Tailored IoT & BigData Sandboxes and Testbeds for Smart, Autonomous and Personalized Services in the European Finance and Insurance Services Ecosystem

Infinitech

D6.10 – Sandboxes for FinTech and InsuranceTech Innovators - I

Revision Number	3.0
Task Reference	T6.5
Lead Beneficiary	ENG
Responsible	Domenico Messina - Susanna Bonura
Partners	Participating partners in Task according to DOA
Deliverable Type	Report (R)
Dissemination Level	Public (PU)
Due Date	2021-02-28
Delivered Date	2021-03-03
Internal Reviewers	GRA,FBK
Quality Assurance	CCA
Acceptance	WP Leader Accepted and Coordinator Accepted
EC Project Officer	Pierre-Paul Sondag
Programme	HORIZON 2020 - ICT-11-2018
	This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no 856632

Contributing Partners

Partner Acronym	Role[1]	Author(s)[2]
ENG	Lead Beneficiary	Domenico Messina – Susanna Bonura
HPE	Contributor	
NOVA	Contributor	
JRC	Contributor	
ATOS	Contributor	
SILO	Contributor	
WEA	Contributor	
GEN	Contributor	
GFT	Contributor	
GRA	Internal Reviewer	
FBK	Internal Reviewer	
CCA	Quality Assurance	

[1] Lead Beneficiary, Contributor, Internal Reviewer, Quality Assurance

[2] Can be left void

Revision History

Version	Date	Partner(s)	Description
0.1	2020-10-12	ENG and contrib. partners	Table of Contents
0.2	2020-10-26	ENG, NOVA, HPE	Contributions to section 2
0.3	2020-11-30	JRC, ATOS, SILO, WEA, GEN	Contributions to section 3
0.4	2020-12-18	ENG,HPE,GFT	Updated contributions to section 3
0.5	2021-01-15	ENG,HPE,GFT	Initial contribution to section 4
0.4	2021-02-01	ENG and contrib. partners	Updated contribution to section 4
0.5	2021-02-05	ENG and contrib. partners	Updated contributions to section 2
0.6	2021-02-12	ENG and contrib. partners	Updated contributions to section 3
0.7	2021-02-15	ENG and contrib. partners	Updated contributions to section 4
0.8	2021-02-18	ENG,HPE,GFT	Updated ToC
0.9	2021-02-19	ENG	Contribution to section 1
1.0	2021-02-22	ENG	First Version ready for Internal Peer Review
1.1	2021-24-24	HPE	Overall review
2.0	2021-01-01	GRA,FBK,CCA	Version after Internal Peer Review and QA
3.0	2021-03-03	ENG	Version for Submission

Executive Summary

Within the INFINITECH Work Package 6 - Tailored Sandboxes and Testbeds for Experimentation and Validation, this document describes the first of the three expected results of the task T6.5 - FinTech/InsuranceTech Testbed Establishment and Customization.

First of all it provides an overview of a generic on-premise environment, some considerations about how it differs from a cloud-provider environment, and moves its focus onto the specific machines provided by NOVA that will be used for the first approach to testbed tailoring on-premise within the INFINITECH project. A minimal and a recommended set of requirements are provided in order to get a “just fine” environment versus a “high availability” and recommended environment, documenting how to initialize and configure them to get started with a complete INFINITECH development experience.

Then, this document presents the FinTech/InsuranceTech pilots and their sandbox tailoring, but not as an end in itself. The per-pilot process is documented as a guideline to be adopted by all future cases, and it consists of three steps: *(i)* the definition of the objective and the overview of the pilot that is a piece of information that provides the main idea behind the pilot at a glance; *(ii)* the first stage component deployment which contains information about how the software and its dependencies (I.e. framework used, 3rd party software, etc...) are currently built and deployed, their requirements and compatibilities, their condition in the network (I.e. the ports that a certain service listens to); *(iii)* the TO-BE strategy mapping which consumes all the information provided in the previous steps, combines them with the inputs coming from the deliverable, and produces the objects and the software configuration files, described in the document in an easily readable diagram form, for the target environment (I.e. Kubernetes objects and templates). This bundle of artifacts will generate the sandboxes that will be ready to be used during the whole development process. Finally, the results of the work done in the deliverable and the next steps foreseen for the related tasks are summarized.

Table of Contents

List of Figures

List of Tables

Abbreviations/Acronyms

Abbreviation	Definition
AgI	Agricultural Insurance
AMI	Amazon Machine Image
API	Application Programming Interface
AWS	Amazon Web Services
AWS EBS	Amazon Web Services Elastic Block Store
AWS EKS	Amazon Web Services Elastic Kubernetes Service
AWS ELB	Amazon Web Services Elastic Load Balancer
AWS KMS	Key Management Service
BP	Blueprint
CICD	Continuous Integration Continuous Development
CLI	Command Line Interface
CNCF	Cloud Native Computing Foundation
CNI	Container Network Interface
DNS	Dynamic Name Resolution
ENI	Elastic network interfaces
EKS	Elastic Kubernetes Service
EO	Earth Observation
GKS	Google Kubernetes Engine
HA	High Availability
HCL	Hashi Corp Configuration Language
HTAP	Hybrid Transactional and Analytical Processing
IAM	Identity and Access Management
IP	Internet Protocol
K3S	Lightweight Kubernetes
K8S	Kubernetes
ML/DL	Machine Learning Deep Learning
PoC	Proof of Concept
PV	Persistent Volume

PVC	Persistent Volume Claim
RBAC	Role-Based Access Control
RKE	Rancher Kubernetes Engine
SHARP	Smart, Holistic, Autonomy, Personalized and Regulatory Compliance
SSH	Secure Socket Shell
YAML	YAML Ain't Markup Language
VaR	Value-at-Risk
VM	Virtual Machine
VPC	Virtual Private Cloud

1 Introduction

A key innovation of the INFINITECH project is the implementation of the means for the provisioning and configuration if tailored sandboxes over the project's testbeds, which will comprise specific data sources, ML/DL algorithms, APIs and regulatory compliance algorithms. The INFINITECH sandboxes/testbeds aim to facilitate innovators in their efforts to produce BigData/IoT applications that disrupt the sector based on their SHARP properties.

Within the WP6 - Tailored Sandboxes and Testbeds for Experimentation and Validation, task T6.5 aims at establishing and making available one testbed to FinTech/InsuranceTech enterprises of the consortium for their pilots.

This document, D6.10 - Sandboxes for FinTech and InsuranceTech Innovators, is the first result of T6.5 and represents the reference guide of how to set up a testbed on-premise for hosting the software of the FinTech and InsuranceTech firms involved in the INFINITECH project. The on-premise testbed will run the cloud-native artifacts developed according to the INFINITECH software development process (please see D6.4 for more details) and it will be configured in order to match the ease of management of the equivalent cloud provider counterpart as much as possible.

"Cloud Nativeness" is, by definition, the purpose of building software and services that are capable of running in cloud environments by design. A cloud-native Application differs from its legacy equivalent not only from the fact that it runs within a sandboxed execution environment that exploits a containerization technology, but the whole development process is tailored to produce an artifact with such characteristics. There are many other benefits to preferring this kind of approach over the production of a legacy application: the horizontal scaling capability, the adoption of a microservice architecture, the support of the deployment strategy that best suits the needs.

Within this context, there's a very well-known challenge: what if one has a cloud native application and wants to deploy it in an on-premise environment instead of in a cloud provider's ecosystem? Even if there are no changes under the intrinsic configuration of the application, there can be a lot of differences in terms of networking and persistence migrating from the cloud to the premise. There may be a lack of a lot of services offered by the providers "out of the box", such as the load balancer, the automatic management of DNS records, a flexible persistence provisioner, etc.

1.1 Objective of the Deliverable

The testbed tailoring mechanism is an important solution that supports the sandboxed software development process under the umbrella of the INFINITECH project, from now the "INFINITECH way".

The objective of this deliverable is to document the effectiveness of this testbed tailoring mechanism as well as its suitability for the on-premise environments taking into account the needs of the FinTech and InsuranceTech presences within the INFINITECH consortium. This document is not just a per pilot report of sandbox construction per se, but it's more a reference guide for all those individuals who want to participate and exploit the ecosystem of INFINITECH to be ready to develop, build and test new artifacts for the platform into dedicated execution environments. The NOVA infrastructure will be the host of the first attempt at an on-premise environment federation to the INFINITECH approach. This infrastructure will run five sandboxes, one for each participant pilot. Having these five independent (but not isolated) workflows will be helpful to analyze which are the most common frameworks and dependencies the developers need to have in place to complete all the integration tests of the novel artifacts and to allow them to run correctly as expected.

1.2 Insights from other Tasks and Deliverables

WP6 has been contributing to different WPs and deliverables and in turn relies on inputs coming from other WPs, as shown in the following figure.

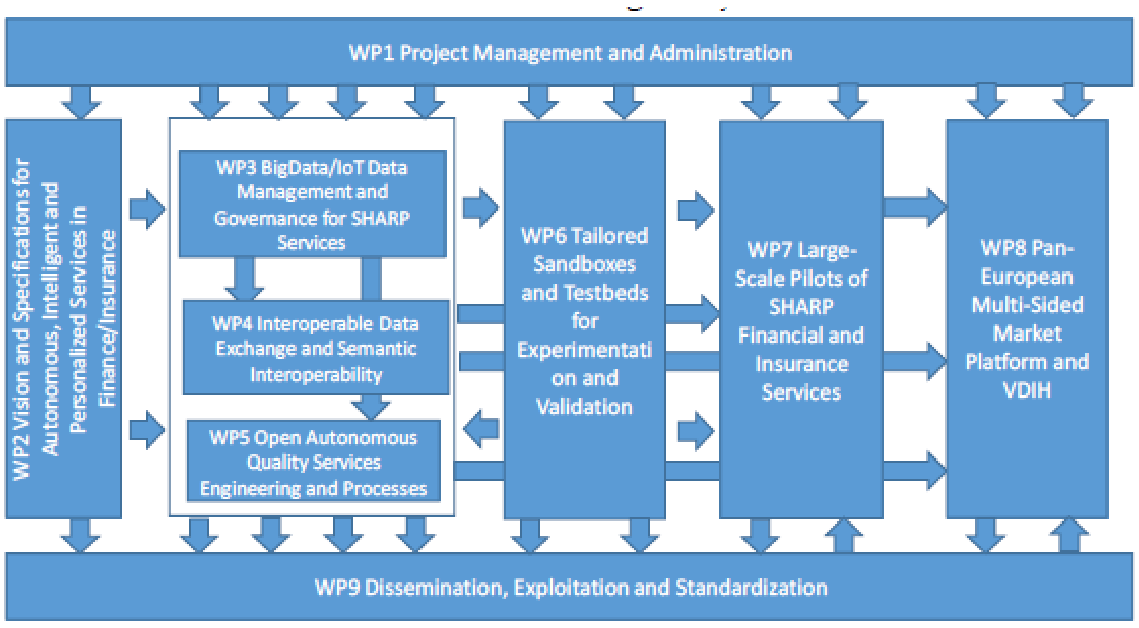
WP1 Project Management and Administration
WP2 Vision and Specifications for Autonomous, Intelligent and Personalized Services in Finance/Insurance
WP3 BigData/IoT Data Management and Governance for SHARP Services
WP4 Interoperable Data Exchange and Semantic Interoperability
WP5 Open Autonomous Quality Services Engineering and Processes
WP6 Tailored Sandboxes and Testbeds for Experimentation and Validation
WP7 Large-Scale Pilots of SHARP Financial and Insurance Services
WP8 Pan-European Multi-Sided Market Platform and VDIH
WP9 Dissemination, Exploitation and Standardization

Figure 1 - INFINITECH Work Breakdown Structure

Within T6.5 the deliverables submitted until now from WPs3-4-5 have been taken into account.

In addition, the following deliverables within the WP6 are key inputs to this document:

- D6.1 - Testbeds Status and Upgrades, which contains the initial analysis for the current status of the existing partners' infrastructure (hardware & software) that will be used as the basis for all the testbeds to host the Pilots of the INFINITECH Project.
- D6.4 - Tools and Techniques for Tailored Sandboxes and Management of Datasets – I, which describes the preliminary results of INFINITECH WP6 Tasks T6.2 "Mechanisms and Tools for Tailored Sandboxes Provision and Configuration" and T6.3 "Integrated Management of Testbeds' Datasets".

It is worth noticing that there is a relation between this deliverable and the deliverable on the Sandboxes in Incumbent Testbeds (D6.7) since they share the same goals but from different perspectives.

Finally, the progress of task T6.5 (together with the other tasks in WP6) is a key driver of the INFINITECH work package WP7, which is focused on the Large Pilots Operations and Stakeholders Evaluation of the proposed Financial and Insurance Services.

1.3 Structure

In addition to a first and introductory section that provides the context and explains the objectives of the deliverable, this document consists of the following sections:

- Section 2 provides an overview of how to initialize and configure the NOVA testbed to get started with the complete INFINITECH testbed establishment.
- Section 3 is related to the FinTech/InsuranceTech pilots and their sandbox tailoring. The pilots to be hosted within the NOVA testbed are pilots 2, 11, 12, 13, 14.
- Section 4 summarizes the results of the work done in the deliverable and the next steps foreseen for the related tasks.
- Section "Appendix A: Literature" provides details about all the cited work.

2 INFINITECH System Design for shared testbed

2.1 Shared testbed overview

Deliverable "D6.4 – Tools and Techniques for Tailored Sandboxes and Management of Datasets - I" gives a detailed description of the scenarios regarding all of the envisioned INFINITECH testbeds. In this section, we recall the design of the shared Data Center hosted by the partner NOVA, which will be established and provisioned in order to support the experimentation of the five (at the time of writing) FinTech and InsuranceTech innovators pilots within the consortium.

