OpenRiskNet

RISK ASSESSMENT E-INFRASTRUCTURE

Deliverable Report D3.2

First documentation of the core e-infrastructure

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
Start 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) P12 Vrije Universiteit Amsterdam, Netherlands (VU)

Deliverable Report identification

Document ID and title	Deliverable 3.2 First documentation of the core e-infrastructure
Deliverable Type	Report
Dissemination Level	Public (PU)
Work Package	WP3
Task(s)	Task 3.1
Deliverable lead partner	IM
Author(s)	Tim Dudgeon (IM), Alan Christie (IM), Daniel Bachler (DC), Lucian Farcal (DC), Harry Sarimveis (NTUA), Philip Doganis (NTUA), Pantelis Karatzas (NTUA), Ola Spjuth (UU) Reviewed by Iseult Lynch (UoB) and Barry Hardy (DC)
Status	Final
Version	V1.0
Document history	2017-11-16 Draft version 2017-12-06 Final version

Table of Contents

SUMMARY

This report describes the first documentation of the core OpenRisknet e-infrastructure with examples of an initial development status implementation. This report forms part of this documentation, along with other parts located in the OpenRiskNet GitHub repository. This documentation describes the creation of the e-infrastructure and the deployment of the first partner application to the e-infrastructure.

This documentation will be updated throughout the project.

The work described in this report addresses Task 3.1 Documentation of Core Infrastructure and Deployment in WP3, but also covers the following tasks in WP2 from the description of work:

- Task 2.1 Creation of development environment and D2.1 Creation of development environment - An initial development environment was described at Month 6. The work described here extends this by moving the Jenkins CI/CD environment from a standalone system to one running in the OpenRiskNet development reference site, and by providing more advanced CI/CD capabilities available through OpenShift;
- Task 2.3 Establish security environment - This work describes the security environment that has been set up for the development reference site, including access control for both administrators and developers in the reference site, and a separate system (based on Keycloak) for end users accessing the applications deployed to the development reference site;
- Task 2.4 Services discovery - The development reference site provides some underpinnings for service discovery by means of the built-in capabilities of OpenShift. Creating services with the appropriate labels and annotations should allow them to be discovered by end Users and the broader research communities;
- Task 2.5 Deployment of virtual infrastructures and container orchestration frameworks - This has essentially been achieved (at least in an initial form) by the creation of the development reference site which runs on a virtual infrastructure (OpenStack) and allows orchestration of containerised applications through OpenShift. The deployment of the Squonk Computational Notebook to this infrastructure illustrates this;
- Task 2.6 Establishment and maintenance of OpenRiskNet reference instance - The development reference site is the first instigation of the OpenRiskNet reference instance. A production site will follow later in the project.

INTRODUCTION

This report follows on from the report for deliverable D2.1 “Development infrastructure online” which described an initial setup for development of the OpenRiskNet e-infrastructure. This report describes the setup of a more complete e-infrastructure that supports deployment and operation of applications for safety and risk assessment as well as their development. The environment is at a preliminary stage but already supports the ability to provision to physical servers and cloud environments, an ability to scale across multiple servers, provisioning of core infrastructure needed by applications and the deployment of an initial OpenRiskNet partner application.

This new environment replaces the standalone Jenkins environment described in the D2.1 report.

Most documents that are described in this report can be found in the ‘home’ GitHub repository of the OpenRiskNet project [1] and are publicly accessible.

CORE E-INFRASTRUCTURE

Objectives

As an e-infrastructure project, OpenRiskNet aims to create computational infrastructure and software tooling to support the processes of chemical risk assessment. One key aspect of this is providing tools that facilitate the ability to create a computational environment in which this scientific work can be performed. OpenRiskNet not only aims to provide such an environment as its reference site, but also to allow third parties to create their own environments (referred to as a Virtual Research Environment, or VRE) on their own internal or cloud infrastructures using the OpenRiskNet platform.

Creating such a VRE should be relatively straightforward for a skilled IT administrator, and full guidance notes are provided (via the OpenRiskNet Wiki - see also D3.3). It should be possible to deploy this environment to a range of hardware, both physical and cloud-based, ranging from single server installations to powerful multi-server distributed environments providing substantial computational power.

OpenShift Architecture

Background

After researching several suitable technologies, we chose to base the infrastructure on Docker, Kubernetes and OpenShift technologies.

Docker [2] is the leading containerisation technology available today. Containerisation allows packaging software tools and services, and all their dependencies, into units (containers) that can be readily started, stopped and interconnected. Containers can be considered similar in some ways to Virtual Machines, but are much more lightweight and efficient. Most modern software deployment approaches are converging on containers and Docker, and as such this technology not only provides us with an effective mechanism for deployment, but also future-proofs the work as, even if other tiers of the framework might change, it is very likely that Docker containers will still be at the heart of most approaches over the coming years. We also note that several other European e-infrastructure projects including INDIGO-Datacloud [3], PhenoMeNal [4], and the recently-formed EOSC Hub have container technology as a core component of their software architecture.

Whilst Docker allows containers to be deployed to an individual computer (server), high performance environments that are needed to support complex scientific workflows or larger number of users require workloads to be distributed across multiple servers. Thus there is the need to orchestrate the deployment and inter-connectivity of containers across multiple servers (often in the range of 10 - 100 servers, but potentially in the 1000s). Thus, a distributed container orchestration platform is needed. This can be thought of as a distributed operating system. The two leading solutions for this are Docker's Swarm, and Google's Kubernetes (see [5] for a comparison of these two technologies). Investigations led us to choose Kubernetes as it is more capable and has larger traction in the marketplace, and some partners already had experience with using it.

