job processes as code, stored and versioned in source repository
    - Distributed build environment that provides jobs automation over git platform events
    - Designed for distributed build environments
    - Allow to use different environments for each project
    - Workload balancing among multiple agents running jobs in parallel

JePL: where PaC become JCasC

- **Jenkins framework** provides an implementation of **Pipeline as Code (PaC)**



```
@Library(['github.com/indigo-dc/jenkins-pipeline-library@feature/serviceqa']) _

def projectConfig

pipeline {
    agent any

    stages {
        stage('SQA baseline dynamic stages: wordpress') {
            steps {
                script {
                    projectConfig = pipelineConfig(
                        configFile: './.sqa/config.yml'
                    )
                    buildStages(projectConfig)
                }
            }
            post {
                cleanup {
                    cleanWs()
                }
            }
        }
```

JePL: where PaC become JCasC

- **JePL shared library** enhances the pipeline with **Jenkins Configuration as Code (JCasC)**
  - Define pipeline using human-readable configuration files (config.yml)
  - Easy means to compose Jenkins code pipelines (`Jenkinsfile`)

JePL: where PaC become JCasC

- **JePL shared library** enhances the pipeline with **Jenkins Configuration as Code (JCasC)**
  - Support for the criteria defined in the Software & Service QA baselines
    - Defined through a `config.yml` file (added to code repo)
  - Built-in support for Python's tox build tool and Java's maven build tool
    - Besides that, any tool is already supported with commands property in config.yml

JePL: where PaC become JCasC

- **JePL shared library** enhances the pipeline with **Jenkins Configuration as Code (JCasC)**
  - Support for IM (Infrastructure Manager) and EC3 (Elastic Cloud Computing Cluster)
    - Tools launched with docker-compose and all operations are executed from provided container maintained by GRyCAP from UPV
  - Support kubectl (normal k8s receipts and kustomizations) and helm (helm charts)
  - These tools are used at SvcQC.Dep for the infrastructure and services deployment

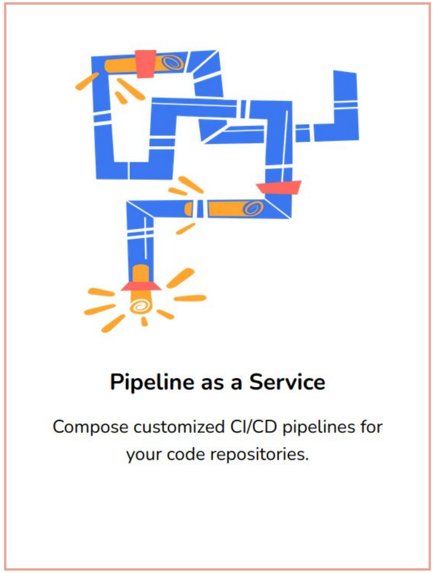
JePL adoption advantages

- JePL provides **easy adoption of the QA criteria** compiled in the **SW and SVC baselines**
  - Hence, fostering SQA practices on research software, e.g. EOSC services
  - EOSC-Synergy Thematic Services are gradually adopting JePL
- JePL requires 3 files, but only one is the **fundamental basis**→**`config.yml`**
  - `Jenkinsfile` & `docker-compose.yml` are dependencies for *automation* & *resource provisioning*, respectively

JePL adoption advantages

- **JePL focus on supporting**:
  - Additional QA criteria from the SW and SVC baselines
  - Additional composers to integrate with different platforms, like K8s
  - Additional tools delivered as Docker images
- The **SQAaaS solution leverage JePL to graphically compose on-demand CI/CD pipelines** https://sqaaas.eosc-synergy.eu/#/auth/select-option

