OpenRiskNet

RISK ASSESSMENT E-INFRASTRUCTURE

Deliverable Report D2.1

Development infrastructure online

This project is funded by
the European Union

OpenRiskNet: Open e-Infrastructure to Support Data Sharing, Knowledge Integration and in silico Analysis and Modelling in Risk Assessment

Project Number 731075

www.openrisknet.org

Project identification

Grant Agreement	731075
Project Name	OpenRiskNet: Open e-Infrastructure to Support Data Sharing, Knowledge Integration and in silico Analysis and Modelling in Risk Assessment
Project Acronym	OpenRiskNet
Project Coordinator	Douglas Connect GmbH
Star date	1 December 2016
End date	30 November 2019
Duration	36 Months
Project Partners	P1 Douglas Connect GmbH Switzerland (DC) P2 Johannes Gutenberg-Universitat Mainz, Germany (JGU) P3 Fundacio Centre De Regulacio Genomica, Spain (CRG) P4 Universiteit Maastricht, Netherlands (UM) P5 The University Of Birmingham, United Kingdom (UoB) P6 National Technical University Of Athens, Greece (NTUA) P7 Fraunhofer Gesellschaft Zur Foerderung Der Angewandten Forschung E.V., Germany (Fraunhofer) P8 Uppsala Universitet, Sweden (UU) P9 Medizinische Universitat Innsbruck, Austria (MUI) P10 Informatics Matters Limited, United Kingdom (IM) P11 Institut National De L'environnement Et Des Risques, France (INERIS)

Deliverable Report identification

Document ID and title	Deliverable 2.1 Development infrastructure online
Deliverable Type	Demonstrator
Dissemination Level	Public (PU)
Work Package	WP2
Task(s)	Task 2.1 Creation of development environment
Deliverable lead partner	IM
Author(s)	Tim Dudgeon (IM) Ola Spjuth (UU) Frederic Bois (INERIS) Daniel Bachler (DC)
Status	Final
Version	V1.0
Document history	2017-05-12 Draft ready for review 2017-05-31 Final version

Table of Contents

SUMMARY 4

INTRODUCTION 4

PLANNING 5

Options Investigated 5

OpenRiskNet Technical Environment 6

Management of the Technical Environment 7

Security 7

Backup and recovery 7

Guidelines for development 8

Core artifacts should be Docker containers 8

Container orchestration 8

Impact of Containerisation 9

Configuration 9

Build and test processes 9

Partnering with other Projects 9

CURRENT DEVELOPMENT ENVIRONMENT IMPLEMENTATION 11

GitHub 11

Components of GitHub 11

GitHub Access control 12

Modes of operation 12

Core project code 12

Code from partner with existing repo 13

Partner wanting to create new project for OpenRiskNet 13

Integrate external tools/services 13

Jenkins 14

EXAMPLE PROJECTS 15

Example 1: simple Java servlet example 15

Example 2: simple Python project 16

RISK ASSESSMENT 18

CONCLUSIONS 18

SUMMARY

This Deliverable describes the computational infrastructure, frameworks and systems for the development and testing of the Virtual Research Environments, APIs and data and software integration of the OpenRiskNet project. Development tools selected for source code control, issue tracking, continuous integration and deployment, and containerization and container orchestration are discussed and guideline for service development are outlined. Finally the current development environment is described and its operation is demonstrated with a simple example.

INTRODUCTION

OpenRiskNet will create and establish an openly accessible e--infrastructure to deal with the increasing demands concerning the prediction of safety of existing and new molecular entities for health and environmental sustainability. OpenRiskNet will address the challenges arising from the current fragmentation and insufficient harmonization of data, software, and user guidance by developing means to provision Virtual Research Environments (VREs), where components are containerized and deployed as microservices that can communicate via Application Programming Interfaces (APIs). The development of APIs is covered in Task 2.2 and reported in D2.2 and D2.4. The work towards containerization of tools and data is covered in WP4 and reported in D4.1-3. OpenRiskNet will also provide its own reference VRE that will act as the public showcase for the project.

The current service and tool development in the field of chemical safety assessment is fragmented, with individual tools following different development strategies, timeframes and with varying degrees of stability, testing, and support for standards. This puts a heavy burden on end users that wish to use these tools, as they explicitly need to deal with locating the data and tools, downloading, compilation/installation, and in many cases conversion between file formats and responding to changes in APIs when updating to new versions. OpenRiskNet will provide a system that collects the data and tools components in a single system, provides the means for building and testing each component individually and integrated, and packaging it into containers for interoperable use in the VRE. In order to sustain such a continuous integration and continuous deployment (CI/CD), a well-designed development environment is needed, and that environment needs to be able to facilitate easy deployment of the components it builds to individual VREs.