Whilst Kubernetes can provide this distributed operating system, various additional components are needed for a fully functional system. Red Hat's OpenShift [6] is a

distribution of Kubernetes that adds many of these additional aspects, such as Continuous Integration and Continuous Deployment (CI/CD), a strong emphasis on security, and the option of a commercial support model. Consequently we have chosen to base the OpenRiskNet Virtual research Environment around Red Hat's OpenShift, and Red Hat have joined the OpenRiskNet Associated partner program (see Deliverable D1.2) as a technology partner to assist us with delivering this platform.

Overview of OpenShift

As mentioned, OpenShift is a distribution of Kubernetes with additional components added so can be thought of as an extension of Kubernetes, with the core "engine" being Kubernetes but with some OpenShift specific add-ons.

An OpenShift cluster comprises one or more nodes (servers) providing the following:

- Master node(s) that controls and monitors the cluster and provides a REST API and web console that the developer or administrator interacts with;
- Etcd node(s) that records system state that is manipulated by, and used by, the master node(s);
- Infrastructure node(s) to which the core OpenShift infrastructure components are deployed;
- Worker node(s) to which the OpenRiskNet software tools are deployed.

An OpenShift environment executes software as containers (typically Docker containers) running in a "Pod", which is the fundamental execution unit in Kubernetes. A pod is scheduled to run on a particular node, and can be configured to restart if it fails, and multiple pods can be run in parallel to allow the application to scale.

Pods do not normally exist in isolation, but are managed by a Replication Controller. Pods are normally accessed through a Service that allows access to the pods from within the OpenShift cluster. Access from outside the cluster (e.g. from your web browser) is provided by a Route which acts as a proxy for the service. Pods, Replication Controllers, Services and Routes are just a small number of types of component (knows as API objects) that can be present in an OpenShift cluster. A full description can be found in the OpenShift architecture documentation [7].

Description of deployment types

The master, etcd, infrastructure and worker functions mentioned above can be deployed in a variety of ways. OpenRiskNet aims to support several of these.

All-in-one

All functions deployed to a single server. This provides a very simple environment that is suitable for getting started, for testing or for some simple studies, but not suitable for long term production environments or computationally demanding workflows.

A special case is Minishift, a shrink-wrapped distribution of OpenShift that is suitable for deployment to a standard laptop.

Standard Availability

Here the functions are deployed to multiple servers (nodes) providing some scalability (including multiple worker nodes for performing computationally demanding work), but most functions only have a single instance so there are multiple potential points of failure. It does however allow a moderately capable environment to be created whilst being economical with computing resources. While there are several variants of this, a typical environment is as follows:

1. One node running the master and etcd
2. One node running the infrastructure components
3. One or more worker nodes running the OpenRiskNet software tools.

If resources are particularly limited the master, etcd and infrastructure components could be run on a single node, or the infrastructure components could be run on worker nodes.

The OpenRiskNet environment plans to support basic standard availability scenarios.

High Availability

Production environments should aim to minimise the amount of downtime needed for maintenance or caused by hardware or software failure. Such high availability (HA) environments need redundancy of components so that if one fails or is taken down for maintenance other(s) are still present to allow operations to continue. A general rule is that there should be three of each component (where HA is imperative this can be extended to five of each component). As such, an ideal HA environment would have 3 masters, 3 etcd, 3 infrastructure and any number of worker nodes. In addition, a load balancer is needed to distribute requests (e.g. to the master) across the nodes.

Whilst this provides much improved robustness it comes with a significant extra cost of resources.

The OpenRiskNet environment plans to support a basic high availability scenario that an administrator can use to create a VRE for their organisation.

Autoscaling

Scientific workflows are slightly unusual in that there are typically relatively few end users, but they can run very computationally demanding jobs. Contrast this to typical consumer applications where there may be a very large number of concurrent users, but the computational demand from each user is relatively modest. This results in scientific workflows often being much more “lumpy” in their resource needs, often requiring lots of computational power for a short period, followed by periods of relative inactivity. Keeping sufficient resources permanently available is costly and inefficient. Instead it is better if the underlying computational resources can be scaled in and out on demand. This is particularly appropriate for cloud based environments where you only pay for the resources that are being used.

The OpenRiskNet environment plans to support basic auto-scaling capabilities allowing compute intensive jobs to be run over multiple servers. The extent of this auto-scaling will be configurable by the administrator of the VRE.

Persistent Storage

Most applications need to store data in some way, and this involves writing to permanent storage (files or a database) that will ultimately be backed by disk storage. In a distributed

computing environment this can be tricky to achieve as services can potentially be deployed to any node in the cluster, and can move between different nodes over time. Either the service has to be deployed to where the storage is available or the storage has to be able to "follow" the service and be accessible from wherever the service is deployed.

OpenShift supports a number of persistent storage options. We are investigating 3 of these:

1. NFS where a single file server provides storage to other servers across the network.
2. GlusterFS where a series of storage servers provides storage to other servers across the network. This scales better than NFS but requires significantly more resources.
3. Cloud native storage where the native storage mechanism (e.g. Cinder for OpenStack, EBS for Amazon Web Services) is use to mount volumes to the node on which the particular service that needs it is located. This is ideal for services such as a database that needs fast locally attached storage, but can only be attached to a single node at any time.

Further information can be found in the OpenShift persistent storage documentation [8].

NFS is generally most suited for small, standard availability environments, GlusterFS for high availability and large clusters, and Cloud native storage for any cloud based environment, but only for certain purposes.

Core OpenRiskNet Infrastructure Components

An OpenRiskNet VRE will comprise a number of core components that provide the basic infrastructure and support to other components (typically applications) that provide the scientific content. These core infrastructure components will include:

Single Sign On

Most components that an end user interacts with will require some form of authentication and authorisation. The user experience is greatly improved (and the effort needed to manage it) if the system supports Single Sign On (SSO) so that the user logs into a single centrally managed system (e.g. by specifying their username and password). That login is used to control access to the multiple systems available. Logging in once logs the user into all applications, though the level of access (e.g. read or write access) can differ for each application.