SQAaaS module selection

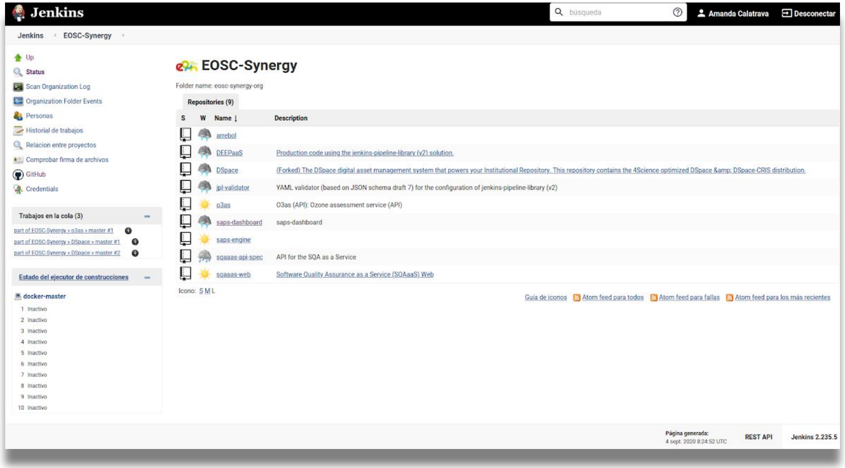


Pipeline as a Service

Compose customized CI/CD pipelines for your code repositories.

Jenkins instance to check the pipeline logs

- Checks automatically all the projects in EOSC Synergy Github organization: https://github.com/EOSC-synergy
- You can also use your own instance of Jenkins in case of repositories with restricted access.
    - You can install a local deployment of the Jenkins pipeline to run the tests.



Trabajos en la cola (3)

part of EOSC-Synergy » o3as » master #1
part of EOSC-Synergy » DSpace » master #1
part of EOSC-Synergy » DSpace » master #2

Estado del ejecutor de construcciones

docker-master
1 Inactivo
2 Inactivo
3 Inactivo
4 Inactivo
5 Inactivo
6 Inactivo
7 Inactivo
8 Inactivo
9 Inactivo
10 Inactivo

EOSC-Synergy

Folder name: eosc-synergy-org

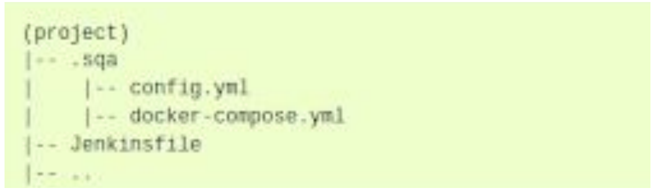
Repositories (9)

S	W	Name ↓	Description
		arrebol	
		DEEPaaS	Production code using the jenkins-pipeline-library (v2) solution.
		DSpace	(Forked) The DSpace digital asset management system that powers your Institutional Repository. This repository contains the 4Science optimized DSpace &amp; DSpace-CRIS distribution.
		jpl-validator	YAML validator (based on JSON schema draft 7) for the configuration of jenkins-pipeline-library (v2)
		o3as	O3as (API): Ozone assessment service (API)
		saps-dashboard	saps-dashboard
		saps-engine	
		sqaaas-api-spec	API for the SQA as a Service
		sqaaas-web	Software Quality Assurance as a Service (SQAaaS) Web

EOSC Synergy Jenkins instance: https://jenkins.eosc-synergy.eu/job/eosc-synergy-org/

JePL: Software Quality Assurance (SQA)

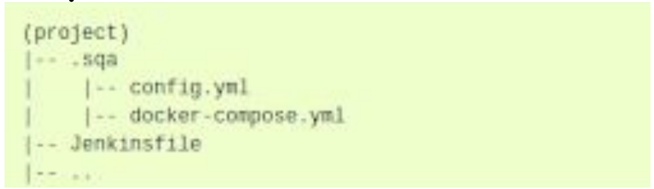
- **Includes support for style checking (QC.Sty), unit tests (QC.Uni), code metadata (QC.Met), licensing (QC.Lic), security (QC.Sec) and documentation (QC.Doc).**
- Configuration files
  - The configuration file: `config.yml`
  - The services: `docker-compose.yml`
  - The pipeline: `Jenkinsfile`



```
(project)
|-- .sqa
|     |-- config.yml
|     |-- docker-compose.yml
|-- Jenkinsfile
|-- ..
```

JePL: Service Quality Assurance (SvcQA)