On the lowest level of the OpenRiskNet e-infrastructure are Virtual infrastructures, comprising compute nodes, networks, storage, middleware, software, workflows etc - all glued together into a VRE where end users have the necessary data and computational resources and tools available to carry out the intended tasks. Users should be able to instantiate VREs on different cloud providers, including public cloud providers such as Google Cloud Platform, Amazon Web Services, Microsoft Azure but also on private cloud providers including OpenStack, and on local computers/servers.

PLANNING

Options Investigated

The project partners have a wide range of expertise in developing and deploying a wide range of software. We wanted to build on this experience, but also ensure we adopt best practice techniques in the mainstream IT sector. As a result we chose to investigate a number of technologies before choosing the most appropriate. This investigation is not yet complete, but has allowed to establish a number of technologies that we expect to use, and some of these have been used to set up the initial development environment described in this report.

A non-exhaustive list of technologies investigated are listed in the table below. The selection of widely used open source projects was strongly preferred.

Table 1. List of technologies

Purpose	Alternatives	Preferred choice(s)
Configuration management tool	Ansible Chef Puppet Terraform	Ansible Terraform
Containerisation	CoreOS/rkt Docker Mesos	Docker
Container orchestration	Docker Compose Docker Swarm Kubernetes OpenShift	Kubernetes OpenShift
Security (esp OAUTH and SAML)	Keycloak	Keycloak
Source code control	Bitbucket GitHub GitLabs Gogs	GitHub
Continuous integration and deployment	Jenkins Travis	Jenkins
Artifact repository	Artifactory Nexus DockerHub	Nexus DockerHub
Issue tracking	GitHub JIRA Bugzilla Taiga	GitHub

OpenRiskNet Technical Environment

After reviewing the project's needs and capabilities of the partners involved, we concluded that the overall OpenRiskNet environment will need to handle two core aspects.

1. Runtime environment (referred to as a Virtual Research Environment, or VRE) that allows a partner or third party to start up a new environment for their needs and to deploy a set of OpenRiskNet services to that environment. This could be a long running public environment (such as the OpenRiskNet reference site), private environment running inside a company, or it could be a transient environment created for a specific piece of work and then destroyed following completion.
2. A development environment that allows the pieces that comprise a VRE to be built, as well as the services that can be deployed to such a VRE.

This report due at month six of the OpenRiskNet project describes setting up an initial version of the development environment (item 2). However, this needs to be considered in the context of item 1 which has a timeframe that spans the three year duration of the project.

The overall environment can be expected to have the following components

1. Source code repository for the project's code and deployment configuration (infrastructure as code). We have chosen to use GitHub for this.
2. Systems for issue tracking and technical documentation. GitHub provides an issue tracker and Wiki for this.
3. System for continuous integration (CI) and continuous deployment (CD). We have chosen Jenkins for this.
4. Runtime environment for a VRE, and the systems to operate and manage it. This is currently being investigated within this work package.
5. Services that can be deployed to a VRE. This work commences at month six of the project.

This document describes setting up an initial development environment, which covers items 1-3, but not 4-5.

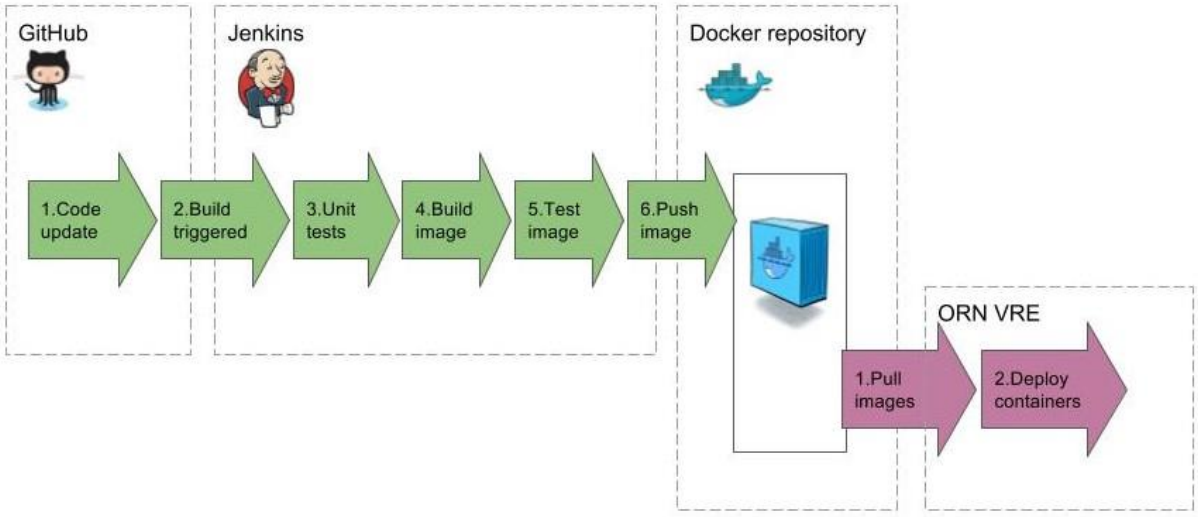

Figure 1. Overview of the development (top line in green) and deployment (bottom line in purple workflows showing how a Docker image that is created by the development process is deployed to a VRE that is set up and managed by a user organisation.