OpenRiskNet will provide a SSO environment through the Keycloak project [9] from Red Hat. Keycloak is an open source "upstream" project that forms the basis of their Red Hat SSO [10] commercially supported product. Keycloak is already provisioned as part of the initial OpenRiskNet reference site.

Metrics and Logging

When administering a complex deployment like a VRE, it is important to be able to track and monitor what is happening within the system. When the system is distributed across

multiple servers this can become tricky, with log file and usage metrics being distributed across the servers in different places.

Consolidated logging provides a mechanism to consolidate logs from the nodes and the pods running the software into a single searchable database. OpenShift provides an implementation for consolidated logging based on Elasticsearch (as the searchable database for the logs), Fluentd (for consolidating the logs from the different sources into the database), and Kibana (as a query tool and dashboard for examining the logs). More information can be found in the Openshift aggregated logging documentation [11].

Consolidated metrics provides a mechanism to consolidate runtime metrics such as memory usage, CPU utilisation and network usage into a single place so that resource utilisation can be monitored and assessed. Services can be auto-scaled based on these metrics (e.g. to scale out the pods or the nodes to support higher computational loads as needed). OpenShift provides an implementation for consolidated metrics based on a Cassandra database, Heapster and Hawkular. More information can be found in the OpenShift cluster metrics documentation [12].

Consolidated logging and metrics are already provisioned as part of the initial OpenRiskNet reference site.

Certificate management

Providing secure web sites using encryption of the data in transit using HTTPS as opposed to unencrypted using plain HTTP is considered good practice, and is now expected to be the norm for web based applications. HTTPS (actually usually TLS) encrypts the pages using public/private key pairs that form part of a certificate that a web browser can inspect to determine if it should trust that site. Certificates can include a chain of trust back to a trusted root certificate authority (CA) that web browsers are preconfigured to treat as trusted sources (this is achieved by signing a certificate with another certificate forming a chain of trust that extends up to a trusted root CA). Self signed certificates allow encryption but do not provide this chain of trust, so browsers typically provide warnings to the user that the web site is not trusted, which most users find disturbing.

To prevent users being warned about the OpenRiskNet reference site (and any VRE that a third party deploys) not being trusted we need to ensure that trusted certificates are obtained and deployed for all public facing services (typically those exposed as OpenShift routes) and also that they are updated before they expire. This is a non-trivial exercise.

Historically, trusted certificates are obtained from an organisation such as Verisign that charges for this service. Typically this process is partly manual (e.g. involving sending emails). This process is neither suitable for a VRE where we want a high degree of automation, nor cost effective for academic groups or groups with low budgets.

More recently Let's Encrypt [13] was set up with support from many mainstream IT companies to provide a public, free and transparent mechanism for generating trusted certificates, with the processes for generating and renewing certificates being completely automatable. As such it is ideally suited for the needs of the OpenRiskNet VRE.

OpenRiskNet aims to provide a mechanism to generate Let's Encrypt certificates for all its public facing services. This has partly been done for the initial reference site.

Jenkins and CI/CD

Continuous Integration and Continuous Deployment (CI/CD) have become a key part of modern software development practices. OpenRiskNet wants to allow provision of CI/CD within a VRE, and will include this in the OpenRiskNet reference site. This will allow the VRE to be used not only for deploying working software but also for building and testing it in the first place (e.g. building and testing a deployable version of a piece of software every time the source code or any dependencies are updated).

It is expected that a typical VRE will not utilise the full CI/CD capabilities, but will just deploy software that has been built and tested by another VRE (primarily the OpenRiskNet reference site), but where this is wanted it should be possible to provide.

One of the key additions that OpenShift provides over plain Kubernetes is CI/CD capabilities. This comes in multiple flavours, of which 2 are the most important.

1. Source2Image (s2i) that uses Docker containers as a “build machine” that generates the resulting Docker images containing the packaged software, and to deploy to a Docker image repository (either within the OpenShift cluster or to a public repository such as Docker Hub)
2. Jenkins [14] that provides a mechanism for defining and running more complex CI/CD pipelines, but achieving similar goals.

Source2Image is built into OpenShift so is automatically available, and will be suitable for many build processes. As many OpenRiskNet partners already use Jenkins pipelines we also want to provision Jenkins to a OpenRiskNet VRE to make it simple for partners and 3rd parties to migrate their built CI/CD process to an OpenRiskNet VRE.

The initial OpenRiskNet reference site has Jenkins provisioned, and also provides full s2i capabilities.

OpenRiskNet End User applications

The prime purpose of an OpenRiskNet VRE is to provide access to applications that an end user (e.g. chemist or safety assessor) can use for chemical risk assessment. As such the most important components will be these end user applications provided by the OpenRiskNet partners and third parties.

As the state of development of the infrastructure is at a very early stage, work in the deployment of end user applications has only just started and is currently very limited in scope. However, many of the requirements necessary for this and the procedures needed have been identified.

Key to this is the need to provide applications in the form of Docker images, and to define OpenShift templates and/or Kubernetes Helm charts that describe how the container image will be deployed to the OpenRiskNet VRE and what resources it requires.

Squonk

The Squonk Computational Notebook is a product from IM that was chosen as the prototype application to deploy. It was chosen as it was already fully containerised and designed for environments like Kubernetes and OpenShift. Nonetheless, some significant work was needed to allow the Squonk Computational Notebook to be deployed to

OpenShift, and some further work is still required to complete the process.