- **Includes support for automated deployment (SvcQC.Dep), API tests (SvcQC.API), integration tests (SvcQC.Int), functional tests (SvcQC.Fun), security tests (SvcQC.Sec) and documentation (QC.Doc).**
- `Configuration files are the same`



```
(project)
|-- .sqa
|    |-- config.yml
|    |-- docker-compose.yml
|-- Jenkinsfile
|-- ...
```

- IM, EC3, K8s and Helm test pipelines continuous testing of releases https://jenkins.eosc-synergy.eu/job/eosc-synergy-org/ (JePL-*-test repositories)
- In case of doubts, please open an issue in: https://github.com/EOSC-synergy/issue-tracker/issues/new/choose

How difficult is to use JePL to test software?

Goal: **use JePL to check the compliance of 2 types of criteria from the SW QA baseline** [QC.Sty, QC.Sec]

- Test with **a real application** delivered through the EOSC portal
  - ⇝ DEEP as a Service: https://github.com/indigo-dc/DEEPaaS
- Mimic the process of **JePL adoption by a first-timer**
  - Following the step-by-step guide at:
    - ⇝ https://indigo-dc.github.io/jenkins-pipeline-library/
- Results appear in EOSC-Synergy's Jenkins instance
  - ⇝ https://jenkins.eosc-synergy.eu/job/eosc-synergy-org/job/DEEPaaS/

How difficult is to use JePL to test software?

1. Let's start cloning the code repository (from fork’s master branch):

```
git clone -b master https://github.com/EOSC-synergy/DEEPaaS
```

2. Create an “jepl_demo” branch for the JePL-required files:

```
cd DEEPaaS && git checkout -b jepl_demo
```

How difficult is to use JePL to test software?

3. Create `config.yml` and `docker-compose.yml` *under the `.sqa` folder* (pre-composed files, “eosc-synergy” branch):

```
mkdir .sqa && wget -P .sqa
https://raw.githubusercontent.com/EOSC-synergy/DEEPaaS/eosc-synergy/.sqa/config.yml
https://raw.githubusercontent.com/EOSC-synergy/DEEPaaS/eosc-synergy/.sqa/docker-compose.yml
```

4. Create the `Jenkinsfile` *in the repo root path with the code provided in the documentation*:

```
wget https://raw.githubusercontent.com/EOSC-synergy/DEEPaaS/eosc-synergy/Jenkinsfile -O
Jenkinsfile
```

How difficult is to use JePL to test software?