Management of the Technical Environment

Security

An assessment of project needs identified a number of important security criteria.

1. Whilst the project's assets (e.g. source code) will be mostly or completely open access, we still need a good level of security for the project's systems to ensure that our data does not get compromised or hacked. Additionally we want to use Single Sign On (SSO) where possible to avoid the need for users to manage multiple username/passwords, and to avoid the need and risk of managing such information.
2. Security within a VRE is of big concern as potential users for such an environment include pharmaceutical and biotech companies who are typically very security conscious. Not only do we need a SSO solution within a VRE, but we also need this to be based on a robust and widely recognised system that will be accepted by third parties.

To provide the SSO capabilities we plan to use Keycloak that supports OpenID Connect and SAML protocols. Partners and outside parties wanting to provide OpenRiskNet services will need to ensure their systems are compatible with this approach. Details of this will be provided once partners are in the position to adapt their applications to the OpenRiskNet environment.

Backup and recovery

To make management of the environment simple we need to minimise the amount of information that needs to be backed up, but at the same time ensure critical information is not lost in case of events like disk failures. We aim for high levels of resilience and to get systems running again quickly after failures.

The decision to host core information in GitHub is a key part of this, as the project's source code, technical issue management and technical documentation will all be managed by GitHub who already have a higher degree of resilience than this project could reasonably achieve and this is all provided by GitHub with no effort needed from our side. In addition the distributed nature of the Git version control system means that even in the unlikely event of loss of data in GitHub (note, though, that such an event happened with the rival GitLab system earlier this year https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/) each OpenRiskNet developer will have a full copy of the repositories being worked on.

Another core aspect is to use GitHub for configuration information as well as the source code. This follows the best practice of "infrastructure as code" where the configuration of the environment is treated in the same manner as the source code, and stored with the source code. Examples of this would be to store the definition of the CI/CD pipelines along with source code (e.g. using Jenkinsfile) and storing Kubernetes deployment descriptors with the source code in the same manner.

In contrast to the use of GitHub, the CI/CD environment will be treated as being essentially disposable. We expect it to be replaced several times during the project as we move to an environment that also handles deployment of VREs. To allow this we store as much information as possible with the source code (e.g. the Jenkinsfile defining the CI/CD pipeline) and define a minimal amount of configuration that needs to be backed up and restored when a new CI/CD environment needs to be created.

The final part of the development infrastructure is repositories for the artifacts that are built by the CI/CD environment. The most important part of this is a Docker repository, but it is expected that as the project progresses we will also need repositories for the lower level

programming artifacts (e.g. a Maven repository for jar and war files for Java projects). In the short term we will use public repositories (primarily Dockerhub for Docker images) but it is anticipated that we may need to set up our own repositories in the longer term. If so, the contents of these will need to be backed up.

We have started to describe policies and practices for backup and recovery that can be found at https://github.com/OpenRiskNet/home/wiki/CI-CD-environment, which will be continuously updated while these processes evolve as the project matures.

Guidelines for development

We have initiated writing guidelines for development, which are available on the OpenRiskNet wiki: https://github.com/OpenRiskNet/home/wiki/Development-Guidelines. This will be continuously developed and refined during the project.

These guidelines will cover a number of aspects, including:

1. Repository creation
2. Approaches to testing
3. Approaches for defining pipelines
4. Approaches for creating Docker images
5. Approaches for deployment
6. Approaches for configuring containers

Core artifacts should be Docker containers

The tools and services provided by the OpenRiskNet partners are typical in that they encompass multiple technologies and multiple programming languages. Establishing a single standard approach even among the OpenRiskNet partners, let alone the wider community is not practical. However, to be able to integrate these diverse services into an interoperable environment requires some form of standardization. Instead of standardizing the underlying technologies we concluded that we should agree to use a service based architecture (in today's terms often referred to as microservices, but previously referred to as Service Oriented Architecture, or SOA) and to standardize the approaches to deployment by using containerisation. The key benefit is that it does not matter how a service is implemented (which programming language etc.), only what API it provides, and by delivering as containers you can deploy to multiple environments. In essence the service being delivered becomes a black box, and you do not need to worry about what is going on inside the box.

We plan to deploy OpenRiskNet services as Docker containers. Whilst there are other containerisation technologies based around the same Linux Namespaces and CGroups technology, Docker is by far the best established and some partners already have extensive experience of this. Suppliers would provide their services as a Docker image that can be deployed to any OpenRiskNet VRE as a running container. In essence Docker containers are OpenRiskNet's "unit of currency".