We remind readers that in INFINITECH the set of hardware resources in a data center (like storage, computing resources and network) will be considered as a testbed, as shown in the following picture:

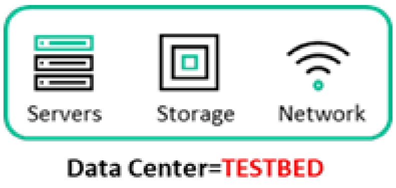

Figure 2 – Parallelism between Testbed and Data Center

In general, we envision that each INFINITECH pilot will have one or more Use Cases (composed of one or more pilot Applications that, individually, should be composed of one or more INFINITECH microservices). In our vision, each Use Case will be a Sandbox provisioned by the leverage of Kubernetes Namespaces [1].

In Kubernetes (K8S) we use Namespaces to obtain separation between Kubernetes objects. They are a logical grouping of a set of Kubernetes objects to whom it's possible to apply some policies, in particular:

- Quote sets the limits on how many hardware resources can be consumed by all the objects.
- Network defines if the namespace can be accessed or can get access to other Namespaces, in other words, if the Namespace is isolated or accessible.

Network segregation is obtained via network policies: such policies are enforced by Cilium CNI (Container Network Interface) configurations, and the only intra-cluster communication allowed will be through the following namespaces: "devops" to enable application deployments and "kube-system" for cluster maintenance [2].

On top of Cilium, a networking and security observability platform called Hubble [3] is available and its main goal is to support network troubleshooting and monitoring.

In the Shared Testbed provided within the NOVA data center, one Kubernetes cluster for each pilot will be provided, as shown in the following picture:

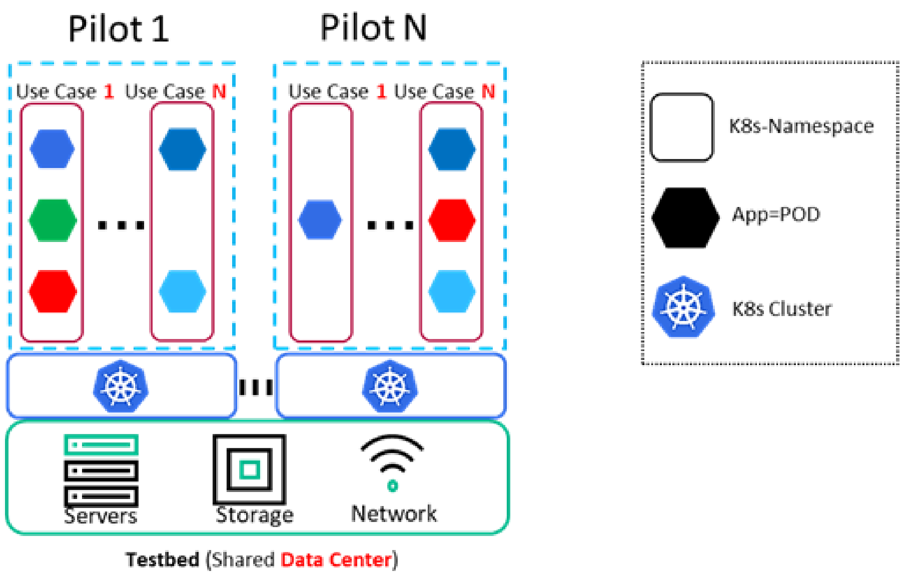

Figure 3 - Shared testbed configuration

2.2 Minimal requirements and recommended requirements

As mentioned in the previous section, INFINITECH testbeds will rely on Kubernetes to orchestrate containers and to manage the resources of the nodes conveniently. According to the number of nodes available and the amount of dedicated resources, a Kubernetes cluster can be initialized using a low-end configuration (but still conforming to the kubernetes best practices) or as a production-ready configuration with a high available topology.

In the Kubernetes ecosystem, there are two types of nodes: worker node and control plane node. Worker nodes are the nodes that run all the user workload pods, while control-plane nodes are service nodes that manage the whole cluster in terms of scheduling, API access, health, and so on. By default, the cluster doesn't schedule user pods on the control-plane node for security reasons, therefore, if the administrator has to initialize a Kubernetes cluster with a single-node (i.e., a development machine), this workflow must explicitly configure the node to accept workload. For the low-end testbed, at least the following requirements are needed:

- One or more machines running a deb/rpm-compatible Linux OS; for example, Ubuntu or CentOS.
- 2 GB or more RAM per machine.
- At least 2 CPUs on the machine used as a control-plane node.
- Full network connectivity among all machines in the cluster. A public or a private network can be used.

For a High Availability cluster, which is recommended for production and fault-tolerant environments, several nodes can be configured in order to have a proper set of control-plane nodes and worker nodes which would guarantee a high level of resilience whenever accidents occur. Further details are available in the official Kubernetes documentation at https://kubernetes.io/docs/setup/.

In the specific case of the NOVA infrastructure, three nodes will be available and all three nodes will have the control-plane and worker roles within the cluster with the following characteristics:

- CPU: 4x Intel Xeon-G 6238R (Total: 112 Cores / 224 vCores)
- Memory: 1 TB

- GPU: 1x NVIDIA Tesla M10 Quad GPU (2560 CUDA Cores)
- Storage: (6 to 12 TB) x 3

2.3 Shared testbed setup

The purpose of this section is to describe the process to recreate the Blueprint (BP) testbed for a specific INFINITECH Pilot on a bare-metal environment hosted by the NOVA infrastructure in terms of all software layers depicted in Figure 3 - Shared testbed. Such a process should be automated as much as possible, but full automation is complicated as before replicating a K8S cluster from Amazon Web Services (AWS) to a bare-metal environment, we need to install and configure all software layers beneath K8S. In the rest of the section we will explore a possible way to automate this process.

Let's focus our attention on a single on-premise Kubernetes Cluster. The first assets we need are the Linux systems which are the baseline of where Kubernetes will be installed. Each Linux system will be a VMware [4] Virtual Machine located on a VMware Cluster: in order to simplify as much as possible, we assume starting with a certain number of vSphere ESXi [5] ready to install our virtual machines.

Before going more into detail about the process, we want to recall some concepts about vSphere. The following picture depicts the concept of vSphere VM template:

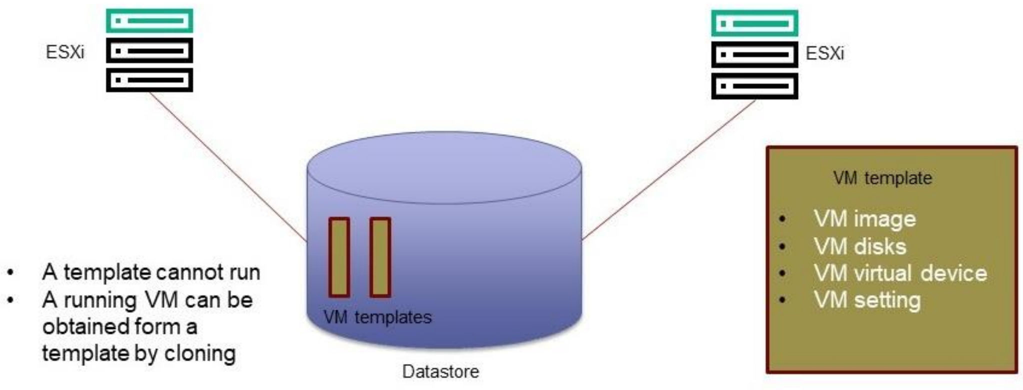

Figure 4 - vSphere VM Template

A VM template on vSphere is like a "package", and it is composed by the *VM image*, the *VM disks*, the *Virtual Devices* and all the *VM settings*. A vSphere VM template can be cloned to create a new virtual machine instance as shown in the following picture:

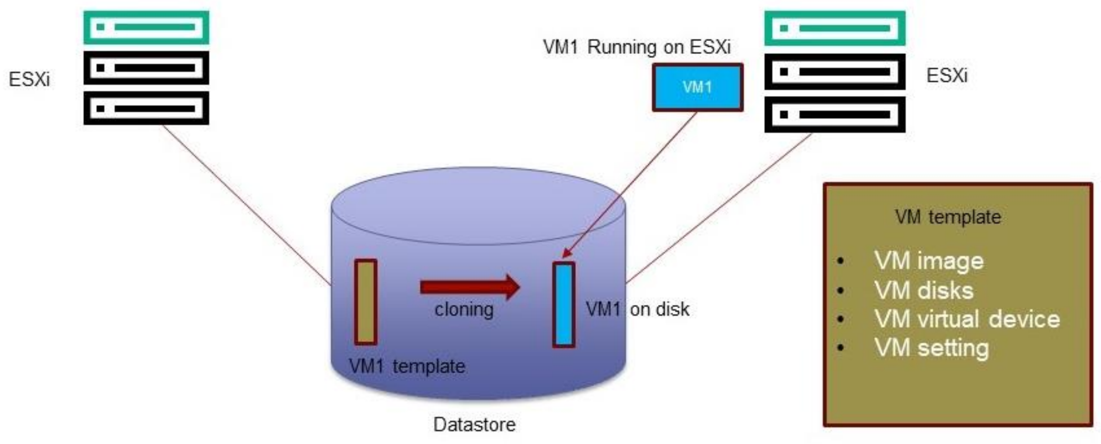

Figure 5 - Cloning a VM Template

The idea is to have a certain number of VM templates in the datastore (the virtual storage on VMware) and to clone them when it is necessary to have a running Virtual Machine.

The intention is to have a VM running in two phases:

1. VM Template creation: VM templates are stored in the datastore (the virtual storage on VMware).
2. VM template cloning to obtain a running virtual machine.

The problem now is to find a way to automate these two phases.
Phase one will be automated using a tool called *Vagrant* [6], while phase two will be automated using a tool called *Terraform* [8]. The process is depicted in the next picture.

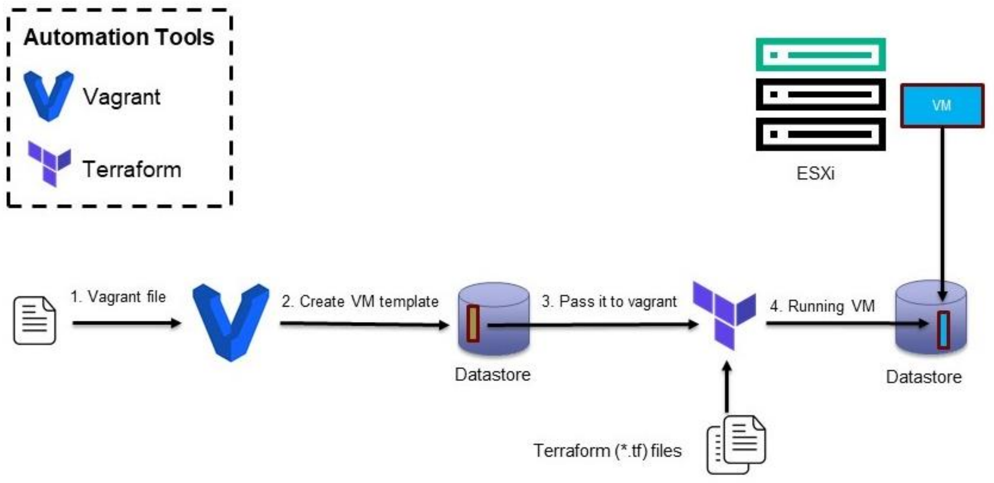

Figure 6 - Automation tools Flow

To fully understand the features of Vagrant and Terraform it is possible to access the product website (see [6]and [8]. A brief overview of these tools follows.

Vagrant [6] is an automation tool for creating and managing virtual machine environments in a single workflow. It can work with different kinds of hypervisors like Oracle Virtual Box [9], VMware workstation [10], vSphere [5] and others. It can manage the whole lifecycle of the virtual machines, without interacting directly with the specific hypervisor. The Vagrant files are where the instructions on how to build a specific virtual machine are located. The syntax of a Vagrant file is Ruby [11] but usually is not necessary to have any programming language knowledge to configure Vagrant.

Vagrant can work with VMware vSphere and is able to automate the creation of the vSphere VM template, so it solves the first deployment phase we have previously described.

Terraform [8] is an open-source infrastructure and it works as a code software tool that provides a consistent CLI workflow to manage hundreds of cloud services. Terraform codifies cloud APIs into declarative configuration files. It is a tool for infrastructure provisioning capable of building an infrastructure setup as code.

It works with many cloud providers: AWS, Google, Azure, and others. In the case of AWS for example it can build any EC2 [12] instance from an Amazon Machine Image (AMI) available in the marketplace and can configure the software on the instance using custom data. For Terraform running and building we need a resource file (.tf) [8] that describes all the infrastructure objects we want to build on a specific provider. The language used in this file is called Hashi Corp Configuration Language (HCL) [13] and is the "code" that describes the architecture.

Another interesting feature of Terraform is the concept of "Plan, Apply and Destroy": when the resource file (.tf) is ready it is possible to run a preview through the command `terraform plan`, and Terraform will provide some insights into what this specific ".tf" is going to build, such as a list of resources and settings; at this point, it is possible to carefully check the plan preview and eventually proceed with the build running the command `terraform apply` that starts the infrastructure build process. Finally, the command `terraform destroy` will delete the infrastructure.