5. We only need to commit and push the previous changes:

```
git add .sqa Jenkinsfile
```

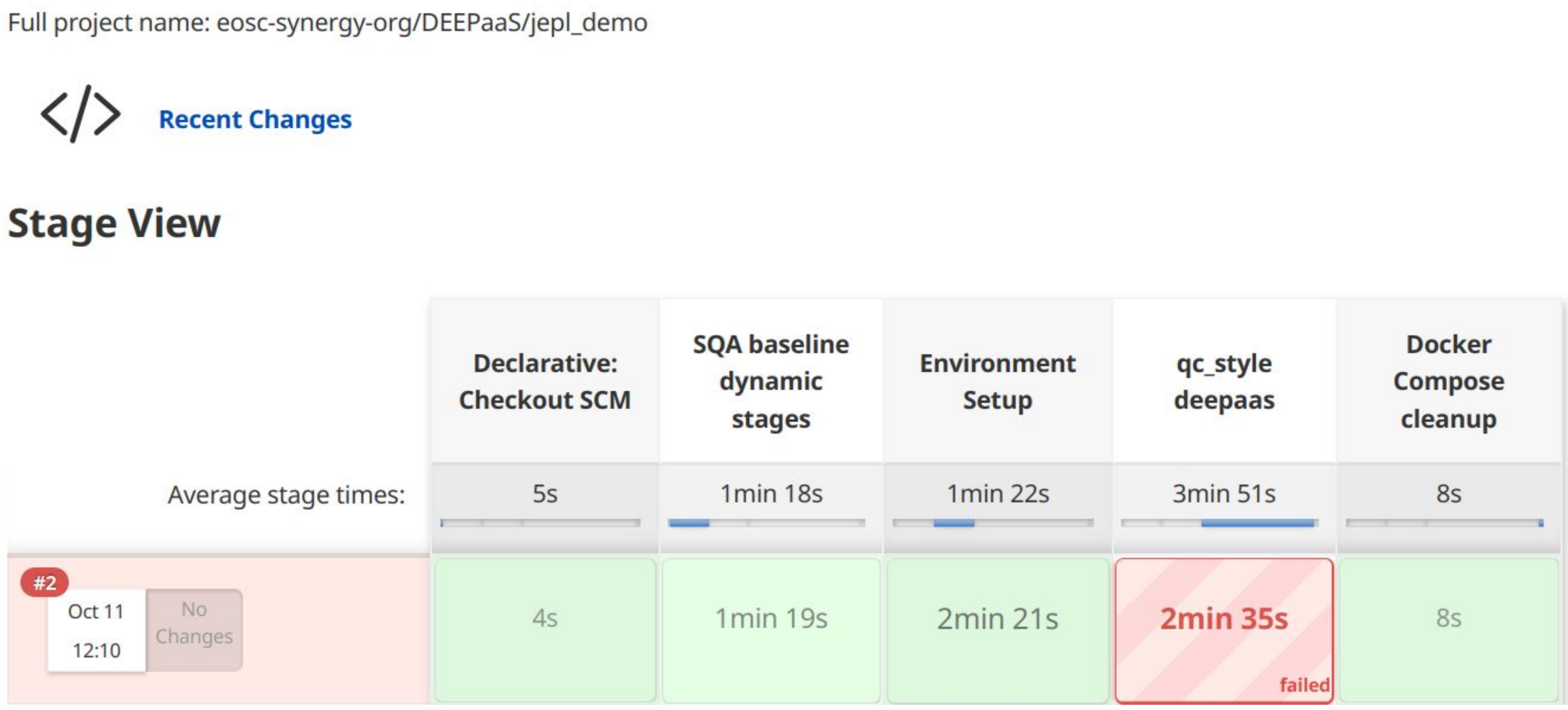
```
git commit -m "Initial skeleton of JePL files"
```

```
git push -u origin jepl_demo
```

Now we can see the magic happening and wait for the results

https://jenkins.eosc-synergy.eu/job/eosc-synergy-org/job/DEEPaaS/job/jepl_demo/

How difficult is to use JePL to test software?



Full project name: eosc-synergy-org/DEEPaaS/jepl_demo

Recent Changes

Stage View

	Declarative: Checkout SCM	SQA baseline dynamic stages	Environment Setup	qc_style deepaas	Docker Compose cleanup
Average stage times:	5s	1min 18s	1min 22s	3min 51s	8s
#2 Oct 11 12:10 No Changes	4s	1min 19s	2min 21s	2min 35s failed	8s

How difficult is to use JePL to test services?

Goal: **use JePL to check the compliance with deployment criterion from the Service QA baseline** [QC.Dep]

- Test with **a real application** from samples available for K8s
  - ⇝ Wordpress: https://github.com/EOSC-synergy/JePL-k8s-test.git
- Mimic the process of **JePL adoption by a first-timer**
  - Following the step-by-step guide at:
    - ⇝ https://indigo-dc.github.io/jenkins-pipeline-library/
- Results appear in EOSC-Synergy’s Jenkins instance
  - ⇝ https://jenkins.eosc-synergy.eu/job/eosc-synergy-org/job/JePL-k8s-test/

How difficult is to use JePL to test services?

1. Let's start cloning the code repository (from fork’s master branch):

```
git clone -b master https://github.com/EOSC-synergy/JePL-k8s-test
```

2. Create an “jepl_demo” branch for the JePL-required files:

```
cd JePL-k8s-test && git checkout -b jepl_demo
```

How difficult is to use JePL to test services?