Container orchestration

Containers can be run as standalone instances, much like a virtual machine. Often multiple containers need to be deployed together e.g. a web application running in one container need to access a database running in a second container. In simple cases these can be managed separately, but in unison, to create an operational system. However, for all but the simplest systems this becomes cumbersome and error prone. A series of ecosystems have been created

to improve and automate these processes (the overall approach of creating and deploying such environments is often referred to as DevOps). We are evaluating the leading ecosystems to establish what is most suitable to the project's needs. These include Docker Compose, Docker Swarm, Kubernetes and OpenShift. At present it appears that the most suited for our needs are either Kubernetes (backed by Google) or Openshift (RedHat's distribution of Kubernetes with important extra capabilities). Further investigations will be needed to decide the best choice for the project. This will define the runtime environment that an external party will be able to set up and run for their own purposes.

The key to choosing to base our approach around Docker containers is that they are a core component of all the leading orchestration options that are being considered so that we keep our options open while we decide on the best solutions, and in addition we future proof ourselves should the orchestration landscape change in the future (considering the rapid speed of change this is quite possible).

Impact of Containerisation

The use of containerisation has some impact on development processes that need to be considered.

Configuration

A service typically needs some kind of configuration for the environment it is deployed to. For instance, if a service uses a database it needs to know the details (location, username, password, etc.) of the database it will use. The Docker and orchestration systems (e.g. Kubernetes) that we plan to use provide mechanisms for injecting configuration information into containers, for instance as environment variables and configuration files. For services to be deployable to OpenRiskNet they will not only need to be provided (or be built) as Docker images, but will also need to allow for this configuration to be injected so that they can be deployed to individual environments.

Build and test processes

Most software systems that partners will provide will have mechanisms for building and testing the software. Typically this results in building a binary artifact (e.g. a war file for a Java web application) that can be deployed into some runtime environment. However OpenRiskNet deals more with making these systems deployable which goes beyond building and testing the primary software artifacts and onto creating Docker images and mechanisms for deploying these. Consequently, the providers of systems (OpenRiskNet partners and third parties) will need to handle these aspects too. One approach to this, using a Jenkinsfile is outlined below.

Partnering with other Projects

A key aim of the project is to generate a robust and secure e-infrastructure that would be of interest to risk averse organisations such as chemical, pharmaceutical, cosmetics and nanomaterial producing companies. To assist with this we have engaged with RedHat (https://www.redhat.com/) who provide many of the technologies in use in mainstream IT (such as RedHat Enterprise Linux) and are in use in a wide range of organisations including banks. RedHat are willing to assist OpenRiskNet in this area. To date we have had one face-to-face and three teleconferences with RedHat to discuss how we can exploit their technologies and expertise, especially in the areas of container orchestration and security. Additionally, we

are in contact with the PhenoMeNal project and are contacting other EU funded infrastructure project following similar approaches for change of knowledge and experience.

CURRENT DEVELOPMENT ENVIRONMENT IMPLEMENTATION

The OpenRiskNet development is operational and consists of the following components.

GitHub

GitHub is a publicly hosted system that handles many of the common needs of software development. An OpenRiskNet organisation has been set up in GitHub to provide the source code control, issue tracking and developer documentation that is necessary for the project. Access to this is at https://github.com/orgs/OpenRiskNet/. Within this organisation we can create multiple repositories for different aspects of the project (effectively each repository can be thought of as an OpenRiskNet sub-project).