In addition to working with a Cloud Provider, Terraform is also compatible with vSphere VMware products like ESXi and vCenter. In particular, it can deploy Virtual Machines starting from VM templates, so it solves the second phase automation problem previously discussed. To better understand what Terraform files look like, we attach two files necessary to clone a VM on vSphere (the complete example could be found at [14]).

```
#cat terraform.tfvars
vsphere_user = "terraform_user@vsphere.local"
vsphere_password = "SuperSecretPassword"
vsphere_server = "192.168.100.50"
vsphere_datacenter = "datacenter"
vsphere_datastore = "datastore-1"
vsphere_resource_pool = "cluster/Resources"
vsphere_network = "VM Network"
vsphere_virtual_machine_template = "centos_7_template"
vsphere_virtual_machine_name = "terraform-vsphere-clone-test"
----
#cat main.tf
provider "vsphere" {
  user           = "${var.vsphere_user}"
  password       = "${var.vsphere_password}"
  vsphere_server = "${var.vsphere_server}"
  allow_unverified_ssl = true
}

data "vsphere_datacenter" "dc" {
  name = "${var.vsphere_datacenter}"
}
data "vsphere_datastore" "datastore" {
  name          = "${var.vsphere_datastore}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
data "vsphere_resource_pool" "pool" {
  name          = "${var.vsphere_resource_pool}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
data "vsphere_network" "network" {
  name          = "${var.vsphere_network}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
data "vsphere_virtual_machine" "template" {
  name          = "${var.vsphere_virtual_machine_template}"
  datacenter_id = "${data.vsphere_datacenter.dc.id}"
}
resource "vsphere_virtual_machine" "cloned_virtual_machine" {
  name             = "${var.vsphere_virtual_machine_name}"
  resource_pool_id = "${data.vsphere_resource_pool.pool.id}"
  datastore_id     = "${data.vsphere_datastore.datastore.id}"

  num_cpus = "${data.vsphere_virtual_machine.template.num_cpus}"
  memory   = "${data.vsphere_virtual_machine.template.memory}"
  guest_id = "${data.vsphere_virtual_machine.template.guest_id}"

  scsi_type = "${data.vsphere_virtual_machine.template.scsi_type}"

  network_interface {
    network_id   = "${data.vsphere_network.network.id}"
    adapter_type = "${data.vsphere_virtual_machine.template.network_interface_types[0]}"
  }

  disk {
    label = "disk0"
    size = "${data.vsphere_virtual_machine.template.disks.0.size}"
  }

  clone {
    template_uuid = "${data.vsphere_virtual_machine.template.id}"
  }
}
```

Finally, we can summarize the flow that we have previously described as:

1. We will use Vagrant to create the VM template on vSphere.
2. We will use Terraform to automate the deployment of VMs based on templates created in the previous step.

To complete the environment setup, we will use Rancher [15] to automate the Kubernetes cluster creation in addition to the previous flow, as shown in the picture below.

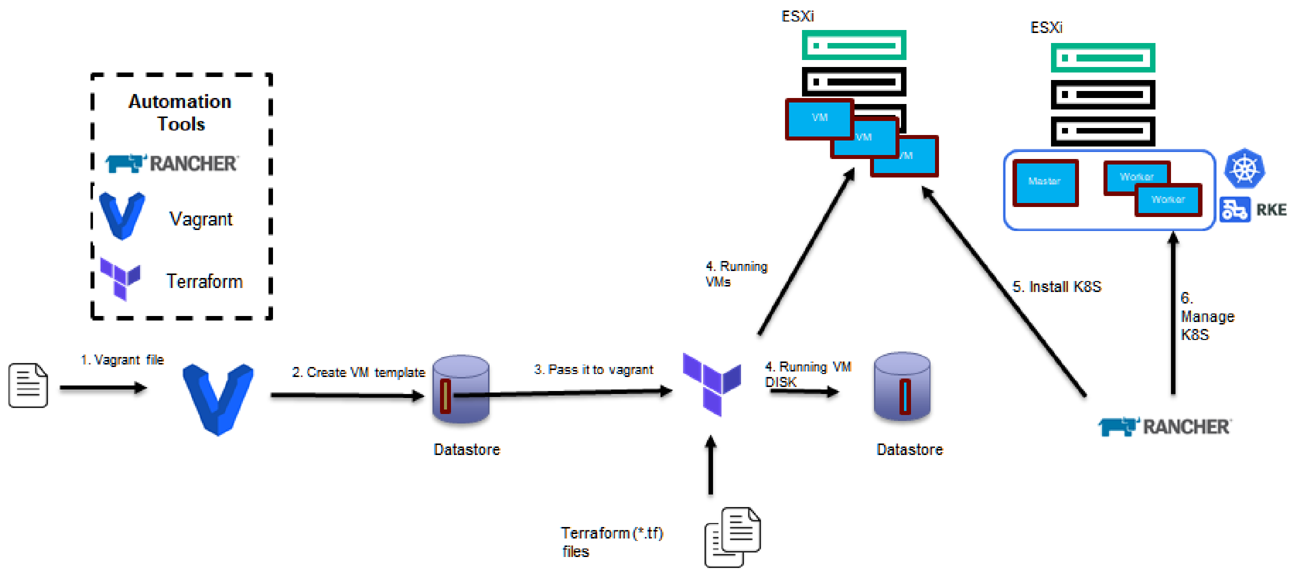

Figure 7 - Automated K8S cluster creation with Rancher

As described in section 2.1, several pilots will be working on a NOVA data center. To ensure the best isolation, each pilot will have its own Kubernetes cluster where each use case will run on a different Kubernetes namespace.

To simplify and optimize the Kubernetes clusters management, the Rancher tools will be used. Rancher is an open-source framework that enables multiple cluster management and provisioning. It supports several Kubernetes distributions both on-premise (RKE, K3S, standard Kubernetes) and as a service (EKS, AKS, and GKE). The Rancher framework acts as a traversal management layer covering the five K8S clusters (related to the five INFINITECH pilots hosted in NOVA) and gives administrators the ability to modify all clusters, and to add/remove nodes, controlling access and adding new clusters if needed. The administrator will be able to replicate clusters with the eventual necessary modifications, to obtain all the pilot clusters while reducing the setup time. Configuration management will be feasible since a cluster definition can be created via web application and configuration file, possibly versioned.

Between all Rancher Kubernetes supported flavors, the better choice for its completeness of support is RKE (Rancher Kubernetes Engine), which is a CNCF (Cloud Native Computing Foundation) [16] certified Kubernetes distribution that leverages Docker container features and SSH to install Kubernetes nodes and which is fully integrated in Rancher. The previous picture shows that all pilot clusters will be RKE K8S managed by a Rancher service, which will also be running on an RKE cluster itself.

Figure 8 below, shows the new layout of Sandboxes in a shared Testbed.

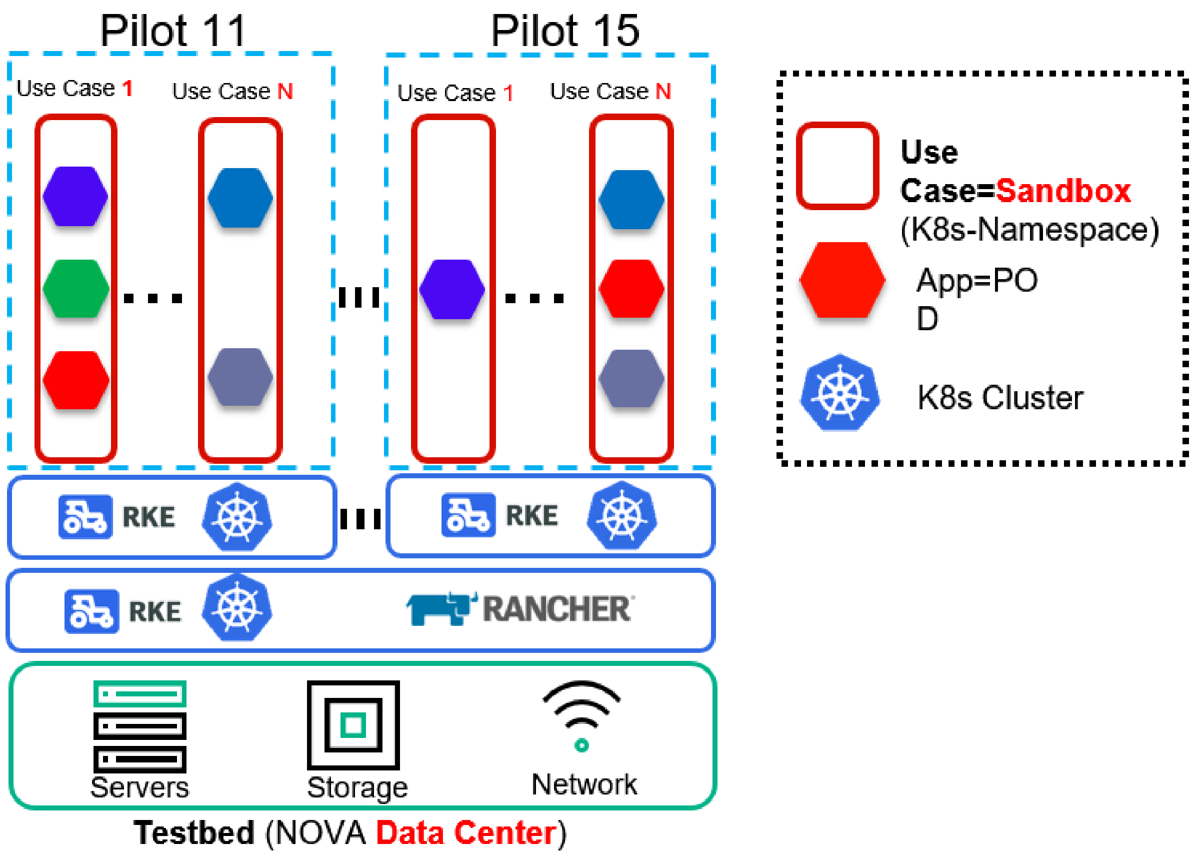

Figure 8 - Layout of Sandboxes in NOVA Testbed

2.3.1 Rancher installation

Rancher can be installed on a single node Kubernetes cluster or on a multiple node cluster to obtain higher availability. Since the configuration and the management tasks of all the clusters are demanded to Rancher and its API server, it should not be prone to the weakness of a single point of failure, therefore it has to be deployed appropriately.

For this reason, Rancher can be also installed in HA[3], if a Kubernetes dedicated cluster has more than two etcd nodes and more than one control-plane node. To obtain better performance and more security, we will dedicate an RKE Kubernetes cluster with the following requisites:

[3] https://rancher.com/docs/rancher/v2.x/en/overview/architecture-recommendations/

- Number of nodes: 3
- Node roles: etcd, control-plane, api and worker each
- Each node hardware requisite[4] [5]:
  - OS: Ubuntu 16.04, 18.04, 20.04, for our installation will be 18.04
  - HW: 2vcpu and 8GB Ram

Rancher implements Role-Based Access Control (RBAC) and supports OpenLDAP [17], so Rancher and cluster users will be managed centrally by BP OpenLDAP.

2.3.2 Cluster Operations

All NOVA clusters will be GKE-based and managed via Rancher, as this enables centralized provisioning and access control.

The picture below shows how a user can manage Rancher deployments of different downstream clusters: it is possible to manage an RKE provisioned cluster (it is the first choice for on-premise clusters in Rancher) or to manage an AWS provisioned cluster (one of the supported hosted services). The picture also shows that Rancher enables users to manage each cluster with kubectl in the same way as on a vanilla K8S but connecting to the Rancher Authentication Proxy. However, if correctly configured, RKE clusters can be accessed also directly via the master node. Rancher Authentication Proxy integrated with open LDAP simplifies user management, concentrating all users' operations on one instance.

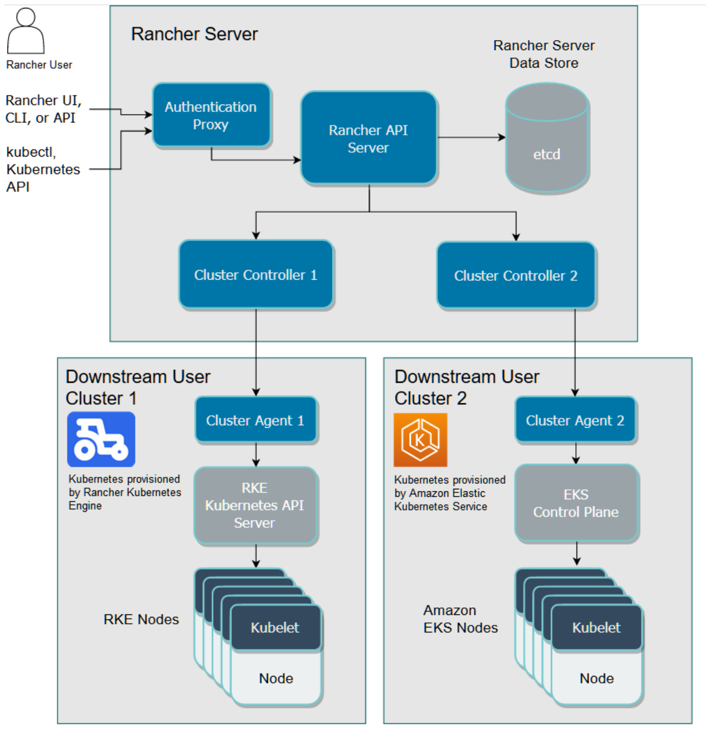

Figure 9 - Rancher operation[6]

[4] https://rancher.com/docs/rancher/v2.x/en/installation/requirements/

[5] https://rancher.com/support-maintenance-terms/

[6] https://rancher.com/docs/rancher/v2.x/en/overview/architecture/

2.3.3 Cluster Configuration

All clusters will be as much as possible adherent to a BP cluster configuration: there will be one namespace for use cases, so multiple namespaces per cluster will be present.

In order to manage public IP, we would like to use MetalLB [18] which will be deployed on Rancher. Its configuration will be defined accordingly with the NOVA lab owner.