3. Create `config.yml` and `docker-compose.yml` *under the `.sqa` folder* (pre-composed files, “eosc-synergy” branch):

```
mkdir .sqa && wget -P .sqa
https://raw.githubusercontent.com/EOSC-synergy/EOSC-synergy/JePL-k8s-test/feature/serviceqa/.sq
a/config.yml
https://raw.githubusercontent.com/EOSC-synergy/EOSC-synergy/JePL-k8s-test/feature/serviceqa/.sq
a/docker-compose.yml
```

4. Create the `Jenkinsfile` *in the repo root path with the code provided in the documentation*:

```
wget https://raw.githubusercontent.com/EOSC-synergy/JePL-k8s-test/feature/serviceqa/Jenkinsfile
-O Jenkinsfile
```

How difficult is to use JePL to test services?

5. We only need to commit and push the previous changes:

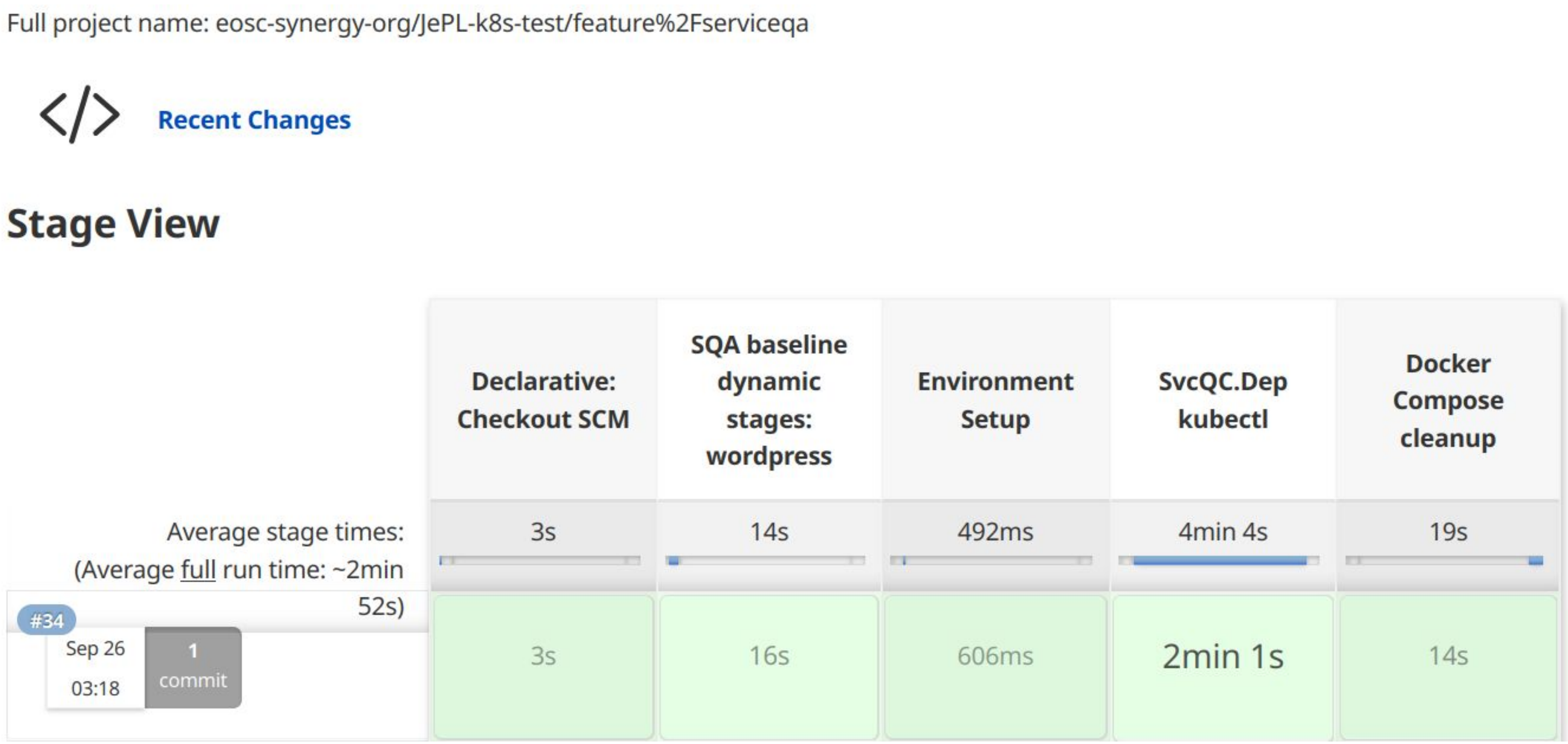
```
git add .sqa Jenkinsfile

git commit -m "Initial skeleton of JePL files"

git push -u origin jepl_demo
```