Planned applications

Applications from other OpenRiskNet partners that are expected to be incorporated into the OpenRiskNet VRE environment include:

- NTUA Jaqpot modelling platform [15]. A collection of RESTful webservices for QSAR modelling, read-across, dose-response modelling, biokinetics modelling, optimal experimental design and interlaboratory comparison.
- JGU WEKA REST Service. RESTful API webservice to WEKA Machine Learning Algorithms.
- lazar (nano-lazar) (Lazy Structure- Activity Relationships) a modular framework for predictive toxicology.
- ToxCast and ToxRefDB data APIs with more to be added in the future.
- Conformal Prediction toolkits and web GUI for QSAR modeling.

In addition we plan to incorporate third party applications. Some of these will come from members of the associated partner program, where the core OpenRiskNet partners will assist those associates to incorporate their tools, and some may be third party tools from organisations with no formal association to OpenRiskNet. We anticipate that these additional applications might include:

- Jupyter Notebooks

The ultimate aim is that a third party can provide what is necessary to deploy their application to an OpenRiskNet VRE (by means of providing the container images and deployment templates) purely from the documentation and examples that OpenRiskNet provides.

Interoperability

A key objective of the OpenRiskNet project is to provide interoperability between applications within a VRE, so that, for instance, a service from one application can be consumed by a client from another application without either application needing to know of the other's existence.

Work is ongoing to address interoperability, such as activities described in report D2.2 "Initial API version provided to providers of services", and the work of the Ontologies Task Force. These are not described in this report as they will be covered elsewhere (e.g. D2.3, D4.2, D4.3). However they are a key objective of the overall project, and can be considered an implicit part of the work in this report.

Deployment procedures

The mechanism for deployment into a running VRE, in simple terms, is the same for infrastructure components and end user applications that run on top of that infrastructure:

1. Create or locate container image(s) for the specific application(s) to be incorporated;

2. Generate deployment templates;
3. Deploy using those templates.

Create or locate container image(s)

The process of generating the container images is outside the scope of the OpenRiskNet project as most organisations will have their own development procedures and we do not wish to change these. However a VRE may offer the ability to run the build process that will create container images. The OpenRiskNet reference site will provide such facilities.

Whatever the mechanism, the end result of step 1 will be container image(s) accessible from a registry such as Docker Hub or the OpenShift container registry, running within a VRE.

Generate deployment templates

Kubernetes and OpenShift work on the basis of “API objects”. These are objects that defined the working parts of an application. Examples include the Pods which run the actual containers, the Services that provide access to those Pods from other Pods in the systems, and Routes that provide external access to those services. There are many other types of API objects that are relevant. See [7] for more details.

These API objects are typically defined in files in JSON or YAML format and can be created directly using the CLI (the oc tool), the web console or the REST API. However it is usually more flexible to create API objects by using templates that allow easy parameterisation of the process e.g. for generating passwords or configuration options.

This process of generating templates that describe how the application (e.g. its container images) should be deployed is a critical part of the overall process. Complete examples are described in the Recipes and Deployments sections. A simple example is outlined in Annex 1.

Deploy using those templates

As indicated above, templates are deployed to the OpenShift environment, typically using the CLI, but deployment can also be done using the web console or the REST API.

The person doing the deployment is not necessarily the same person as the person who creates the container images and templates, especially where elevated privileges are required.

Typically applications are deployed into separate “Projects” (equivalent to a Kubernetes Namespace) allowing isolation of application and security aspects. OpenShift provides a comprehensive security and privileges infrastructure, but describing this is out of scope for this document.

Recipes

As described above, generating the deployment templates is one of the most critical aspects and will be a process that is new to most developers. These templates can range from very simple to quite complex. To aid the process of learning how to generate these

templates and to disseminate this knowledge to partners, Associated partners and ultimately to third parties (Virtual Access Users) we have taken the approach of creating 'recipes' for deploying applications to an OpenRiskNet VRE (based on OpenShift). These describe the templates needed and the steps needed to deploy. As such they provide streamlined (e.g. partly automated) examples that can be followed. They allow a user to run through the whole process and to get a good understanding of what is involved. As such they are expected to be primarily learning exercises. Contrast this with the Deployments section below.

In addition, we have generated recipes for deploying OpenShift environments of different types into which these applications can be deployed. Different types of environment are covered. Again these are primarily aimed as learning exercises, and should be contrasted with the Environments section below which provide more concrete examples.

Both types of recipes will continue to be created and enhanced during the lifetime of the OpenRiskNet project.

Both types of recipe are contained in the OpenRiskNet GitHub repository (https://github.com/OpenRiskNet/home/tree/master/openshift), and are publicly accessible. See the README document [1] for current details, though at the time of writing some (not all) of the recipes include:

OpenShift deployment:

- Creating a Minishift environment
- Creating a single server "All-in-one" environment
- Creating a multi server environment.

Application deployment:

- Deploying a PostgreSQL database
- Example deployment of Wordpress and MySQL
- Deploying the 3rd party CDK Depict application
- Deploying Keycloak for SSO
- Deploying example servlet application using SSO
- Deploying Django application using SSO
- Securing routes with TLS using ACME controller.

Uppsala workshop

A workshop was held at Uppsala University on 25-26 September 2017 for partners to gain experience with aspects covered in this document. The principles of Docker and Kubernetes were explained, and the recipes described above were used for hands-on sessions where the participants tried these out in practice. The workshop was attended by participants from partners IM, DC, UU, UM, NTUA, JGU and CRG.

Other aspects that were covered were:

- IM outlined a plan of what would be needed for adapting the Squonk Computational Notebook for deployment to an OpenRiskNet VRE.
- CRG provided an overview of Nextflow and how it can be used to orchestrate complex multi-step workflows to an environment like an OpenRiskNet VRE.
- DC led a discussion on providing common APIs and a service for discovering those APIs.
- UU presented KubeNow developed in the PhenoMeNal H2020 consortium for

cloud-agnostic instantiation of on-demand Kubernetes clusters.