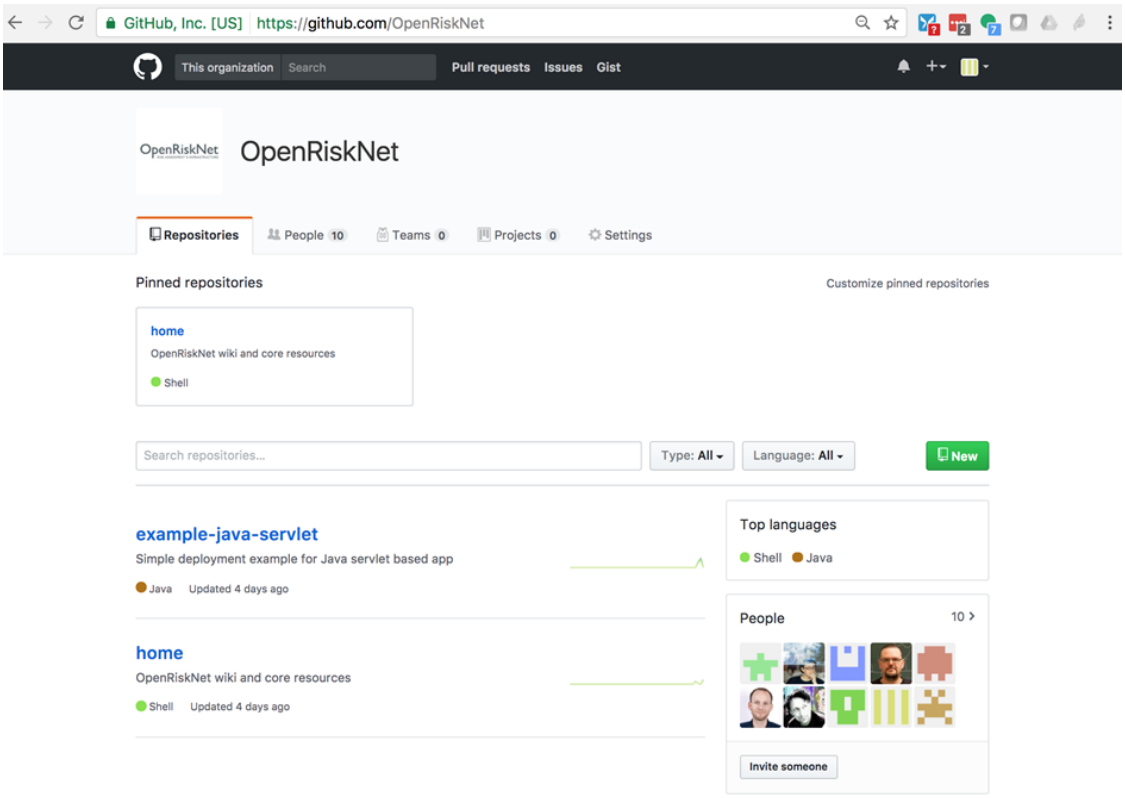

Figure 2. Screenshot of the OpenRiskNet GitHub

Access to these repositories is expected to be public, allowing non-members of the project read-only access which provides good visibility and transparency and allowing anyone to clone or fork any individual repository. Any repository within the organisation could potentially be made private with restricted access should that be required, but this this is thought to be unlikely.

Components of GitHub

Each repository we create has the following aspects:

1. **Source code management**: using the Git distributed version control system

(https://git-scm.com/). This provides us with the ability to manage and version any source code or deployment information for the project. Git has become the predominant tool for this purpose in recent years and most project members are already familiar with it.

2. **Issue tracking**: using GitHub's built in issue tracking system. This is a relatively basic, though widely used, issue tracker, but is thought to be adequate for the project's needs. A repository's issue tracker would be used for managing issues for that particular repository's contents e.g. bugs, feature requests, proposals for enhancement etc. Higher level project level issue tracking is handled elsewhere, using Freedcamp (https://freedcamp.com/Projects_MTP/OpenRiskNet_cMu/todos).
3. **Documentation**: using a Wiki that is provided for each repository. This will be used for technical documentation for that repository, including aspects such as how to build and deploy the repository's contents.

GitHub Access control

To access any GitHub repository a user must have a GitHub account (this is free). A user can then joins the OpenRiskNet organisation (usually by an administrator adding them). Within the organisation that user can be configured as an owner or a member. Most users will be members. Such users can be assigned to "Teams" within the OpenRiskNet organisation. Access rights to individual repositories can be assigned through the team. e.g. there could be a "Core infrastructure" team that had read-write access to certain repos, but provide read-only access to another team.

The GitHub Organisation also allows for OAuth authentication from external systems. This is used by the Jenkins CI/CD system (see below) to authenticate against GitHub avoiding the need (and risk) to manage usernames and passwords.

Modes of operation

The OpenRiskNet project is atypical for a normal software project in that it focusses more on deployment and interoperability of existing software tools than development of new software from scratch. Though we may develop new infrastructure services (e.g. for service discovery) we are not primarily developing new computational tools (e.g. a new predictive modelling tool), but are instead incorporating existing services of the project partners. But to deploy these we need to at least generate additional deployment information (e.g. build/deployment pipeline, Kubernetes descriptors). Of most importance to OpenRiskNet are the Docker images (either built by OpenRiskNet or pulled from other repos) and deployment descriptors that allow those services to be orchestrated. This impacts the organisation of projects. We see a number of possibilities. Partners have been asked to establish that at least one of these models is suitable for their purposes, and this seems to be the case, but we can't rule out the need for additional approaches to cover other scenarios.

Core project code

Option 1: Code directly related to the core infrastructure (e.g. discovery service code) that could potentially be "owned" by the project rather than any one partner.

This code would want to reside in an OpenRiskNet GitHub repository and all aspects of the code (including deployment descriptors) would be part of that repo. Docker images would be built and deployed through the CI/CD system.

Code from partner with existing repo

Code for an existing project/service (e.g. Lazar, Jaqpot) that is owned by one particular partner and already resides in their own repository. The partner would not want to transfer this to the OpenRiskNet repository, but the OpenRiskNet project might need that code to be extended in some ways (e.g. create a Jenkinsfile, add Kubernetes descriptors).

To handle this the OpenRiskNet repository could fork the master repo which would act as an upstream project. To add the additional info one of the following scenarios could happen:

Option 2.1: All changes are made in the partner's own repository and the CI/CD process accesses this directly, with no need to fork the repository.

Option 2.2: The partner who owns the project makes all the changes needed (e.g. enhances their own version) and these get pushed to the fork where the software artifacts and Docker images are built and deployed.