To implement the network policies to isolate each namespace from each other, and eventually also to manage the connection between different PODs within the same namespace (i.e., to guarantee the security requirements), we will implement the Cilium [2] as CNI (Container Network Interface) as prescribed by Blueprint.

Cilium will be installed with Helm [19] with some features enabled:

- fully-routable ENI IP address for each pod;
- metrics and Hubble activated: they provide insights into the state of Cilium itself (agent and operator).

```
Cilium policy deployment follows:
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "allow-within-namespace"
specs:
  - endpointSelector:
      matchLabels: {}
    egress:
    - toEndpoints:
      - matchLabels:
          "k8s:io.kubernetes.pod.namespace": devops
    ingress:
    - fromEndpoints:
      - matchLabels:
          "k8s:io.kubernetes.pod.namespace": devops
  - endpointSelector:
      matchLabels: {}
    egress:
    - toEndpoints:
      - matchLabels:
          "k8s:io.kubernetes.pod.namespace": kube-system
          "k8s:k8s-app": kube-dns
  - endpointSelector:
      matchLabels: {}
    egress:
    - toEntities:
      - host
      - remote-node
  - endpointSelector:
      matchLabels: {}
    ingress:
    - toEntities:
      - host
      - remote-node
      - world
  - endpointSelector:
      matchLabels: {}
    egress:
    - toEndpoints:
      - matchLabels:
          "k8s:io.kubernetes.pod.namespace": ingress-nginx
    ingress:
    - fromEndpoints:
      - matchLabels:
          "k8s:io.kubernetes.pod.namespace": ingress-nginx
```

Cluster modification will be controlled via configuration management outside Rancher framework, as it is possible to dedicate a project on INFINITECH GitLab to keep versioning and to deploy new cluster configurations via Rancher UI.

2.4 Configuration management adaptation

As pointed out in the previous section, to simplify access management to all the services (like Rancher [15] console), they will be integrated with BP Open-LDAP [17] service, in order to simplify access management and to control new services, a new ad hoc user group will be created.

One of the key features of BP resides in its Continuous Integration / Continuous Deployment (CI/CD) infrastructure, which enables all processes from source code control to applications deployment on clusters, to be deployed automatically. Such infrastructure will be leveraged also for NOVA Lab, enabling BP CI/CD to deploy on NOVA K8S [1]clusters.

GitLab [20] is already reachable from the Internet, enabling all INFINITECH users to upload their source code from everywhere. The same applies to Harbor [21], which enables users to upload their cooked images without forcing the disclosure of source code and with the capabilities to distribute to any cluster connected to the Internet.

Jenkins [22] has two main tasks: the first one is to automatically build an application image and push the build results on the Harbor repository; the second one is to deploy applications.

Image push will be possible in BP in two ways:

1) building from source code with a dedicated Jenkins task (pipeline) which will automatically pushes images on the Harbor repository, in case of open-source projects;

2) manually loading an image on Harbor, in the case of a project with closed-source software.

The picture below shows a different flow for closed-source projects.

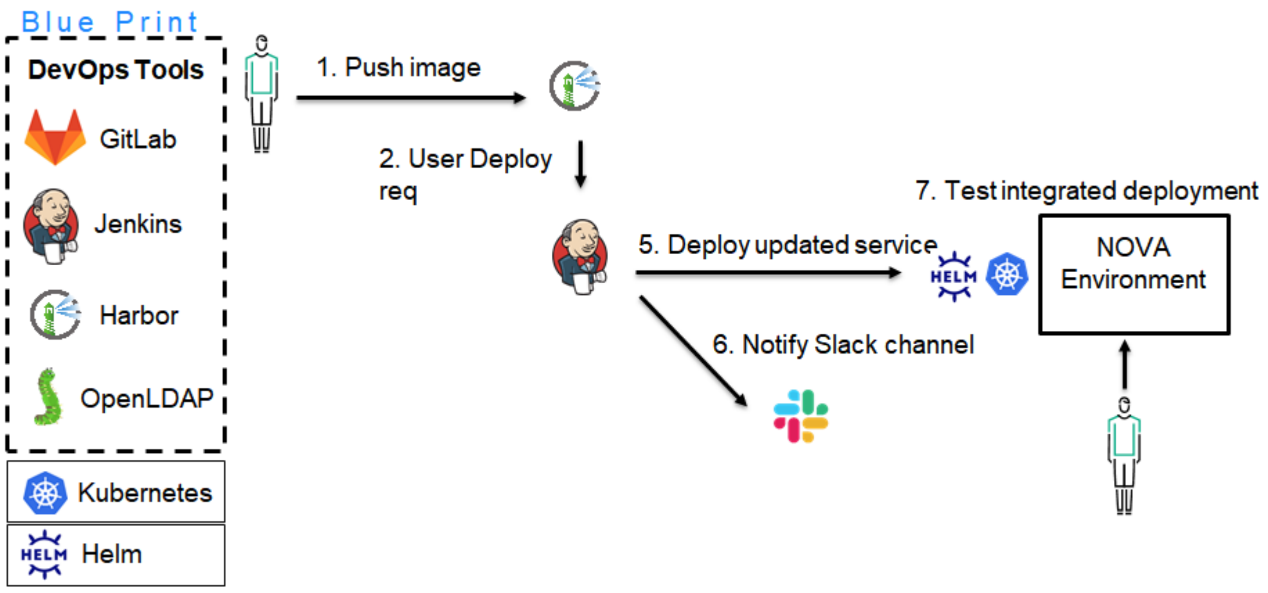

Figure 10 - Different flow for closed-source projects

To deploy images on NOVA clusters, we will explore the possibility to expose the Rancher cluster-management ports to BP Jenkins services, so that it will be possible to distribute configurations and finalize application deployments.

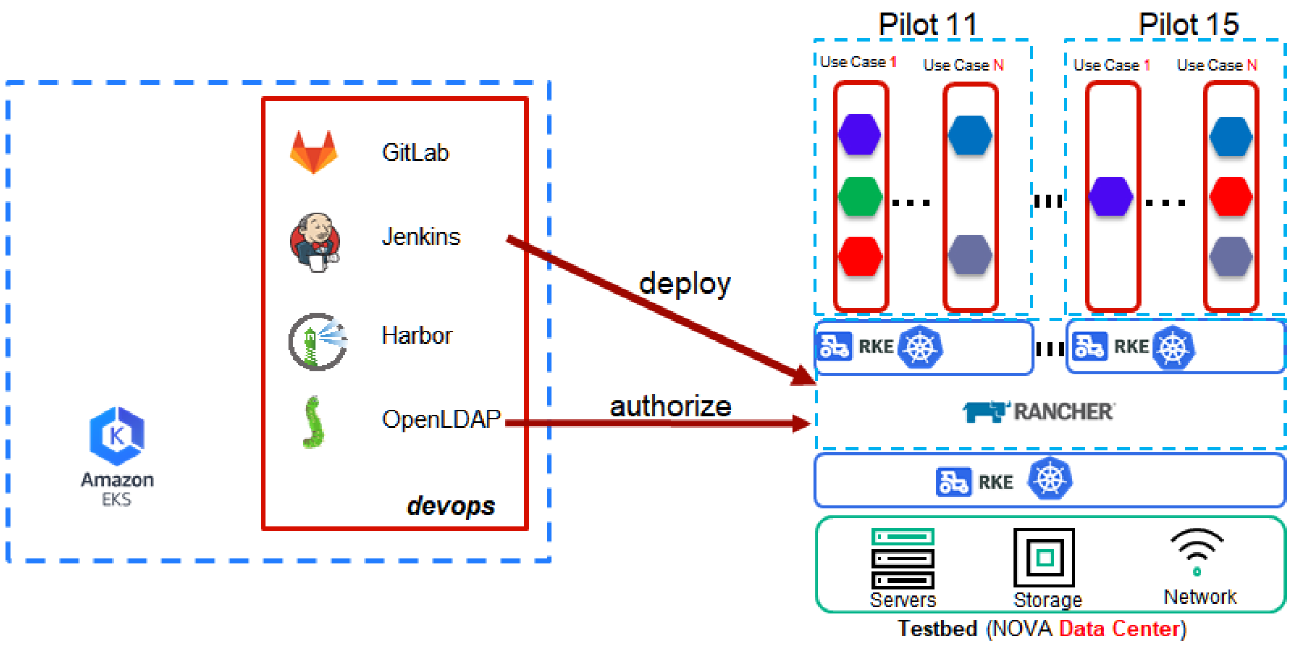

Figure 11 - Deployment Interactions

2.5 Handling hybrid scenarios

One of the biggest challenges of the NOVA testbed establishment is how to deal with hybrid scenarios. Hybrid scenarios occur whenever there are indelible constraints that prevent a module from being downward-mapped in a sandbox within the NOVA premise. Nevertheless, an exceptional component can still match an upward mapping because the function that it fulfills can find a suitable logical placement within the INFINITECH reference architecture.

There are two categories of constraints that have been identified: one is the intrinsic constraint which is related to the fact that a certain component is not containerized using a containerization system compliant with the specific Kubernetes installation in the testbed which, in the specific case is provided by NOVA. The other category is the extrinsic constraint which refers to all the software (containerized and/or legacy) which cannot run within the testbed and must be deployed in a dedicated 3rd party execution environment that often corresponds to the private infrastructure owned by the module's provider.

Whenever these constraints occur, the software can be wrapped and handled using the **Sidecar deployment**, that allows a component to be hosted remotely, regardless of its containerization status (Figure 12).

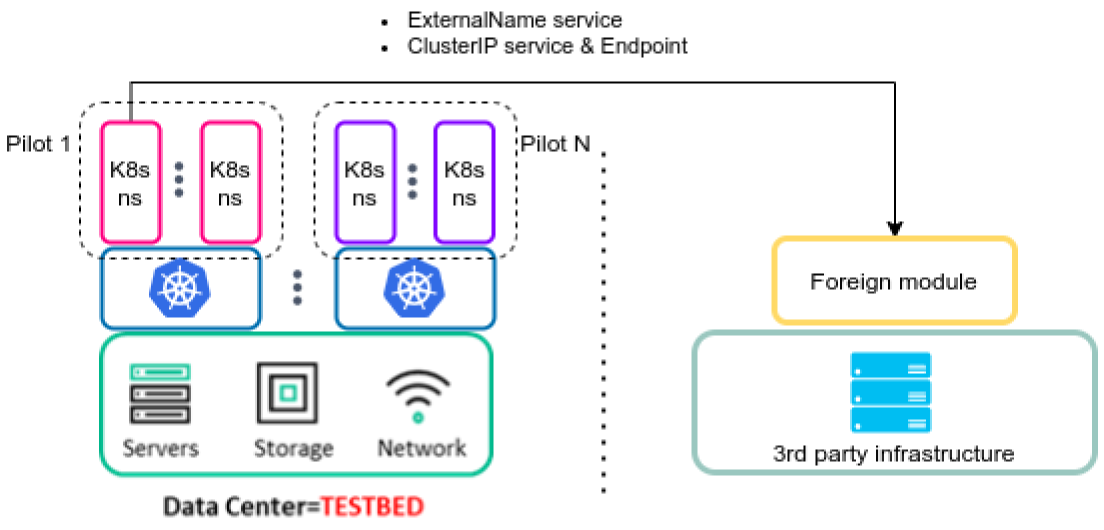

Figure 12 - Sidecar deployment

The hybrid deployment can be managed thanks to the built-in Kubernetes service discovery mechanisms and its networking abstraction layer. In particular, the solution involves the creation of a static Kubernetes service without pod selectors alongside a new Endpoint object that will send the traffic to the target software, according to the Kubernetes documentation [23].

```
kind: Service
apiVersion: v1
metadata:
 name: my-other-module
 namespace: my-pilot
Spec:
 type: ClusterIP
 ports:
 - port: 8081
   targetPort: 8081
```

```
kind: Endpoints
apiVersion: v1
metadata:
 name: my-other-module
 namespace: my-pilot
subsets:
 - addresses:
     - ip: 10.105.25.56
   ports:
     - port: 8081
```

In the case that the administrator of the 3rd party execution environment provides a public hostname to get access to the considered module, the solution is to create a Kubernetes service of type “ExternalName” and allow the in-cluster sandbox to communicate through it.

```
kind: Service
apiVersion: v1
metadata:
 name: my-foreign-module
 namespace: my-pilot
spec:
 type: ExternalName
 externalName: foreign-module.3rdparty.org
```

In general, static services, endpoints and *externalname* services can be combined to overcome such kinds of constraints in an elegant and maintainable manner.

3 Sandboxes for FinTech and InsuranceTech Innovators

3.1 Mapping approach

The overall mapping approach consists of four subsequent steps (see Figure 13):

1. **the first step** is the analysis of the technical specification related to each dependency and subcomponent that the pilots have to use in order to reach their objectives, in other words, how to get a docker image of the components and/or how to get it out from the legacy software in case the component has never been containerized before. The docker image has to be configured at deploy time to be used without any constraint within a Kubernetes pod.
2. **The second step**, namely "upward mapping" consists in the mapping of all the components used and produced by the pilots into the INFINITECH reference architecture main categories (please see D2.13 for further details), so that all the artifacts will be categorized and handled within the context of the INFINITECH project as a whole for an easier fetch in the INFINITECH software catalogue and for an easier solution addressing the use-case problems. As a quick recap, this logical categorization is composed of the following groups: data sources, data ingestion, data management, data security and privacy, blockchain and information sharing, data model and semantics, internet of things, analytics and machine learning, interfaces.
3. **The third step** consists in the production of one or more Kubernetes objects. This step strictly depends on the scope and on the requirements of the modules that have to be mapped. As a rule of thumb, some factors that may condition the deployment mapping are:

   - If two or more components are tightly coupled and match a functionality together, then it's better to have them within the same pod.
   - If the correct execution of a module strictly depends on the hardware characteristics of a particular node it should run onto, for example if software requires the computational capabilities of the graphic card, then the pod has to use the node affinity feature available in Kubernetes.
   - If the module needs storage persistence, and the storage class is not a distributed storage or a network-reachable storage, then it's better to use a statefulset or node affinity.
   - It's always better to externalize any configuration property file using environment variables and/or Kubernetes configmaps in order to avoid any kind of hardcoded tuning of the software.
   - If the usage of a module is recurrent among several pilots, the possibility to have it deployed within a shared namespace in the testbed and linked with an external service has to be considered.