Environments

Whilst the recipes described above aim to provide training materials for how to create an OpenShift environment, more concrete examples are needed to provide real instances that can be used to deploy real test or production environments. To achieve this we have created an “Environments” section that will provide concrete examples of how to deploy environments of particular types. Currently, there is only a description for creating a standard availability (no redundancy) environment to an OpenStack cloud platform, but we plan to create recipes for high availability environments and to handle other cloud platforms like AWS.

The overall process of setting up an environment in OpenRiskNet is similar for all environments. The differences mostly relate to the number and types of nodes being installed and specific details of the deployment. The general process is as follows:

1. Create the necessary networking environment so that the hostnames of all servers that will be part of the cluster can be resolved from each server.
2. Create a bastion server in the network. This is a machine that the administrator can connect to using SSH and from where they can manage the cluster.
3. Provision the necessary servers (either physical or virtual) that will form the cluster. Centos7 is used as the base operating system, though Red Hat Enterprise Linux can also be used with a paid for subscription.
4. Deploy the additional RPM packages required by OpenShift.
5. Create an Ansible inventory file that defines the details of the cluster. This is the key configuration of the cluster.
6. Run the Ansible playbook provided by OpenShift that provisions the cluster using the inventory file. This deploys the necessary software to different nodes (e.g. master node, infrastructure node etc.), configures the servers and starts all the necessary services.
7. Optionally run a set of diagnostics provided by OpenShift to check that the cluster is functioning correctly.

Full details are present in each of the Environment definitions in GitHub. The expectation is that these can be run by any person who is experienced in system administration.

Currently, many of these steps are somewhat manual in nature, but work is ongoing to provide more improvements in terms of:

- Using the TerraShift project created by UU to better automate and standardise the provision of the underlying servers onto which OpenShift will be deployed.
- Further automation most likely in the form of Ansible Playbooks.
- Establishing processes for backup and disaster recovery.
- Establishing processes for managing and scaling the cluster.
- Better establishing procedures for providing persistent storage.

Details of these environments can be found in the GitHub repository. They will be updated and added to during the project.

Deployments

As Environments provided concrete examples of creating an OpenShift environment, we have created “Deployments” that provide concrete examples of how to provision end user applications to such an OpenShift environment as well as how to provision the necessary

infrastructure components that those applications need. Their purpose, when combined with creating an OpenShift environment, is to provide a relatively straight forward mechanism for setting up an OpenRiskNet VRE.

These deployments follow similar principles to the 'recipes' described earlier, but are more opinionated in terms of the nature of the deployments (e.g. naming conventions) and provide better automation. Currently that automation is in the form of bash scripts, though we may provide further automation in the form of Ansible Playbooks, and these should also be deployable to the OpenShift web console allowing self provisioning of the components in the VRE from a web browser.

These deployments can be found in the GitHub repository and are in a preliminary form. They will change and be extended considerably over the duration of the project. The deployments present at the time of writing are:

- **openrisknet-infra** project containing core infrastructure components that will be required by OpenRiskNet applications (currently a PostgreSQL database and the Keycloak SSO)
- **acme-controller** project providing the ACME controller that can be used to generate and update certificates for HTTPS on publicly exposed routes.
- **jenkins** project providing the Jenkins CI/CD system that replaces the initial stand-alone Jenkins environment described in the D2.1 report.
- Squonk project providing an initial deployment of the Squonk Computational Notebook.

As a simple example that provides an overview of the general approach, the Jenkins deployment consists primarily of:

1. An OpenShift template describing the API objects that need to be created. This is based on one provided by OpenShift, but adapted slightly for OpenRiskNet;
2. An example of how to prepare a persistent volume for use by Jenkins, and how to deploy this (exact details will differ depending on the particular environment to which this is deployed);
3. A bash script for how to deploy an environment / tool from the template;
4. A bash script for how to undeploy that environment / tool if it is no longer needed.

Other deployments follow similar patterns, though some involve more steps.

These deployments have been used to create the current OpenRiskNet reference site described in the next section.

OpenRisknet Reference site

This section describes the current OpenRiskNet reference site.

We label this site as a development site (hence the 'dev' prefix on the domain name). As the project progresses, we will create a more stable production site.

This reference site was created using the standard availability environment on an OpenStack cloud platform that was described in the Environments section and the multiple deployments were described in the Deployments section.

Compute infrastructure

The reference site is running off the Swedish national resource SNIC Science Cloud (SSC) [16] to which OpenRiskNet has been given access through the UU partner. SSC has joined the OpenRiskNet project as a technology partner through the associated partner program. It would also be possible to run the reference site on other cloud platforms like AWS. SSC provides us with access to the following regions (see [17]):

HPC2N

This has limited resources (the 64GB RAM limit is the primary limitation) so we have been able to use this for basic installations, but it is not suitable for larger scale deployments or for high availability environments as we cannot deploy sufficient numbers of servers. However it has sufficient resources for a simple standard availability environment, and the development reference site is currently running in this region.

UPPMAX

This is a new region that is not yet fully accessible. It has significantly more resources, so we plan to use this for experimenting with setting up high availability clusters and for running a production cluster with moderate computational power.

Current Implementation

To summarise the above, the current reference site is:

- In development status, not yet ready for real workloads;
- Running on the HPC2N region Swedish Science Cloud;
- Cloud provisioning is based on OpenStack which is part of the SNIC Science Cloud deployment;
- Standard availability deployed using the recipe described in the Environment section comprising:
    - one master node also running etcd
    - one infrastructure node
    - two worker nodes;
- Nodes are based on the Centos7 operating system with additional packages required by OpenShift installed;
- OpenShift was deployed using the Ansible Advanced installer;
- Current OpenShift version used is 3.6;
- Aggregated logging and metrics have been deployed;
- It comprises the following deployments:
    - PostgresSQL for database storage
    - Keycloak for SSO
    - ACME controller for generation and renewal of TLS certificates
    - Jenkins for running CI/CD pipelines
    - Squonk Computational Notebook as an example end user application.