Now we can see the magic happening and wait for the results
https://jenkins.eosc-synergy.eu/job/eosc-synergy-org/job/JePL-k8s-test//job/jepl_demo/

How difficult is to use JePL to test services?



Full project name: eosc-synergy-org/JePL-k8s-test/feature%2Fserviceqa

Recent Changes

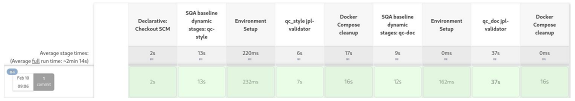
Stage View

	Declarative: Checkout SCM	SQA baseline dynamic stages: wordpress	Environment Setup	SvcQC.Dep kubectl	Docker Compose cleanup
Average stage times: (Average full run time: ~2min 52s)	3s	14s	492ms	4min 4s	19s
#34 Sep 26 03:18 1 commit	3s	16s	606ms	2min 1s	14s

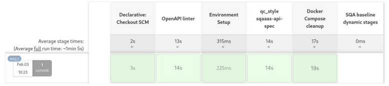
SQAaaS: TSs already using JePL

- Internal JePL usage from SQAaaS services themselves
  - JePL schema validator (validates JSON schema & builds validator's Docker image)
  - SQAaaS Web (builds & publishes production Web)
  - SQAaaS API (validates OpenAPI spec, builds & publishes API docs)

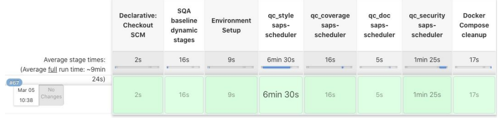
- WP4 thematic services with ready SQA pipelines
  - **WORSICA** https://jenkins.eosc-synergy.eu/job/WORSICA/
  - **O3AS** https://jenkins.eosc-synergy.eu/job/eosc-synergy-org/ (o3* projects)
  - **SAPS** https://jenkins.eosc-synergy.eu/job/eosc-synergy-org/ (saps-* projects)
  - **LAGO** https://jenkins.eosc-synergy.eu/job/eosc-synergy-org/job/onedataSim/
  - **OpenEBench** https://jenkins.eosc-synergy.eu/job/eosc-synergy-org/job/bench_event_api/



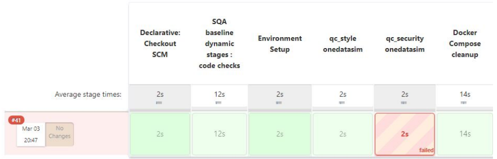
	Declarative: Checkout SCM	SQA baseline dynamic stages: qc-style	Environment Setup	qc_style jpl-validator	Docker Compose cleanup	SQA baseline dynamic stages: qc-doc	Environment Setup	qc_doc jpl-validator	Docker Compose cleanup
Average stage times: (Average full run time: ~2min 14s)	2s	13s	220ms	6s	17s	9s	0ms	37s	0ms
#4 Feb 10 09:06 1 commit	2s	13s	232ms	7s	16s	12s	162ms	37s	16s



	Declarative: Checkout SCM	OpenAPI linter	Environment Setup	qc_style sqaaas-api-spec	Docker Compose cleanup	SQA baseline dynamic stages
Average stage times: (Average full run time: ~1min 5s)	2s	13s	315ms	14s	17s	0ms
#163 Feb 03 10:23 1 commit	3s	14s	225ms	14s	19s	