Option 2.3: The enhancements get made in the OpenRiskNet repo and stay there. Changes the partner makes to their code in the upstream repo are merged into the OpenRiskNet repo, but additions related to deployment made in the OpenRiskNet repo do not go back to the project owner's repo.

Option 2.4: As for 2.3 but the enhancements do get pushed back (preferably via pull requests).

Alternatively the partner does not provide the code, but provides Docker images (e.g. through DockerHub). In which case this is an example of Option 3.2.*.

Partner wanting to create new project for OpenRiskNet

Most likely they would do this in their own repo, in which case it would follow one of the Option 2.* patterns. If instead they wanted to do this in the OpenRiskNet repo then it would follow the Option 1 pattern.

Integrate external tools/services

A web service accessible somewhere on the internet

An example would be the ChEMBL REST services.

Option 3.1: To add this to the OpenRiskNet services we would at the minimum need to provide some information that allowed the service to be discovered, and possibly create a proxy for the service that ran with the OpenRiskNet system. The information needed would be fairly minimal, so we might just have one OpenRiskNet repository handling all of these, each one being a small sub-project in that repository.

A docker image (e.g. in DockerHub)

This is somewhat similar to Option 3.1 except that the service is provided by a Docker image that is fired up within the OpenRiskNet runtime environment. But essentially all is needed is configuration information so the solution is the same as Option 3.1.

There are two variants of this:

Option 3.2.1: Permanently running services, with the container being created when the service (environment) starts and destroyed when it terminates.

Option 3.2.2: The container is created for purpose of running a single job and then terminated. Most likely this could be orchestrated through a REST web service so gives the appearance of being permanently running.

The difference is important, but should not affect how the project operates in GitHub.

Jenkins

Key to achieving interoperable, stable as well as easy to deploy and use software services will be continuous integration (CI) and continuous deployment (CD) capabilities. The CI aspect covers the ability to rebuild and test the software artifacts from the project partners (and external parties) whenever changes are made to the source code and the CD aspect cover the ability to deploy those tested artifacts to public repositories and to runtime environments where they can be used.

OpenRiskNet will be handling the ability to create runtime environments (a VRE) to which the OpenRiskNet services can be deployed, and allowing partners, third party organisations and individuals to set up and run their own VRE. We have been investigating basing these VREs on Kubernetes (https://kubernetes.io/) or Kubernetes plus OpenShift (https://www.openshift.com/). A full discussion of deployment to VREs is out of scope for this document and will be covered in later reports.

This deliverable report covers only the development infrastructure, not the deployment to runtime systems, so the focus is primarily on the CI aspects, but this is being done with future CD needs in mind.

We have chosen to use Jenkins (https://jenkins.io/) for the project's CI/CD needs. Jenkins is a leading open source system for CI/CD and several partners already have experience in using it. As the more complete infrastructure evolves Jenkins will remain a key part of this as its CD capabilities allows to deploy tested artifacts to the runtime environments where they can be access by end users.

To meet the initial development needs we have set up a Jenkins environment that can be accessed here: http://jenkins.openrisknet.org/. This system can be summarised as follows:

- Single VC1M server (4GB RAM and 4 cpu cores) running on the Scaleway cloud (see https://www.scaleway.com/).
- Ubuntu Xenial 64 bit operating system.
- Latest Jenkins and Docker software installed
- Access control configured using GitHub OAuth authentication via the Jenkins GitHub Authentication Plugin (https://wiki.jenkins-ci.org/display/JENKINS/Github+OAuth+Plugin)

Key aspects of this environment are:

- It should be thought of as a temporary and disposable environment. It will be replaced by equivalent environments as the project evolves to include a more complete runtime environment.
- Creating of this environment is currently a manual process and can be done in approx 60 mins. Instructions for doing so can be found here: https://github.com/OpenRiskNet/home/blob/master/jenkins/setup.sh
- To permit this disposability configuration needs to be kept to a minimum so that little information and data needs to be transferred to a new environment.
- Using GitHub authentication provides a Single Sign On environment that encompases GitHut and Jenkins and avoids the need and risk of managing security credentials.

EXAMPLE PROJECTS

To demonstrate the operation of the development infrastructure we have created some simple projects, purely for the purpose of illustrating the development process. Work on adapting partners projects does not start until month six of the project.

The purpose of these projects is to illustrate some of the modes of operation outlined above up to the stage of building a docker image and pushing it to a public Docker repository. As such they can be thought of a blueprints for how partners will be able to handle their more complex projects as the work progresses.