The current reference site for the OpenRiskNet e-infrastructure can be accessed at:

- Admin console: https://dev.openrisknet.org:8443/ (a login is required)
- End user access: https://home.dev.openrisknet.org/

As this is a development site, is currently not expected to be stable and will likely be destroyed and re-created continually. A more stable production site will become available at a later stage in the project.

Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
worker-2	-	• 192.168.0.5	ssc.xlarge	snic	Active	nova	None	Running	1 week, 1 day	Create Snapshot
worker-1	-	• 192.168.0.15	ssc.xlarge	snic	Active	nova	None	Running	1 week, 1 day	Create Snapshot
infra-1	-	• 192.168.0.8 Floating IPs: • 130.239.81.162	ssc.medium	snic	Active	nova	None	Running	1 week, 1 day	Create Snapshot
master-1	-	• 192.168.0.13 Floating IPs: • 130.239.81.165	ssc.medium	snic	Active	nova	None	Running	1 week, 1 day	Create Snapshot
bastion	-	• 192.168.0.20 Floating IPs: • 130.239.81.208	ssc.small	snic	Active	nova	None	Running	1 week, 1 day	Create Snapshot

Figure 1. OpenStack console showing the current development servers running in the HPC2N region of the Swedish Science Cloud

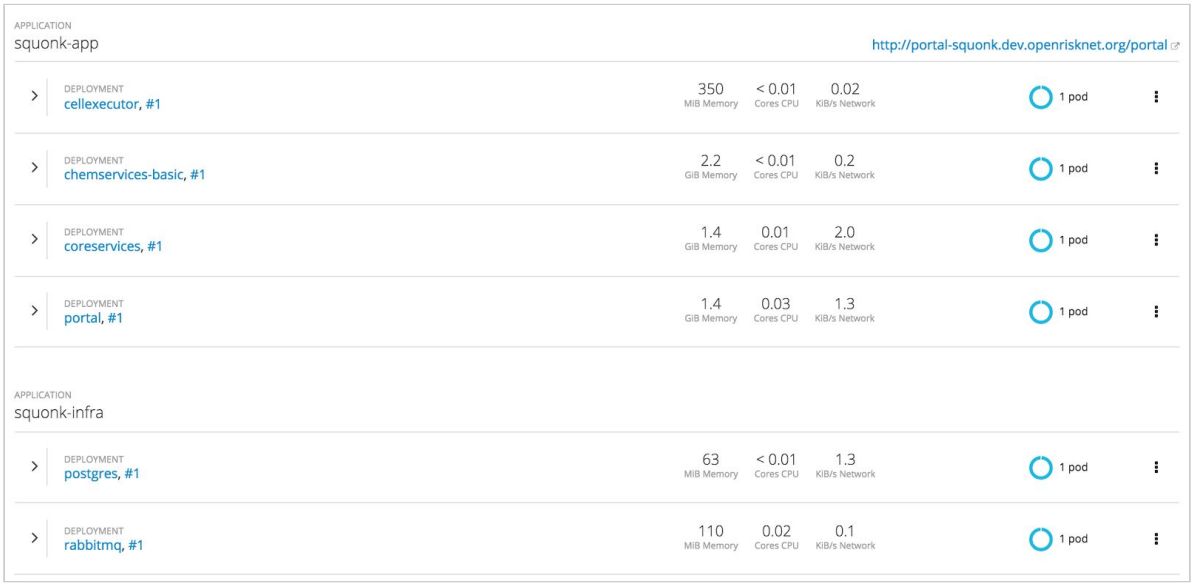


Figure 2. OpenShift console showing the Squonk project that contains the components of the Squonk Computational Notebook

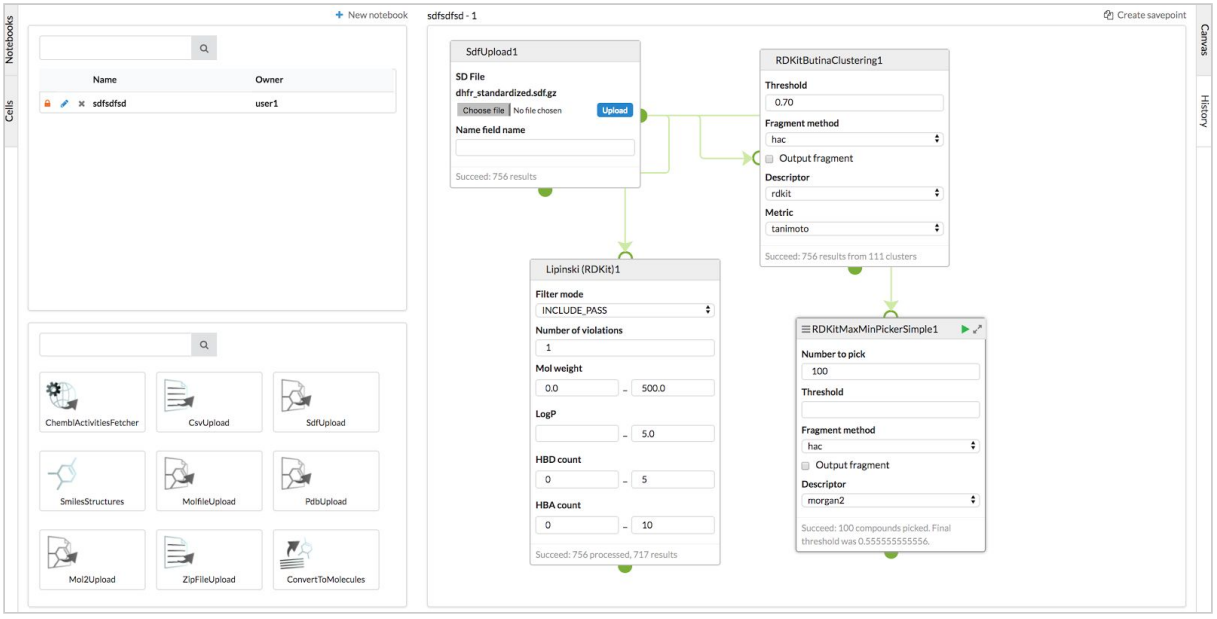


Figure 3. Squonk computational Notebook application running in the OpenRiskNet reference site