	Declarative: Checkout SCM	SQA baseline dynamic stages	Environment Setup	qc_style saps-scheduler	qc_coverage saps-scheduler	qc_doc saps-scheduler	qc_security saps-scheduler	Docker Compose cleanup
Average stage times: (Average full run time: ~9min 24s)	2s	16s	9s	6min 30s	16s	5s	1min 25s	17s
#67 Mar 05 10:38 No Changes	2s	16s	9s	6min 30s	16s	5s	1min 25s	17s



	Declarative: Checkout SCM	SQA baseline dynamic stages : code checks	Environment Setup	qc_style onedatasim	qc_security onedatasim	Docker Compose cleanup
Average stage times:	2s	12s	2s	2s	2s	14s
#41 Mar 03 20:47 No Changes	2s	12s	2s	2s	2s failed	14s

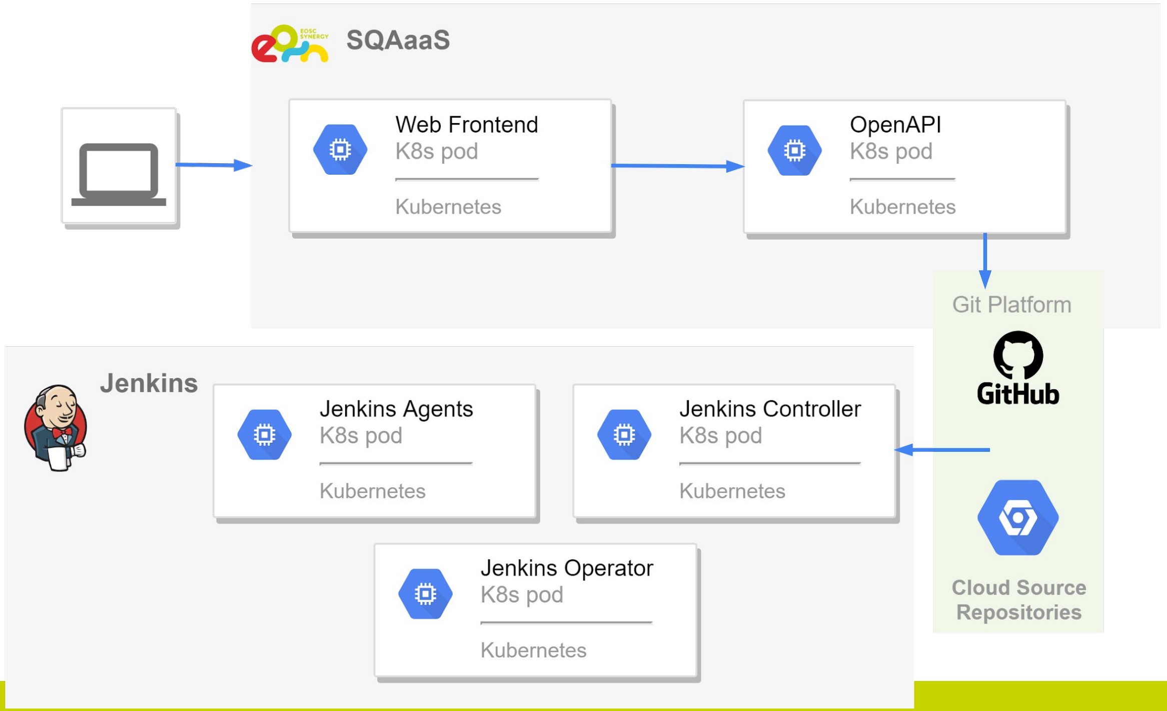
SQAaaS: TSs already using JePL

- WP4 thematic services ongoing work with Service QA pipelines
  - **WORSICA**
  - **O3AS**
  - **SAPS**
  - **MSWSS**
  - **SCIPION**