4. The **fourth and final step** is the refinement of the testbed template (helm or Kubernetes *yaml* file) in order to include the "heavyweight shared frameworks" found within the context of the pilots. A report related to the outcome of this activity is described in section 3.7. Note that in the TO-BE strategy and mapping section dedicated to each Pilot in this document, there is a list of all the Kubernetes API Resources used to map each component in the sandbox and the majority of these API Resources belong to the default Kubernetes API Group.

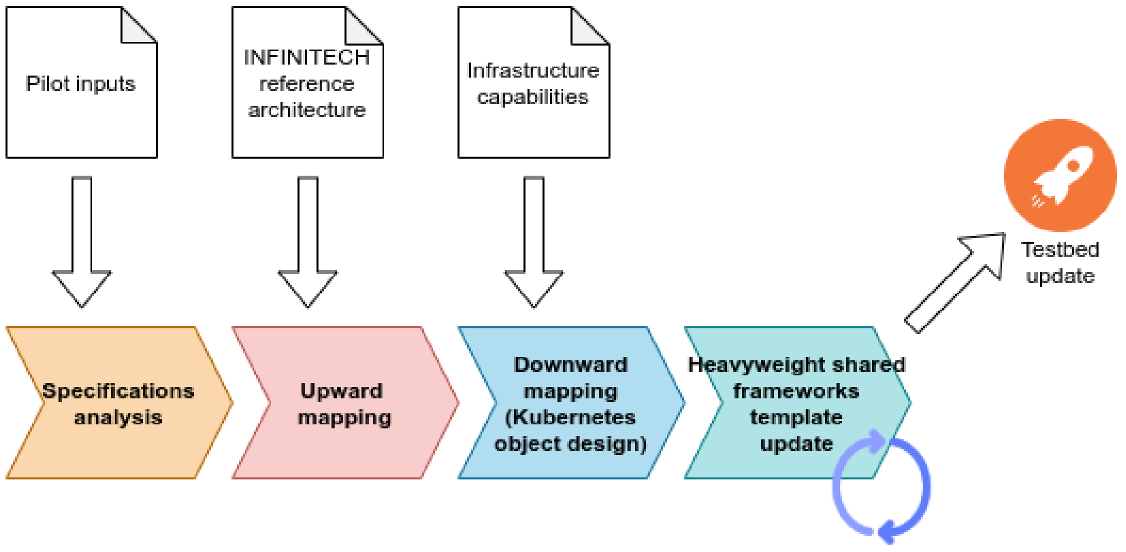
Pilot inputs
INFINITECH reference architecture
Infrastructure capabilities
Specifications analysis
Upward mapping
Downward mapping (Kubernetes object design)
Heavyweight shared frameworks template update
Testbed update

Figure 13 - Overall mapping workflow

After several per-pilot iterations of the process described above, in NOVA's infrastructure (as well as in an on-premise environment where these guidelines can be applied) a logical isolation through Kubernetes namespaces will be generated. The kube-system and istio-system namespaces will be in charge of executing a set of mandatory system components for the correct execution of the Kubernetes stack. Besides these namespaces, there will be in place a sandbox for each pilot that will be hosted on NOVA's available nodes as described in section 2.3. Then, a shared namespace will host all the heavyweight frameworks that work off the shelf and can be accessed from pilots' sandboxes, exposed through Kubernetes' external services, and deployed if and **only if they offer a native isolation mechanism** so that the interaction of a sandbox with the shared frameworks does not affect the interaction of another sandbox with the same framework (Figure 14).

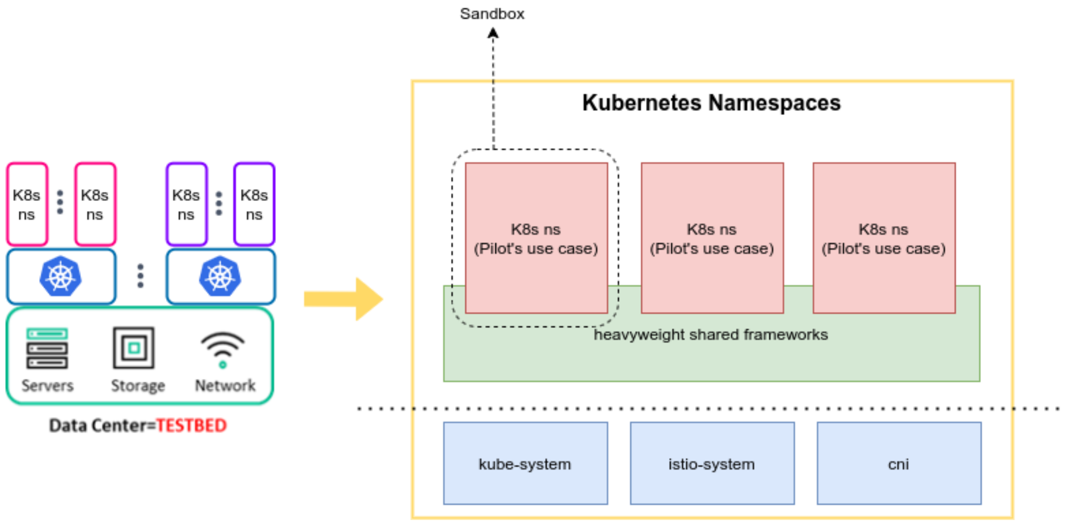
Sandbox
Kubernetes Namespaces
K8s ns (Pilot's use case)
K8s ns (Pilot's use case)
K8s ns (Pilot's use case)
heavyweight shared frameworks
kube-system
istio-system
cni
K8s ns
Servers
Storage
Network
Data Center=TESTBED

Figure 14 – NOVA shared testbed namespaces

Just as an example, if 3 out of 5 pilots require Hadoop HDFS and Apache Kafka to be in place as framework dependency of the INFINITECH's software they develop, it's reasonable to host them within the heavyweight shared frameworks' namespace since they both offer an independent isolation mechanism and both of them are meant to store and transfer a huge amount of data. This choice allows also freeing-up of a lot of system resources as there's no need to replicate "p times r replicas" where p is the number of pilots requiring the dependency and r is the number of replicas of the dependency. Digging deeper into the topic of the downward mapping and therefore, to the Kubernetes object design, here is reported an object example (a deployment) as a reference that will be used for every pilot involved in this activity.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: my-infinitech-component
    infinitech.ra: Ingestion (1)
  name: my-infinitech-component
  namespace: pilot-01 (2)
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-infinitech-component
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: app: my-infinitech-component
    spec:
      containers:(3)
      - image: infinitech-registry/my-infinitech-component:<sha-commit>
        imagePullPolicy: IfNotPresent
        name: my-infinitech-component
        ports: (4)
        - containerPort: 8080
          name: web
          protocol: TCP
        resources: {} (5)
        volumeMounts: (6a)
        - mountPath: /app/my-infinitech-component.properties
          name: configuration
          subPath: my-infinitech-component.properties
          name: my-infinitech-cm
        env: (7)
        - name: MY_ENV_VARIABLE
          value: "Hello variable"
      volumes: (6b)
      - configMap:
        name: configuration
```

As a good practice, a **(1)** “infinitech.ra” label will be added in order to keep a link to the INFINITECH Reference Architecture, this approach would be useful to perform filtering queries using the Kubernetes API as the testbed grows during its lifespan. If the app object contains several modules that belong to different INFINITECH RA groups, i.e. a deployment that has a pod specification with two or more containers, as many “infinitech.ra” labels can be appended in the label section of the object. In this way, the administrators have the possibility to perform informative queries for maintenance such as:

```
$ kubectl get all --all-namespaces --selector=infinitech.ra=ingestion
```

This command returns a list of all the Kubernetes objects in the testbed that are related to the ingestion INFINITECH reference architecture group. The sandbox isolation **(2)** is performed using separated namespaces in the Kubernetes NOVA cluster. The namespace has to be created before using it, issuing the following command from a client attached to one of the control plane master nodes:

```
$ kubectl create namespace <namespace name>
```

A list of containers is defined in section **(3);** the inputs asked in the pilots in the related table of the dedicated paragraphs, contribute to building up the information related to the port used **(4)**, the environment variables **(7)**, and the resource consumption/request to the Kubernetes cluster **(5)**. Last but not least, any application-related configuration file can be handled by defining a ConfigMap Kubernetes object **(6b)** and consuming it as a volume within the filesystem context of the container **(6a)**.

3.2 Pilot#2 - Real-time risk assessment in Investment Banking

3.2.1 Objectives, sandbox and technologies overview

As reported in the deliverable D2.13, the pilot will build a real-time risk assessment and monitoring pipeline for VaR (Value-at-Risk) and ES (Expected Shortfall) risk metrics. The components that will be used are described in the following table.

Table 1 – Pilot #2 technologies overview

Component name	**Type (DEP/INF)** **(please indicate if the component is a (third-party or proprietary) dependency) or an infinitech component to be developed**	**Resources: CPU, memory, storage to deploy the component**	**Exposed port (if applicable)**	**Environment variables**	**Deployment mode (legacy/container/ Kubernetes)**
BigData Management Layer	INF (under the proprietary rights of LXS)	Minimum: 4G RAM/ 2VCores	1529, 2181, 9876, 14400, 9992	Under development at the current phase	container
Custom Injection Simulator	INNOV (demo)				container
Kafka	DEP			ENV KAFKA_VERSION=$kafka_version \ SCALA_VERSION=$scala_version \ KAFKA_HOME=/opt/kafka \ GLIBC_VERSION=$glibc_version PATH=${PATH}:${KAFKA_HOME}/bin	container
Zookeeper	DEP		2181		container
PredictVaR	INNOV		-		container
VisualizeVaR	INNOV		8050		container

3.2.2 First stage components deployment and Local testbed system description

The pilot workflow can be analysed starting from the datasets that will be utilized. The proposed datasets consist of: i) historical market prices, ii) real time market prices iii) trading positions iv) text data (sentiment analysis). Both static data injection and dynamic data injection will be used in the frame of this pilot.

The pilot's architecture status is currently at version 0.2 (pilot2_v0.2), as described in D.7.1 and presented during the internal General Assembly meeting. In the next couple of months version 0.3 will be released. So, in the current deliverable both versions 0.1 and 0.2 will be described, furthermore future upgrades (v0.3) will be discussed as well, in order to present the roadmap and the challenges which arise in each step to accomplish the pilot's objectives, as per the following paragraphs. Finally, a table with the required libraries and frameworks is included.

<u>pilot2_v0.1:</u>

This implementation consists of the following four building blocks (containers):

1. <u>Database (LXS)</u>: This container is responsible for the required data storage. The historical data is downloaded and imported directly from the project repository (Gitlab) to the DB, while the "real time" data is injected to the DB via JDBC connection. It should be mentioned that SSH connection to the project repository is needed (historical data injection).
2. <u>Custom injection Simulator:</u> Due to the fact that an API to access real time data is not currently available, for the purpose of PoC, the given market data is split into two parts, the first of which serves as the historical market prices dataset while the other serves as the real time market prices dataset. Both the historical and real time data are stored in the DB via JDBC connection.
3. <u>VaR Prediction</u>: This container is responsible for the main tasks of this pilot:
    a. Reads both the historical and new data available from the DB
    b. Preprocesses the input data and writes the clean ones to the DB
    c. Performs VaR calculation and writes the results to the DB
4. <u>Visualization:</u> This container serves as the pilot's graphical user interface. It is a web application currently using Python Flask framework.

<u>pilot2_v0.2:</u>

This implementation consists of the following six building blocks (containers):

1. <u>Database (LXS)</u>: same as v0.1.
2. <u>Custom Injection Simulator</u>: The main difference between v0.1 and v0.2 is that the preprocessed "real time" data is published in the kafka pub/sub. The historical prices are injected once in the LXS DB as in v0.1. The main scope for this change is to demonstrate the capability of incorporating kafka pub/sub service which will be used in the final version of the pilot.
3. <u>Kafka/Zookeeper:</u> These containers provide the link between the "Custom Injection Simulator" and the "VaR Prediction".
4. <u>VaR Prediction</u>: This container is responsible for the main task of this pilot, performing the following:
    i. Reads the historical data from the DB
    ii. Preprocesses the input data and writes the clean data to the DB
    iii. Reads "real time" data published from the "Custom Injection Simulator" to the kafka queue

iv. Performs VaR calculation and writes the results to the DB

5. Visualization: same as v0.1.

pilot2_v0.3 (future work):

1. Database (LXS): The difference between earlier versions and this one will be that real time data will be injected as a stream, possibly utilizing Apache kafka streaming platform.
2. Custom Injection Simulator: Given that real time data will be provided by the JRC and LXS kafka interface for storing real time tick data, the simulator will not be needed.
3. VaR Prediction: Will retrieve incrementally data from LXS DB (either kafka or JDBC connection). Moreover, Spark may also be utilised in order to parallelize calculations.
4. Visualization: Flask will be used in conjunction with an application server and load balancer instead of using Flask's built-in web server.

Next versions:

Market Sentiment components will be included. The main task of this container will be to retrieve alternative data (e.g., data from news) that will be used for market sentiment analysis based on NLP (Natural Language Processing Techniques).

Provides an interface for accessing to FIBO data, while supporting their parsing. The semantic annotation and the structuring of the data according to FIBO is performed in WP4 and hence the relevant description is beyond the scope of this deliverable.

3.2.3 TO-BE strategy and mapping according the INFINITECH way

The upward mapping, according to the information provided by Pilot #2 and to the INFINITECH Reference Architecture, gives results as follows:

Table 2 - Pilot #2 mapping with INFINITECH Reference Architecture Group

INFINITECH Reference Architecture Group	Component
Data Source	
Data Ingestion	Custom Injection Simulator
Data Management	Big Data Management Layer
Data Security and Privacy	
Blockchain and Information Sharing	
Data Model and Semantics	
Internet of Things	
Analytics and Machine Learning	PredictVaR
Interface	VisualizeVaR
Cross Cutting	Zookeeper namespace Kafka

At the first instance, the downward mapping can be summarized using the following table, in which the Kubernetes API resources used in the sandbox are specified per component. The content of the list may vary during the development of the activities performed within task T6.5, according to the evolution of the development of the involved INFINITECH components. A sprint can easily re-shape some of the objects listed below at any time.

Table 3 - Pilot #2 mapping with Kubernetes objects

Component	**Componentstatus**	**Configmap**	**Endpoint**	**PersistentVolumeClaim**	**PersistentVolume**	**PodTemplate**	**Secret**	**ServiceAccount**	**Service**	**Daemonset**	**Deployment**	**Statefulset**	**HorizontalPodAutoscaler**	**CronJob**	**Job**	**Ingress**
Custom Injection Simulator		X									X					
Big Data Management Layer		X		X	X				X			X				
PredictVaR									X		X					
VisualizeVaR									X		X					X
Zookeeper				X	X				X			X				
Kafka							X		X			X				

3.3 Pilot#11 - Personalized insurance products based on IoT connected vehicles

3.3.1 Objectives, sandbox and technologies overview

Pilot#11 is framed within Cluster #4 that relies on AI technologies and models to exploit wide IoT datasets from end users and related context. The objective of the cluster is to update and enhance the common methodologies the insurance companies use to analyze the different risks tied to the offered products whilst allowing them to tailor the offered services according to the insured client's behaviour. This way (please see D7.1 for further details), the pilot focuses on its car insurance environment to develop two AI powered services: Pay as you Drive, that allows the insurance company to adapt prices by classifying the driver according to the way he/she drives; and Fraud Detection which helps to identify the actual driver of a vehicle involved in an accident. These two services rely on a driving profiling tool that requires datasets from connected vehicles to define, identify and train the different profiles as ML models. Other external data sources, such as traffic accidents or weather, will be used to classify the driver, contextualizing its assigned driving profile.

The following table shows a detailed list of the components, their roles within the architecture and some deployment detail that will be subject to some slight changes in the near future.

Table 4 - Pilot #11 technologies overview

Component name	**Type (DEP/INF)** **(please indicate if the component is a (third-party or proprietary) dependency) or an infinitech component to be developed**	**Resources: CPU, memory, storage to deploy the component**	**Exposed port (if applicable)**	**Environment variables**	**Deployment mode (legacy/container/ Kubernetes)**
Connected Car Framework (Context Broker)	DEP	CPU: 1 Small + 2 Medium Nodes (9 Cores total) 20 GB RAM 1 TB Storage	n/a		Kubernetes
Connected Car Framework (PeP Proxy)	DEP		1 port to be provided in deployment phase		Kubernetes
Connected Car Framework (Historical DB: CrateDB)	DEP		(if required) 1 port to be provided in deployment phase		Kubernetes
Connected Car Framework (Context DB: Mongo)	DEP		1 port to be provided in deployment phase (27017)		Kubernetes
Connected Car Framework (QuantumLeap)	DEP		1 port to be provided in deployment phase		Kubernetes
Connected Car Framework (Weather Injector)	INF		n/a	AEMET API Key; List of Weather Staions' Ids to monitor	Kubernetes

Connected Car Framework (IoT Agent)	INF/DEP		n ports required for n sources injecting (1 port required for this deployment)		Kubernetes
Connected Car Framework (Grafana)	DEP		1 port to be provided in deployment phase		Kubernetes
Security Framework (IDM)	DEP		1 port to be provided in deployment phase		Kubernetes
Anonymiser (GRAD Anonymiser)	INF/DEP	1 Node (4 Cores) 16 GB RAM 1 TB Storage	TBD		TBD
EASIER.AI (Elasticsearch)	DEP	CPU: 1 Small + 2 Medium + 1 Large + 1 XXL Nodes (21 Cores total) 68 GB RAM 4 TB Storage GPU, FPGA, TPU, or other special AI hardware. AVX2 Instructions set support	n/a		Kubernetes
EASIER.AI (kibana)	DEP		1 port to be provided in deployment phase		Kubernetes
EASIER.AI (logstash)	DEP		n/a		Kubernetes
Pay as You Drive Service	INF	As a first approach, EASIER infrastructure will be used	TBD		Kubernetes
Fraud Detection Service	INF	As a first approach, EASIER infrastructure will be used	TBD		Kubernetes

3.3.2 First stage components deployment

The final implementation of Pilot#11 requires data from the components set shown in Figure 15, extracted from D7.1 and D6.1. These will be deployed within the NOVA infrastructure, using the INFINITECH CI/CD methodology (based on docker and Kubernetes technologies). In this sense:

- **IoT infrastructures** and **Data Normalization** (from D7.1) components include a set of Data Injectors, based on NGSI-LD and FIWARE data models, developed in Python and Node and distributed as docker images.
- **Data Collection & Aggregation** and **Data Processing** components constitute the Connected Car framework. This is composed of the FIWARE Orion Context Broker, that supports all context management functionalities (context information broker), and an instance of the FIWARE QuantumLeap General Enabler (context information persistence) that supports historical information management. A first instance, covering these two components, has been implemented and deployed in Atos' infrastructure. The full composition of this building block will be provided as docker images and Kubernetes yaml files.
- Gradiant's **Anonymizer** tool implementation will be also provided according to CI/CD guidelines.
- **EASIER-AI** is the AI Hybrid (Cloud/Edge) framework that helps to develop, measure, monitor and deploy customised AI models. It is built on top of the Elastic Search, Kibana and TensorFlow slate of three and enables different ML/DL technologies deployment.
- The access to these frameworks (Connected Car and EASIER-AI) is protected by an OAuth **identification and authentication** component that relies on the FIWARE KeyRock IdM. SSL/TLS is used to protect communications. This is deployed and integrated with the Connected Car framework, and as all components of the P#11 framework, will be uploaded and distributed according to INFINITECH CI/CD templates.

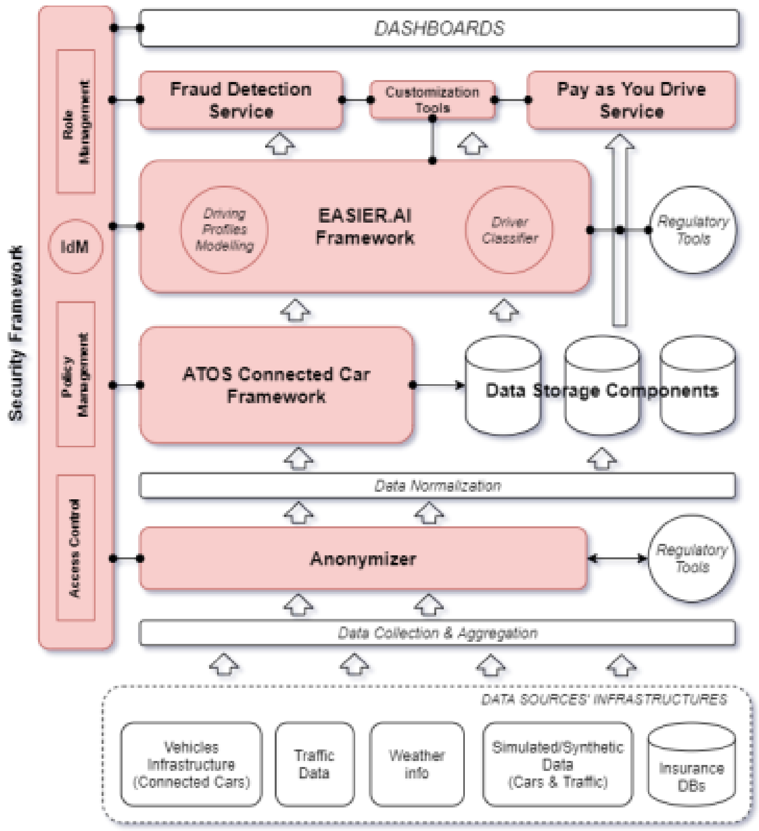

Figure 15 - Pilot #11 High Level Architecture

3.3.3 TO-BE strategy and mapping according to the INFINITECH way

As described by the Pilot, there are several components to be classified and mapped to the INFINITECH Reference Architecture and to the sandbox. A first logical architecture is described, as usual, in document D2.13 and, in the following table, there is the classification of each module and the label that refers to the macro-component using the glossary adopted by the pilot:

Table 5 - Pilot #11 mapping with INFINITECH Reference Architecture Group