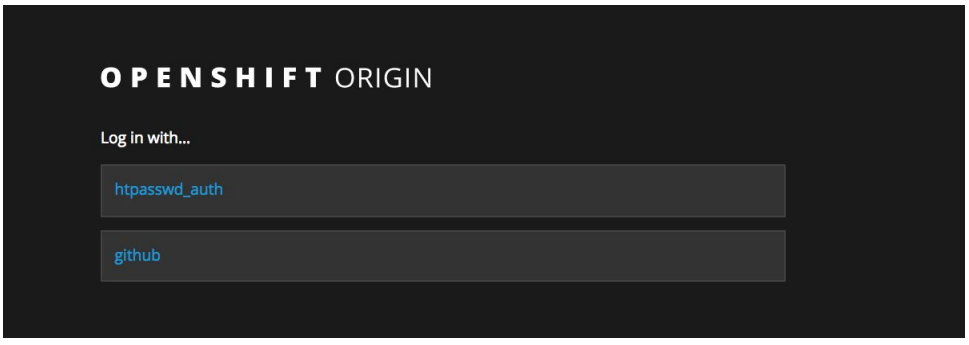
Access control

The current reference site has access control in place.

Two authentication mechanisms are in place:

1. OAuth authentication against GitHub, restricted to members of the 'developers' team in the OpenRiskNet partner organisations. That allows any of the partners who is a member of this team to log in through this mechanism.
2. Additional usernames and passwords specified in a htpasswd file. This is just used for creating special administrator accounts.

When first connecting to the web console, one is prompted to log in using one of these mechanisms as shown below:



End users will not use this mechanism. They will by directed to the Keycloak SSO login. Keycloak provides various ways to manage logins, and it is yet to be decided which we will use. One possibility is to federate out to a social login such as LinkedIn so that OpenRiskNet does not need to manage user passwords. Each VRE administrator can

choose whatever approach is best for their needs. A common option would be to federate to an Active Directory or LDAP directory managed by their organisation.

Future plans

At some stage during the first half of 2018 we expect to be in a position to create a stable production reference site to which we can start to grant access to project and associated partners, and, eventually, third parties (i.e. virtual access Users). This site will have limited high availability capabilities and moderate computation power, sufficient for most of the workflows relevant to the project.

Before we are ready for this, the development site will continue to be used to establish what is needed for a stable production site. This will include:

- Better automation of provisioning of server infrastructure
- Providing high availability capabilities such as multiple masters
- Providing distributed persistent storage (probably though GlusterFS) and cloud native storage
- Providing additional infrastructure components that are required by the end user applications
- Generation and renewal of trusted TLS certificates for all public facing services
- Providing auto-scaling capabilities
- Implementing backup and disaster recovery procedures
- Provision of additional end user applications from partners and third parties
- Use of SSO for all deployed end user applications.

We anticipate having this ready for internal use by the middle of 2018 (Month 18 of the 36 month project) and ready for external use by the end of 2018.

Over this period the various partner and third party applications listed in the “Planned applications” section will be adapted so that they can be deployed via the OpenRiskNet VRE using the approaches described in this document.

Documentation

The documentation is managed in the OpenRiskNet GitHub repository and OpenRiskNet wiki (https://github.com/OpenRiskNet/home/wiki). Currently this is mostly technial information primarily for administrators and developers, and is contained in the openshift directory of the home repository (see https://github.com/OpenRiskNet/home/tree/master/openshift). As we proceed we will also generate end user documentation that we expect will also be stored in GitHub but would be deployed to the reference site and be directly accessible from there as well as from the main OpenRiskNet site.

CONCLUSION

This report describes the initial setup of the development reference site for the OpenRiskNet e-infrastructure, and the accompanying documentation. The report has been written in a manner to provide an overview of the reference site, and as such forms part of the complete set documentation.

Detailed instructions for creating the reference site, or a separate VRE similar to the reference site are contained in the OpenRiskNet GitHub "home" repository (https://github.com/OpenRiskNet/). These documents should provide sufficient information for a reasonably skilled IT administrator to set up such a site. Detailed User guidance notes are also in preparation via the OpenRiskNet Wiki.

The documentation will continue to be enhanced during the project, most notably describing additional types of environment (e.g. High Availability environments) and the deployment of additional applications.

GLOSSARY

The list of terms or abbreviations with the definitions, used in the context of OpenRiskNet project and the e-infrastructure development is available at:

https://github.com/OpenRiskNet/home/wiki/Glossary

REFERENCES

1. OpenRiskNet. OpenRiskNet/home. In: GitHub [Internet]. [cited 1 Dec 2017]. Available: https://github.com/OpenRiskNet/home

2. Docker. In: Docker [Internet]. [cited 1 Dec 2017]. Available: https://www.docker.com/

3. Home | INDIGO DataCloud [Internet]. [cited 1 Dec 2017]. Available: https://www.indigo-datacloud.eu

4. PhenoMeNal – Large-scale Computing for Medical Metabolomics [Internet]. [cited 1 Dec 2017]. Available: http://phenomenal-h2020.eu

5. Kubernetes vs. Docker Swarm | Platform9. In: Platform9 [Internet]. [cited 1 Dec 2017]. Available: https://platform9.com/blog/kubernetes-docker-swarm-compared/