- More than 20 thematic service repositories are already using JePL

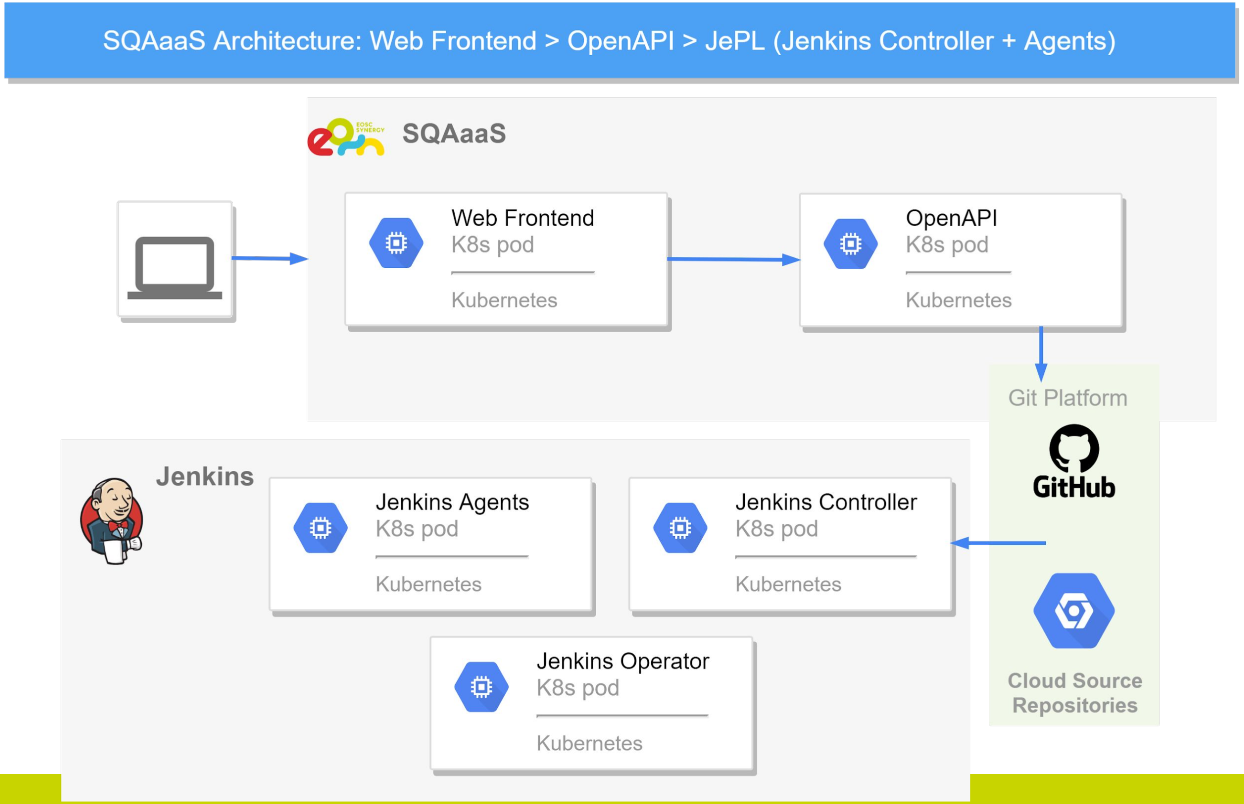
Next steps

SQAaaS Architecture: Web Frontend > OpenAPI > JePL (Jenkins Controller + Agents)



- Adopt Kubernetes as the resource manager
- Jenkins Operator deployment already concluded
- Improve scalability of the platform
- Bypass GitHub platform limitations adding an on-premises Git platform

Next steps



- Support multi-site deployment using Fedcloud services (EGI Load Balancer and EGI Dynamic DNS service)
- Finish JePL v3 that will enhance the integration with K8s and Jenkins Operator for the deployments

Documentation

The user's guide is available on the following url

https://indigo-dc.github.io/jenkins-pipeline-library/

SQA baseline

The latest version of the Software QA criteria can be found in https://indigo-dc.github.io/sqa-baseline

The Service QA criteria is currently in development and is available at https://github.com/EOSC-synergy/service-qa-baseline

Thanks for your attention

[Docs] https://indigo-dc.github.io/jenkins-pipeline-library

Submit an issue for any JePL-related question through GitHub:

https://github.com/indigo-dc/jenkins-pipeline-library/issues

Keep posted for EOSC-Synergy SQAaaS developments:

https://github.com/EOSC-synergy/