INFINITECH Reference Architecture Group	**Component**
Data Source	(Connected Car Framework) CrateDB (Connected Car Framework) MongoDB (Connected Car Framework) QuantumLeap
Data Ingestion	(Connected Car Framework) WeatherInjector (EASIER.AI) Logstash
Data Management	(Connected Car Framework) Context Broker (EASIER.AI) Elasticsearch
Data Security and Privacy	(Connected Car Framework) PeP Proxy (Security Framework) IDM (Security Framework) Anonymizer
Blockchain and Information Sharing	
Data Model and Semantics	
Internet of Things	(Connected Car Framework) IoT Agent
Analytics and Machine Learning	(EASIER.AI) Kibana
Interface	(Connected Car Framework) Grafana Pay as You Drive Service Fraud Detection Service
Cross Cutting	

With regards to the downward mapping, it is worth specifying that not every component corresponds 1:1 to a Kubernetes object file. One or more components as a container may be declared in the same Kubernetes application object.

Table 6 – Pilot #11 mapping with Kubernetes objects

Component	Componentstatus	Configmap	Endpoint	PersistentVolumeClaim	PersistentVolume	PodTemplate	Secret	ServiceAccount	Service	Daemonset	Deployment	Statefulset	HorizontalPodAutoscaler	CronJob	Job	Ingress
CrateDB		X		X	X		X		X			X				
MongoDB				X	X		X		X			X				
QuantumLeap		X		X	X				X							
WeatherInjector											X					
Logstash		X		X	X		X	X	X			X				X
Context Broker											X					
Elasticsearch		X		X	X			X	X			X				X
PeP Proxy									X		X					X
IDM				X	X				X		X				X	X
Anonymizer		X					X		X		X					
IoT Agent		X									X					
Grafana		X		X	X		X	X	X		X	X				X
Kibana		X							X		X					X
Pay As You Drive Service		X							X		X					X
Fraud Detection Service		X							X		X					X

3.4 Pilot#12 - Real World Data for Novel Health-Insurance products

3.4.1 Objectives, sandbox and technologies overview

Pilot#12 focuses on health insurance and risk analysis by developing two AI-powered services: Risk assessment, that allows the insurance company to adapt prices by classifying the client according to their lifestyle; and the Fraud Detection which helps to identify fraudulent behaviour of the clients by using the activity trackers and answering health questionnaires. These two services rely on people modelling that requires actual data and simulated persons to train. An overview of Pilot #12 is given in Figure 16.

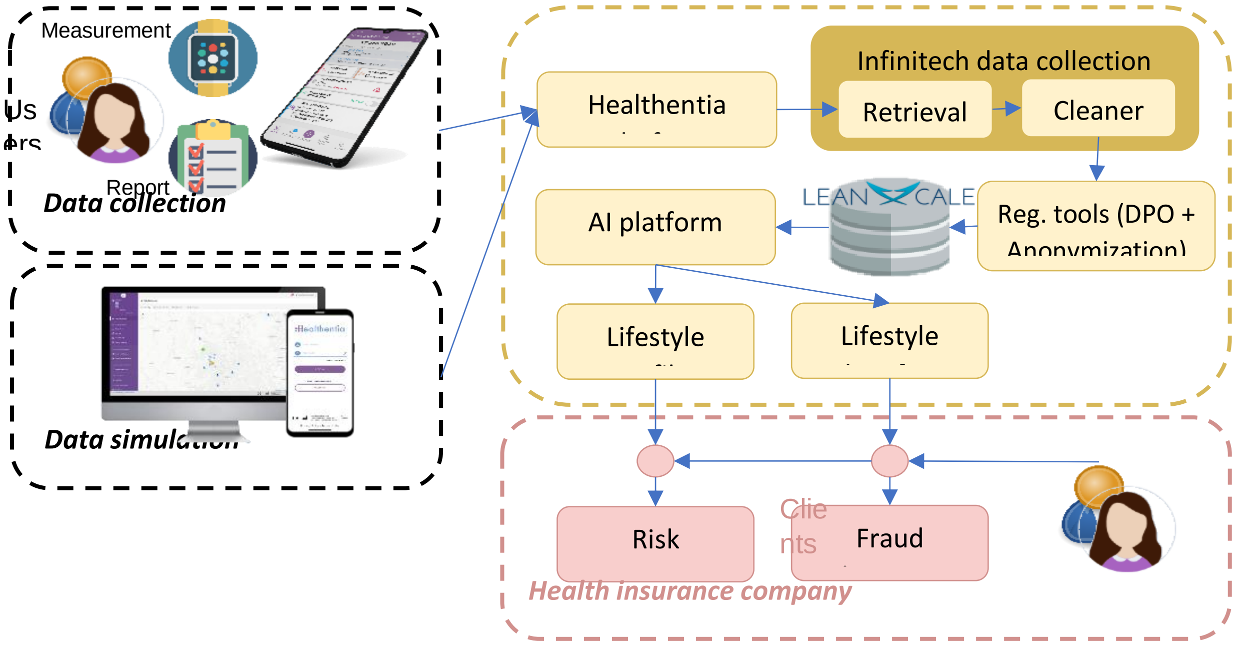

Figure 16 - Pilot #12 Real world data for novel health insurance products overview

Table 7 - Pilot #12 technologies overview

Component name	**Type (DEP/INF)** **(please indicate if the component is a (third-party or proprietary) dependency) or an infinitech component to be developed**	**Resources: CPU, memory, storage to deploy the component**	**Exposed port (if applicable)**	**Environment variables**	**Deployment mode (legacy/container/ Kubernetes)**
UBITECH Data Capturing Tool	INF	4 processors / 18 cores, 64 GB memory, 1TB storage			Container
LeanXcale Database	DEP		Port 1529 is exposed from the LeanXcale docker container to the VM itself and also by the VM to the internet	None	Container
Innovation Sprint's ML services (risk assessment and fraud detection)	INF (this is a Python app with dependencies on 3rd party libraries. The dependencies are listed in requirements.txt and are installed via pip)		None	Configuration variables in a .env file: HEALTHENTIA_API_KEY & HEALTHENTIA_PORTAL_URI (communicate with Healthentia API), SECRET_KEY (authentication)	Container
ATOS Regulatory tool through Data protection Orchestrator (DPO)	DEP		1 port to be provided in deployment phase	None	Container
GRAD Regulatory tool through Anonymization Component	INF		At least 1 port to be provided in deployment phase	TBD	Container

3.4.2 First stage components deployment

Since the final pilot #12 testbed to be hosted by NOVA is not yet available, a temporary testbed is setup by Innovation Sprint. It comprises a VM with 2 vCPUs, 8 GB RAM, 80 GB storage hosted on Hetzner.com (CX31 instance).

The software is setup on Linux 2020LTS, currently including:

- Ubitech's Data Capturing Tool (configured to capture data from Healthentia API, https://saas-api.healthentia.com/swagger/index.html),
- LeanXcale database, and
- Innovation Sprint's ML services (risk assessment and fraud detection).

Installation and usage of LeanXcale is done as follows:

- git clone https://gitlab.infinitech-h2020.eu/pkranas/lxs-store/
- docker build -t lx-store.
- docker run -d --name datastore -p 1529:1529 {image_id}

The latter will start a container in the background, which will install and start LXS, and will expose the 1529 port to the host machine. 1529 is the port that the query engine is listening to and at which the connection is established. The included SQL client called lxClient resides inside the lxs container.

The main DB connection string is: jdbc:leanxcale://135.181.144.50:1529/INFINITECH;user=pilot12

- Regulatory tools through DPO implementation and anonymization components will be provided according to CI/CD guidelines.

3.4.3 TO-BE strategy and mapping according the INFINITECH way

The pilot adopts five key modules: two of them are marked as a dependency and three of them are INFINITECH-related artifacts. They can be regrouped as follows:

Table 8 - Pilot #12 mapping with INFINITECH Reference Architecture Group

INFINITECH Reference Architecture Group	Component
Data Source	
Data Ingestion	UBITECH Data Capturing Tool
Data Management	LeanXcale Database
Data Security and Privacy	ATOS Regulatory tool through Data protection Orchestrator (DPO) GRAD Regulatory tool through Anonymization Component
Blockchain and Information Sharing	
Data Model and Semantics	

Internet of Things	
Analytics and Machine Learning	Innovation Sprint’s ML services
Interface	
Cross Cutting	

Each component is arranged in the Pilot 12 sandbox according to the scheme depicted in the table below. No hybrid environment exceptions are detected for this pilot.

Table 9 – Pilot #12 mapping with Kubernetes objects

Component	Componentstatus	Configmap	Endpoint	PersistentVolumeCl aim	PersistentVolume	PodTemplate	Secret	ServiceAccount	Service	Daemonset	Deployment	Statefulset	HorizontalPodAuto scaler	CronJob	Job	Ingress
UBITECH Data Capturing Tool		X		X	X				X			X				
LeanXcale Database		X		X	X		X		X			X				X
ATOS Regulatory tool through Data protection Orchestrator (DPO)		X		X	X				X			X				
GRAD Regulatory tool through Anonymization Component		X					X		X		X					
Innovation Sprint’s ML services															X	

3.5 Pilot#13 - Alternative/automated insurance risk selection - product recommendation for SME

3.5.1 Objectives, sandbox and technologies overview

Pilot #13 will monitor a risk's changes, so it will be able to radically improve the risk management that small and medium enterprises face in the development of their daily activity. The indicators will be based on information from each of the companies coming from online sources, providing information about the digital presence and activity of those companies like activity, business volume, participation in social networks, number of employees, use of ecommerce, payment platform, etc, etc. The company to be analysed does not need to provide much information, developed tools are in charge of searching and gathering information related to that company from many sources. In this way, risk profiles of each of the companies analysed will be generated, allowing customization of the product offering and the addition of permanent automated risk management. But this is not the only usage of data, insurance companies will use wider information, resulting in better customized products.
An overview of Pilot #13 is provided in Figure 17.

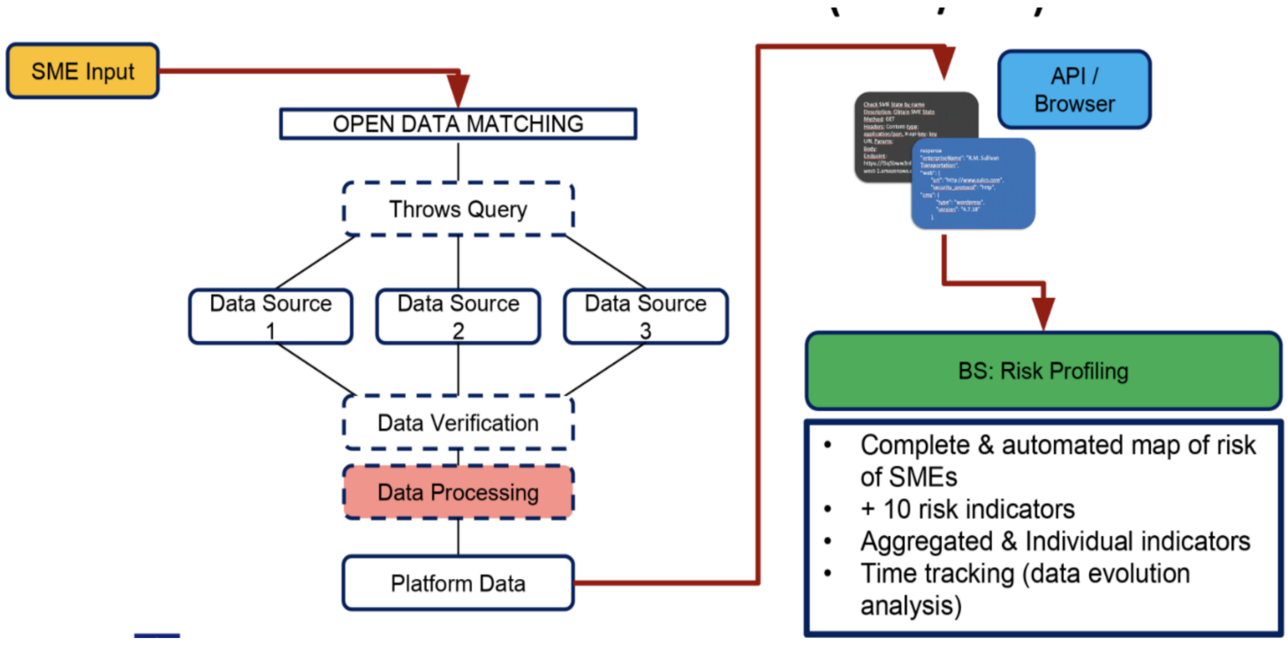

Figure 17 - Pilot #13 overall architecture

In the overall architecture, there can be identified three layers of building blocks of the overall solution:

- Data Acquisition Layer. This layer is used to obtain data from the information sources which are related to the digital presence and activities of the consumers, using automations developed by WEA. It is considered external to the INFINITECH sandbox and will feed the latter with the aggregated information after the initial pre-processing and preparation of the data.

- Data Management Layer. This layer is used to store the data coming from the previous layer and allow for their efficient processing. It will make use of INFINITECH components developed in the project, and more precisely the *infinistore* which includes the HTAP Data store along with its extensions with the polyglot engine.
- Analytics Layer. This layer contains the AI algorithms and analytics that make use of the data available by the data management layer. These AI analytics will be developed by WEA and will be pilot-specific, solving the needs of the pilot by exploiting the technologies provided by INFINITECH for efficient data processing of data coming from a variety of heterogeneous resources. They will be hosted inside the WeAnalyze platform external to the deployed sandbox.
- Visualization Layer. This layer contains the REST endpoints and visualization components that enable the end-users to see the results of the analysis via a User Interface, by importing them to their own applications via the use of the REST APIs. It will consume the results of the analytics layer which will be deployed in the premises of WEA, and therefore, there is no need for the components of this layer to be hosted inside the sandbox deployed in the NOVA infrastructure.

As we can see, the Data Management Layer of the overall solution consists of the *infinistore* component, which encloses both the HTAP data store and the polyglot engine. The following table contains the relevant information for this component.

Table 10 - Pilot #13 technologies overview

Component name	**Type (DEP/INF)** **(please indicate if the component is a (third-party or proprietary) dependency) or an infinitech component to be developed**	**Resources: CPU, memory, storage to deploy the component**	**Exposed port (if applicable)**	**Environment variables**	**Deployment mode (legacy/container/ Kubernetes)**
INFINISTORE	INF	1st phase: CPUS: 2 cores Memory: 8GB Storage: 100GB However these requirements will be further refined in the 2nd phase of implementation, when this component needs to serve additional volume of data	1529, 2181, 9876, 14400, 9992		Kubernetes
Data Acquisition Layer	DEP				3rd party (sidecar deployment)
Analytics Layer	DEP				3rd party (sidecar deployment)
Visualization Layer	DEP				3rd party (sidecar deployment)

3.5.2 First stage components deployment

At the time that this document is being written, the NOVA infrastructure was not yet available to the consortium. In order to accelerate the development process for the first phase of this pilot, we made use of AWS resources to deploy the components that are part of the INFINITECH platform and should be deployed inside the sandbox. Regarding the data collection and preparation, this has been done offline and a dataset was created to be migrated to the infinistore. This process was made manually using the tools provided by the infinistore. The ML algorithms were kept in premise of WEA and developers were given access to the AWS machines where the data management layer had been deployed, allowing the connection to a specific port. As these algorithms will be hosted inside the platform of WEA, this will remain the same after the deployment of the sandbox in the NOVA infrastructure.

The migration and the deployment into the NOVA infrastructure will be facilitated using Kubernetes deployment orchestration. For the first stage components deployment, the same resources will be needed, as in AWS, as at this phase, the pilot is targeting a small part of data. Once the *infinistore* component is deployed, then the ML tools developed by WEA for the needs of the pilot will need to connect to a specific port that needs to be exposed by the sandbox. Moreover, the integration with the Data Acquisition layer needs to take place, in order to automate the process of obtaining data from the information sources and storing them into the Data Management layer, following all steps of data preparation and integration. The data acquisition layer well be hosted outside of the sandbox and it will be required to access the data management layer to additional ports. For this data connectivity, client/connectors developed in the scope of INFINITECH will be used to allow this connectivity, however, as they are client libraries, they will be included in the Data Acquisition layer and therefore, they will be outside of the testbed.

3.5.3 TO-BE strategy and mapping according to the INFINITECH way

The upward mapping is straightforward having one component under development in the Data Management group. The other component are external and match the sidecar scenario described in section 2.5.

Table 11 - Pilot #13 mapping with INFINITECH Reference Architecture Group

INFINITECH Reference Architecture Group	Component
Data Source	
Data Ingestion	Data Acquisition Layer
Data Management	INFINISTORE
Data Security and Privacy	
Blockchain and Information Sharing	
Data Model and Semantics	
Internet of Things	
Analytics and Machine Learning	Analytics Layer
Interface	Visualization Layer
Cross Cutting	

The downward mapping results as per the table:

Table 12 – Pilot #13 mapping with Kubernetes objects

Component	**Componentstatus**	**Configmap**	**Endpoint**	**PersistentVolumeClaim**	**PersistentVolume**	**PodTemplate**	**Secret**	**ServiceAccount**	**Service**	**Daemonset**	**Deployment**	**Statefulset**	**HorizontalPodAutoscaler**	**CronJob**	**Job**	**Ingress**
Data Acquisition Layer									X							
INFINISTORE		X		X	X		X		X			X				X
Analytics Layer									X							
Visualization Layer									X							

3.6 Pilot#14 - Big Data and IoT for the Agricultural Insurance Industry