6. OpenShift: Container Application Platform by Red Hat, Built on Docker and Kubernetes [Internet]. [cited 1 Dec 2017]. Available: https://www.openshift.com/

7. Overview - Core Concepts | Architecture | OpenShift Origin Latest [Internet]. [cited 1 Dec 2017]. Available: https://docs.openshift.org/latest/architecture/core_concepts/index.html#architecture-core-concepts-index

8. Overview - Configuring Persistent Storage | Installation and Configuration | OpenShift Origin Latest [Internet]. [cited 1 Dec 2017]. Available: https://docs.openshift.org/latest/install_config/persistent_storage/index.html

9. Keycloak Team. Keycloak [Internet]. [cited 1 Dec 2017]. Available: http://www.keycloak.org/

10. Product Documentation for Red Hat Single Sign-On - Red Hat Customer Portal [Internet]. [cited 1 Dec 2017]. Available: https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/

11. Aggregating Container Logs | Installation and Configuration | OpenShift Origin Latest [Internet]. [cited 1 Dec 2017]. Available: https://docs.openshift.org/latest/install_config/aggregate_logging.html

12. Enabling Cluster Metrics | Installation and Configuration | OpenShift Origin Latest [Internet]. [cited 1 Dec 2017]. Available: https://docs.openshift.org/latest/install_config/cluster_metrics.html

13. Let’s Encrypt - Free SSL/TLS Certificates [Internet]. [cited 1 Dec 2017]. Available: https://letsencrypt.org/

14. Jenkins. In: Jenkins [Internet]. [cited 1 Dec 2017]. Available:

https://jenkins.io/index.html

15. Jaqpot Quattro Documentation [Internet]. [cited 1 Dec 2017]. Available: http://app.jaqpot.org:8080/jaqpot/swagger/

16. SNIC Science Cloud. In: SNIC Science Cloud [Internet]. [cited 1 Dec 2017]. Available: https://cloud.snic.se/

17. System status. In: SNIC Science Cloud [Internet]. 16 Mar 2017 [cited 1 Dec 2017]. Available: https://cloud.snic.se/index.php/status/

ANNEXES

Annex 1. API Objects and templates

These API objects are typically defined in files in JSON or YAML format. A simple example of a definition of a Pod is:

```
apiVersion: v1
kind: Pod
metadata:
  name: postgresql-1
spec:
  containers:
    - env:
        - name: POSTGRESQL_ADMIN_USER
          value: secret
        - name: POSTGRESQL_USER
          value: user123
        - name: POSTGRESQL_PASSWORD
          value: secret
        - name: POSTGRESQL_DATABASE
          value: mydatabase
      image: centos/postgresql-95-centos7:9.5
      name: postgresql
      ports:
        - containerPort: 5432
          protocol: TCP
      volumeMounts:
        - mountPath: /var/lib/pgsql/data
          name: postgresql-pvol
  volumes:
    - name: postgresql-pvol
      emptyDir: {}
```

pod.yaml: example definition of a Pod API object

These definitions can be introduced into an OpenShift environment by several means, most commonly using the command line interface (CLI). For instance the above definition can be created as follows:

```
oc create -f pod.yaml
```

The result would be that pod running the PostgreSQL database within the system.

Whilst this works for a simple case, it does not provide much flexibility (notice how usernames and passwords are hard coded into the pod definition). To address this, Kubernetes and OpenShift provide a mechanism of templates that allow the user to customise parameters and to easily group together the multiple API object definitions that are usually required to deploy a complete application. For instance the above Pod example can better be defined as a template:

```
kind: Template
apiVersion: v1
metadata:
  name: postgresql
```

```
labels:
  template: postgresql-template

objects:

- apiVersion: v1
  kind: Pod
  metadata:
    name: postgresql-1
  spec:
    containers:
    - env:
      - name: POSTGRESQL_ADMIN_PASSWORD
        value: ${POSTGRESQL_ADMIN_PASSWORD}
      - name: POSTGRESQL_USER
        value: ${POSTGRESQL_USER}
      - name: POSTGRESQL_PASSWORD
        value: ${POSTGRESQL_PASSWORD}
      - name: POSTGRESQL_DATABASE
        value: ${POSTGRESQL_DATABASE}
      image: centos/postgresql-95-centos7:9.5
      name: postgresql
      ports:
      - containerPort: 5432
        protocol: TCP
      volumeMounts:
      - mountPath: /var/lib/pgsql/data
        name: postgresql-pvol
    volumes:
    - name: postgresql-pvol
      emptyDir: {}

parameters:
- displayName: Database Admin Password
  description: Database postgres user password
  name: POSTGRESQL_ADMIN_PASSWORD
  from: "[a-zA-Z0-9]{8}"
  generate: expression
- displayName: Database Username
  description: Database username
  name: POSTGRESQL_USER
  value: user123
  required: true
- displayName: Database User Password
  description: Database user password
  name: POSTGRESQL_PASSWORD
  from: "[a-zA-Z0-9]{8}"
  generate: expression
- displayName: Database name
  description: Database name
  name: POSTGRESQL_DATABASE
  value: mydatabase
  required: true
```

pod-template.yaml: example of specifying a Pod definition using a template

In this example the password for the database is not hard-coded into the pod definition but is parameterised so that it can either be defined by the person deploying the application or can be randomly generated (note: this is not a complete example as for instance you would need to notify the deployer what password was generated, though

there are mechanisms to do this).

To deploy this template you would process the template to generate the actual API object definition e.g. using:

```
oc process -f pod-template.yaml > pod-definition.json
```

and then create the API objects e.g. using:

```
oc create -f pod-definition.json
```

or do both as one step e.g. using:

```
oc process -f pod-template.yaml | oc create -f -
```

NOTE: this example is for illustrative purposes only. A real world example would be more complex.