Example 1: simple Java servlet example

This project follows pattern 2.1 outlined above. It is a simple web application written with the Java programming language and the Java servlet API. It can be summarised as follows:

- Simple Java servlet web application that responds to an HTTP get operation (e.g. request from a web browser) with a simple "Hello world" message.
- The "Hello" part of the message is configurable using an environment variable to illustrate the need for injection of configuration information from the runtime environment. For instance, if the GREETING environment variable is set to "Hiya" the message generated is "Hiya world"
- The "world" part of the message is determined based on the authenticated user to illustrate the need for being able to externally configure authentication. If the user is authenticated the value is the username, otherwise it is "unauthenticated user".
- The software is built using the gradle build system. Other systems such as Maven or Make files could easily be used. The primary build artifact is a war file that can be deployed to a servlet container.
- An Arquillian (http://arquillian.org/) test checks that the servlet works as expected.
- A Dockerfile can be generated and a Docker image built from that.
- The Docker image is deployed to the public Dockerhub repository (https://hub.docker.com/r/openrisknet/example-java-servlet/)
- That image can be run (assuming docker is already installed) using the command docker run -d -p 8080:8080 openrisknet/example-java-servlet and then accessed through a web browser at http://localhost:8080/
- The CI/CD process is driven by a Jenkinsfile that is included with the project source code. This allows the definition of the build pipeline to be encapsulated within the project's code. Jenkins automatically detects the Jenkinsfile and runs the pipeline accordingly. Even if Jenkins is changed to something different in the future the details of the build pipeline are captured allowing re-use.
- A GitHub commit hook triggers the Jenkins build automatically whenever the source code is updated, resulting in a new image buing pushed to Dockerhub without any human intervention.
- The project source code can be seen here: https://github.com/OpenRiskNet/example-java-servlet
- The build pipeline can be seen here: https://jenkins.openrisknet.org/blue/organizations/jenkins/OpenRiskNet%2Fexample-java-servlet/activity

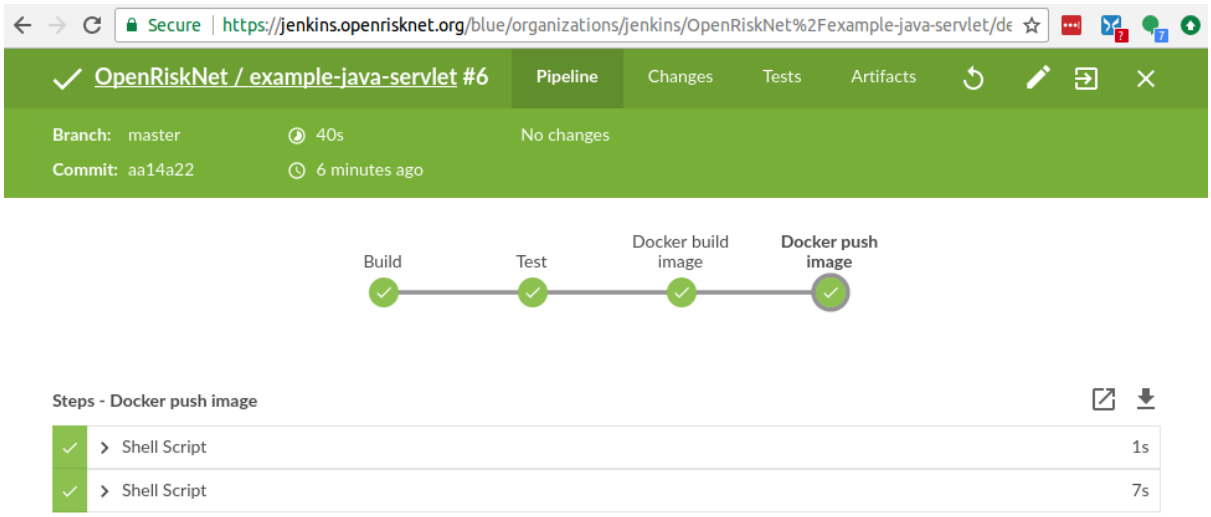

Figure 3. Java servlet example

As such, this project illustrates how building, testing and deploying a project can be completely automated. We anticipate that most partners will already have projects that allow them to be built and tested (the CI part), but that they will need to define how Docker images are built and how the whole pipeline is orchestrated (e.g. by means of a Jenkinsfile).

Example 2: simple Python project

This projects implements the pattern described in 2.2 above. This repository is a tracking fork of an open source web service by one of the consortium partners that allows chemical identifier conversions using a REST like interface using the popular rdkit library.