3.6.1 Objectives, sandbox and technologies overview

The objective of Pilot #14 "Big Data and IoT for the Agricultural Insurance Industry" is to deliver a commercial service module that will enable insurance companies to exploit the untapped market potential of Agricultural Insurance (AgI), taking advantage of innovations in Earth Observation (EO), weather intelligence & ICT technology. EO will be used to develop the data products that will act as a complementary source to the information used by insurance companies to design their products and assess the risk of natural disasters. Weather intelligence based on data assimilation, numerical weather prediction and ensemble seasonal forecasting will be used to verify the occurrence of catastrophic weather events and to predict future perils that could threaten the portfolio of an agricultural insurance company. The INFINITECH AgI-module derived indices will allow and enable the agricultural insurance industry to enlarge its market, while delivering a larger portfolio of products at lower costs and serve areas where classical insurance products could not be delivered.

AgroApps is developing the entire infrastructure for the pilot #14 data products, based on the reference architecture starting from data collection from different sources, over processing and analytics, to user interface & data visualization. The ongoing development of the service module is based on scientific research in the field of agricultural insurance, climate & weather risk modelling and the most recent evolutions in the area of remote sensing technologies. The reason for this is that these three areas will play a crucial role for the future of agricultural insurance providers in order to tap new markets, provide better risk-transfer solutions and make insight-based strategic decisions. To meet the demands of this rapidly evolving field, it is necessary to follow these current developments.

As described in the User Stories (please see D.2.1 for further details), the service module is mainly designed for staff of the underwriting and sales department of agricultural insurance companies (the majority of User stories serves this group of end-users). However, within those departments, there are several roles that can benefit from the services provided by Pilot #14. First of all, Actuaries (business professionals /mathematicians who analyze the financial consequences of risk by using statistics) are able to improve their data set for risk pricing and product development based on the data retrieved from the service module. Based on this information, Underwriters can better evaluate the risk and exposure of potential clients (crop monitoring) and hence make the overall insurance portfolio more resilient by at the same time increasing the outreach to clients (farmers). Additionally, Sales Agents can identify areas where they can prioritize sales activities without increasing the cumulative risk since they are aware of key factors, e.g. regional risk profiles.

Lastly, with the support of data derived from the Octopush EO (damage assessment services), loss adjusters have additional information to make the on-farm process of loss adjusting more efficient and for certain perils they conduct this process remotely via the service module (without visiting the farm/respective field).

In addition to the implementation within insurance companies, at a later stage of the project other users in the insurance value chain can also be considered as end users (see D.2.1.).

A first contact inside an insurance company in the Area of Interest (Serbia) has been made and immediately generated interest because of the benefits the Pilot #14 service module has to offer. The feedback on the presented services was very positive, just a final decision by the management is outstanding.

As in this very first stage of the preparation of the pilot site the receiving of an appropriate and high-quality dataset from the pilot insurance company and the application of the services described in Pilot #14 have highest priority, there are no training plans developed for deployment to the final user yet. However, for the internal deployment at the final pilot site, Pilot #14 can provide an independent web-based user interface for the end users to access the service module via their browser.

Based on the reference architecture (please see D2.13, Figure 43) as well as the Report on Pilot Sites Preparation (please see D7.1), the following components and services will be deployed and used as part of the pilot:

Table 13 - Pilot #14 technologies overview

Component name	**Type (DEP/INF)** **(please indicate if the component is a (third-party or proprietary) dependency) or an infinitech component to be developed**	**Resources: CPU, memory, storage to deploy the component**	**Exposed port (if applicable)**	**Environment variables**	**Deployment mode (legacy/container/ Kubernetes)**
Octopush Service	DEP	vCPU8, 64GB RAM, 10TB Storage			Docker Container (sidecar deployment)
AgroApps Weather Engine (AgroApps WIE)	DEP	vCPU240, 512GB RAM, 10TB Storage			Singularity Container
Data Integrator	DEP				Docker Container (sidecar deployment)
Business and Geospatial DB	DEP	375GB			Docker Container (sidecar deployment)
Web Map Server (WMS Server)	DEP	vCPU2, 64GB RAM, 1TB Storage			Docker Container (sidecar deployment)
RESTful API	DEP				Docker Container (sidecar deployment)
User Interface	INF				Docker Container (sidecar deployment)

3.6.2 First stage components deployment

To ease the development and the deployment of the INFINITECH pilot 14 platform in new computers and systems, the platform uses a virtualization ecosystem based on Docker containers. Currently the INFINITECH pilot 14 stack consists of the following containers:

- PHP 7 with Apache HTTP Server and Python 3 – For the Lumen application and the EO data retrieval tool
- PostgreSQL 10 with PostGIS 2.5 – For the relational storage needs of the platform
- Apache Tomcat 8 with GeoServer 2.13 – For rendering and serving the satellite index data
- Nginx 1.14 – For serving the web application (front-end)
- WRFv4.2 – For the numerical weather prediction system (Weather Intelligence Engine)
- WRFDA – For the atmospheric data assimilation system (Weather Intelligence Engine)
- UPPv4.1 – For the post-processing of the atmospheric fields
- METv9.1- For the verification of the atmospheric fields

3.6.3 TO-BE strategy and mapping according to the INFINITECH way

Pilot #14 has a software logical schema that makes use of the following modules:

Table 14 - Pilot #14 mapping with INFINITECH Reference Architecture Group

INFINITECH Reference Architecture Group	Component
Data Source	(Octopush Service) Apache HTTP Server with PHP (AgroApps Weather Engine) WRFv4.2 (AgroApps Weather Engine) WRFDA (AgroApps Weather Engine) UPPv4.1 (AgroApps Weather Engine) METv9.1
Data Ingestion	Data Integrator
Data Management	(Business and GeospatialDB) PostgreSQL PostGIS
Data Security and Privacy	
Blockchain and Information Sharing	
Data Model and Semantics	
Internet of Things	
Analytics and Machine Learning	(WebMAP Server) Geoserver
Interface	(WebMAP Server) Apache Tomcat (WebMAP Server) RESTful API
Cross Cutting	

The translation of the logical schema into Kubernetes objects is defined as the first instance as follows:

Table 15 – Pilot #14 mapping with Kubernetes objects

Component	Componentstatus	Configmap	Endpoint	PersistentVolumeClaim	PersistentVolume	PodTemplate	Secret	ServiceAccount	Service	Daemonset	Deployment	Statefulset	HorizontalPodAutoscaler	CronJob	Job	Ingress
(Octopush Service) Apache HTTP Server with PHP									X							
AgroApps Weather Engine		X		X	X				X			X				X
Data Integrator									X							
PostGIS									X							
Geoserver									X							
WebMAP Server									X							

AgroApps' Weather Intelligence engine will be the only set of software that will be hosted on the NOVA testbed due to an extrinsic constraint, therefore a sidecar deployment scenario is applied. As reported in the previous table, the four instances of AgroApps Weather Engine are grouped into a single entry and launched as a Statefulset, all the other components will only have Kubernetes Services of kind ExternalName. These services, initialized into Pilot 14's namespace, will work as network references to AgroApp's managed premises, and allow every sandbox module to deliver requests and receive responses from the remote infrastructure.

3.7 Shared frameworks and dependency recurrency

From a first analysis of the information collected by the technical proxies and the pilots, it is possible to extract an important consideration related to the use of all those framework and heavyweight standalone dependencies that can be eligible to take part in the template of a shared sandbox within the NOVA testbed, in particular, it is worth highlighting the frequent presence of the following technologies:

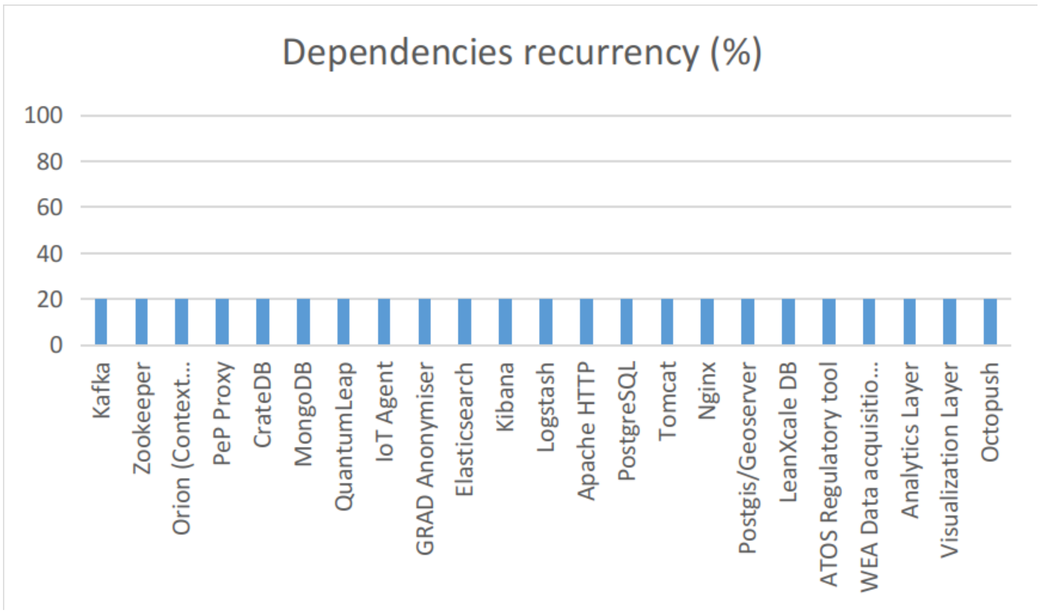

Figure 18 – Dependencies recurrency among pilots

The histogram in Figure 18 is flat, meaning that there are no emerging dependencies worth including in the heavyweight shared frameworks sandbox. For the specific context of INFINITECH Sandboxes for FinTech and InsuranceTech innovators there is no need to build a shared-dependencies sandbox within the NOVA environment, but the concept can be applied in other cases where a new self-hosted testbed has to be initialized to be federated to take part to the INFINITECH development process and to optimize the consumption of the available resources.

4 Conclusions

This document D6.10 - Sandboxes for FinTech/InsuranceTech Innovators – I, has reported the first result of INFINITECH WP6 Tasks T6.5 - FinTech/InsuranceTech Testbed Establishment and Customization.

It describes how specific machines provided by NOVA will be used for the first approach of testbed tailoring on-premise within the INFINITECH project, where sandboxes for pilots 2, 11, 12, 13, 14 will be hosted.

Then the per-pilot process was documented to serve as a guideline to be adopted by future cases, and it consists of three steps: *(i)* the definition of the objective and the overview of the pilot that is a piece of information that provides the main idea behind the pilot at a glance; *(ii)* the first stage component deployment which contains information about how the software and its dependencies (I.e. framework used, 3rd party software, etc...) are currently built and deployed, their requirements and compatibilities, their condition in the network (I.e. the ports a certain service listens to); *(iii)* the TO-BE strategy mapping which consumes all the information provided in the previous steps, combines them with the inputs coming from the deliverable and produces the objects and the software configuration files, described in the document in an easily readable diagram form, for the target environment (I.e. Kubernetes objects and templates). This bundle of artifacts will generate the sandboxes that will be ready to be used during the whole development process.

The work has been carried out in close cooperation and coordination with the other INFINITECH WP6 tasks and work packages 2-3-4-5 tasks and partners, taking into account and integrating the delivered results as inputs to this document, in accordance with the INFINITECH DoW.

The overall progress of such WP6 tasks will be one of the major drivers of the INFINITECH work package dedicated to the Large Pilots Operations and Stakeholders Evaluation of the proposed Financial and Insurance Services (WP7).

Within Tasks T6.5, two additional deliverables are expected at M24 and M33 respectively of the project timeline. At M24 a report will be produced with all the technical details of the actual deployment of pilot sandboxes for the second project iteration on the target NOVA infrastructure. At M33 it is planned a final report that will gather basic and advanced metrics related to the usage of the sandboxes and the infrastructure during the development activities of the pilots.

Appendix A: Literature

[1] Kubernetes. [Online]. Available: https://kubernetes.io/.

[2] Cilium. [Online]. Available: https://cilium.io/.

[3] «Hubble,» [Online]. Available: https://docs.cilium.io/en/v1.9/intro/#intro.

[4] VMware. [Online]. Available: https://www.vmware.com/.

[5] v. ESXi. [Online]. Available: https://www.vmware.com/products/esxi-and-esx.html.

[6] Vagrant. [Online]. Available: https://www.vagrantup.com/.

[7] «Terraform,» [Online]. Available: https://www.terraform.io/.

[8] «Terraform homepage,» HashiCorp, [Online]. Available: https://www.terraform.io/docs/language/files/index.html.

[9] «Oralce Virtual Box,» [Online]. Available: https://www.virtualbox.org/.

[10]«VMware Workstation,» [Online]. Available: https://www.vmware.com/products/workstation-pro.html.

[11]Ruby. [Online]. Available: https://www.ruby-lang.org/en/.

[12]«EC",» [Online]. Available: https://aws.amazon.com/ec2/instance-types/.

[13]«HCL,» [Online]. Available: https://www.terraform.io/docs/configuration-0-11/syntax.html.

[14]«Cloning,» [Online]. Available: https://sdorsett.github.io/post/2018-12-24-using-terraform-to-clone-a-virtual-machine-on-vsphere/).

[15]«rancher,» [Online]. Available: https://rancher.com/.

[16]«CNCF,» Linux Foundation, [Online]. Available: https://www.cncf.io/.

[17]«Open-LDAP,» Openldap Foundation, [Online]. Available: https://www.openldap.org/.

[18]«MetalLB,» [Online]. Available: https://metallb.org/.

[19]«HELM,» [Online]. Available: https://helm.sh/.

[20]«GitLab homepage,» GitLab, [Online]. Available: https://about.gitlab.com/.

[21]«Harbor homepage,» Linux Foundation, [Online]. Available: https://goharbor.io/.

[22]«Jenkins homepage,» [Online]. Available: https://www.jenkins.io/.

[23]https://kubernetes.io/docs/concepts/services-networking/service/#choosing-your-own-ip-address