- As is planned for all OpenRiskNet projects, the web service provides an OpenAPI (aka Swagger) definition describing the service
- Several bidirectional conversions between common identifiers are possible (e.g. SMILES <-> InChI). Where necessary, the CACTUS third party web service is queried for database lookups (e.g. converting from CAS to other identifiers)
- The software is implemented in python and does not need an initial compile step. The pipeline is thus simpler, specifying just the docker build script
- The docker container can be run with the following command: docker run -it -p 8080:8080 douglasconnect/chemidconvert-orn
- As in the first example, the jenkins configuration is done by including a Jenkinsfile in the upstream github repository
- The source code can be found on github: https://github.com/DouglasConnect/ChemIdConvert
- The build pipeline can be seen here: https://jenkins.openrisknet.org/blue/organizations/jenkins/chemidconvert/activity

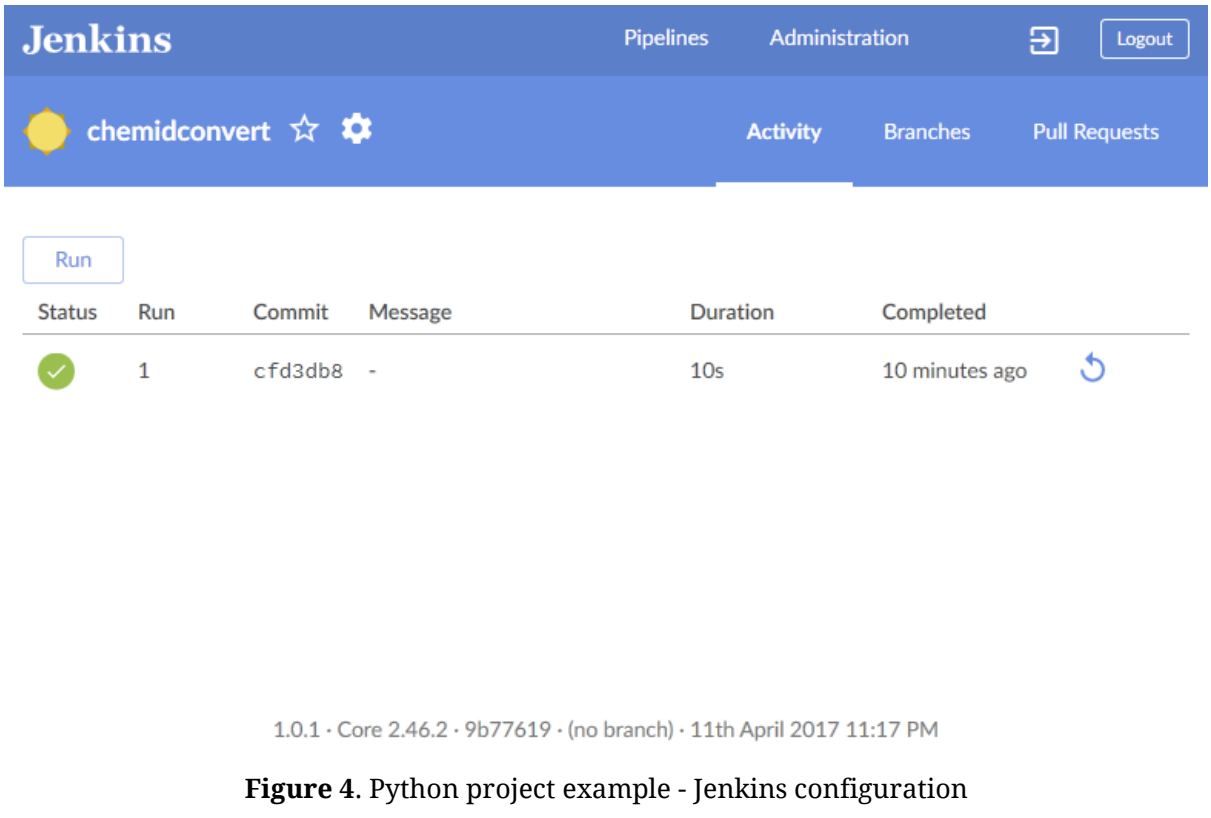

Figure 4. Python project example - Jenkins configuration

RISK ASSESSMENT

Finally we list a set of identified risks and how we plan to mitigate them based on the development infrastructure outlined before and the flexibility inherent in the system.

Risk	Likelihood (1-3, 1 lowest)	Impact (1-3, 1 lowest)	Comment/plan
Developers do not use the infrastructure due to its complexity	1	3	Choose established solutions. Keep up-to-date developer documentation Offer training sessions.
Chosen development components (frameworks/systems) are not sustained	1	3	If possible, pick well-established solutions with a large community. Have a backup plan for remaining solutions.
Development infrastructure has downtime	3	1	Have warning systems in place and clear responsibilities. Allow for mirroring infrastructure. Choose external solutions with recognized good uptime.
Development infrastructure does not have sufficient resources to sustain proper CICD	2	3	Choose a scalable back-end IaaS provider and implement a system that can scale with this.

CONCLUSIONS

This Deliverable presents OpenRiskNet Development Infrastructure as of M6. A survey and testing of tools and frameworks that can sustain development within the project has lead to decisions on frameworks and systems, which has been deployed and is operational. There is no doubt that this will be further developed/refined during the course of this project. In particular we will investigate to extend the infrastructure towards a more scalable implementation, and also establish a continuous integration and continuous deployment (CICD) system with testing, staging, and deployment. To this end we will in particular evaluate OpenShift (by RedHat), which comes pre-packaged with a CICD system, and natively supports containerization of open source repositories.