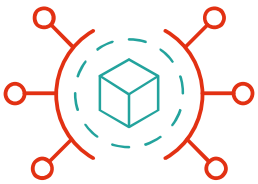


New ICT infrastructure and reference architecture to support Operations in future PI Logistics NETworks

D2.20 ICONET PI Control and Management Platform – Intermediate Version

Document Summary Information

Grant Agreement No	769119	**Acronym**	ICONET
Full Title	New CT infrastructure and reference architecture to support perations in future PI Logistics works		
Start Date	01/09/2018	**Duration**	30 months
Project URL	https://www.iconetproject.eu/		
Deliverable	D2.20 ICONET PI PoC Integration Platform – Intermediate Version		
Work Package	WP2		
Contractual due date	M18	**Actual submission date**	26th February 2020
Nature	Other	**Dissemination Level**	Public
Lead Beneficiary	IBM		
Responsible Author	Kieran Flynn (IBM)		
Contributions from	John Farren (IBM), CLMS, NGS		

Disclaimer

Copyright message

Table of Contents

List of Figures

List of Tables

Glossary of terms and abbreviations used

Abbreviation / Term	Description
API	Application Program Interface
AS	Autonomous System
AWS	Amazon Web Services
DBaaS	Database as a Service
DNS	Domain Name System
DoW	Description of Work
EC2	Elastic Compute 2 {Amazon Service}
ELK	Elasticsearch Logstash Kibana
ESB	Enterprise Service Bus
GA	Grant Agreement
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
IoT	Internet of Things
LAN	Local Area Network
LL	Living Lab
NFV	Network Function Virtualisation
NOLI	New Open Logistics Interconnection
OLI	Open Logistics Interconnection
OSI	Open Systems Interconnection

SB	StockBooking
SCN	Scenario
SDN	Software Defined Networking
SLA	Service Level Agreement
SoA	State of the Art
T&L	Transport & Logistics
TCP/IP	Transmission Control Protocol/Internet Protocol
vCPU	Virtual Central Processing Unit
VLAN	Virtual Local Area Network

1 Executive Summary

This document discusses the work performed in **C** **C** (previously known as Control and Management Platform) for deliverable **C**

The document demonstrates the significant and critical work done under Task 2.7 across four key disciplines:

- Leadership and Design Methodology
- Design Activities & Outputs
- Technical Activities Supporting Deployment and Integration
- Simulation activities.

This document starts by describing the leadership approach and strategy that IBM have applied both for the work in this task and for WP2. The document will go into detail regarding the workshops and meetings that IBM have led, the achievements and outputs from them and how they have fed into the goals of WP2.

The document will further describe these outputs as they relate to the architecture goals of the project. In particular the document will describe the agreement and development of a pre-cursor to a decentralised peer-to-peer based architecture, the benefits and implications of this design and the next steps that followed.

The next steps resulted in a detailed sequence outline that shows the top-level service relationships and interdependencies. This in turn has led to deeper insights regarding service functionality, inputs, outputs and API templates.

Based on the insights derived above, technical configuration and preparation has been carried out in the Proof of Concept Integration Platform, the details of which are discussed in detail. Particular focus is placed on how this work will support the service deployments and requirements.

With an overview of the PoC platform itself complete, the document proceeds to describe the practical PI Service deployments that have taken place, how and using what configurations and technologies.

There is an overview of the initial simulation and PI Service integration efforts and a description of what the next steps are for Task 2.7.

The conclusion summarises the main outputs, successes and overall value of the work described in this deliverable, in particular how the work will move the PI concept closer to realisation and adoption by major logistics stakeholders by providing a robust, dynamic and flexible platform in which PI Services can be safely and securely developed, integrated internally, integrated externally and evaluated in their application to the Living Lab use cases. Furthermore, the inherent design of the PoC platform facilitates and encourages multiple parallel workstreams and an iterative design and development process which will bring value to the PI technical journey.

2 Introduction

This deliverable describes the methodology, design decisions and work done under **C** previously known as **C** The PoC Integration Platform task and associated work has two main goals. The first is to drive the integration of these assets across a number of scenarios. The second is to support and facilitate the development and deployment of PI Services and other technical assets required to further research in the PI domain. The first goal has been accomplished through leadership and design strategies employed in collaboration with WP2 participants and Living Lab participants in WP3 and the second goal has been accomplished by a purely technical approach. The approach and activities taken to address the first goal are described in the section and the second goal is addressed in the **C** , **A** and **A** sections.

2.1! Deliverable Overview

- **C** : A brief outline of this deliverable
- **C** : Relationship of Document to DoA, Goals of the DoA and this document, methodology and leadership strategies employed, summary of effort between V1 and V2 of this deliverable.
- **C** : Discussion of the approach to driving integration, the outputs of that approach and the benefits of those outputs to the goals of this task and WP2.
- **C** **C** : Discussion of the goals of the PoC, the network configuration and topology, security configuration and strategies and services to support the WP2 activities
- **C** **A** : An overview of assets deployed so far, what frameworks were used and how
- **C** **A** : A discussion on integration between PI Services and Simulation services, approach and practical steps taken
- **C** **C** : An overview of lessons learned and next steps in this task

2.2! Methodology & Leadership Approach

IBM have a number of responsibilities within the ICONET project in addition to the defined tasks and deliverables described within the description of work. IBM is the WP2 leader, ICT leader and proof of concept (PoC) platform integration leader for the project. The PoC integration platform is the meeting place where all technical work efforts within WP2 will be realised in term of deployment, integration, development, simulation and testing. This allows for synergy in terms of providing an overall strategy, workplan and direction for WP2 efforts and also allows for progress monitoring as all constituent components such as PI services, IoT and simulation are integrated and harmonised on the PoC Integration Platform. It should be noted that the intent in this effort is not to define architecture per say, but rather to drive tangible development work, component implementation and experimental research through iterative complex integration recipes. The insights derived, decisions made, and strategies chosen, both on an individual level by the various WP2 partners and on an overall WP2 level as driven by IBM and CLMS, continuously feed directly into the final PI reference architecture objective.

In order to drive the above described process IBM leads a weekly teleconference call to facilitate technical discussions, identify key goals and to encourage and guide forward progress against the objectives of WP2. In addition to the weekly meetings, IBM has also hosted and led a number of technical workshops to add impetus to specific and major milestones to accelerate progress for WP2 overall. These workshops have defined agendas and are objectives driven. Attendee contributions are prepared in advance and the workshops are interactive and informal in their execution to make them more conducive to meaningful discussion and collaboration. This facilitates open discussion and critical debate which ultimately leads to decisions based on consensus. Due to the innovative nature of PI, finding a starting point can be difficult as there is limited practical state of the art to refer to. To jump start discussions during these workshops a successful strategy that IBM employed has been to

start from a point that is known to be not optimal and challenge participants step in and provide their ideas and thoughts on the reality of the situation. This has led to healthy technical discussions and consensus on how to progress from the initial starting point to a more developed and productive stage. Many of the decisions and outputs described in this document were achieved through such discussions.

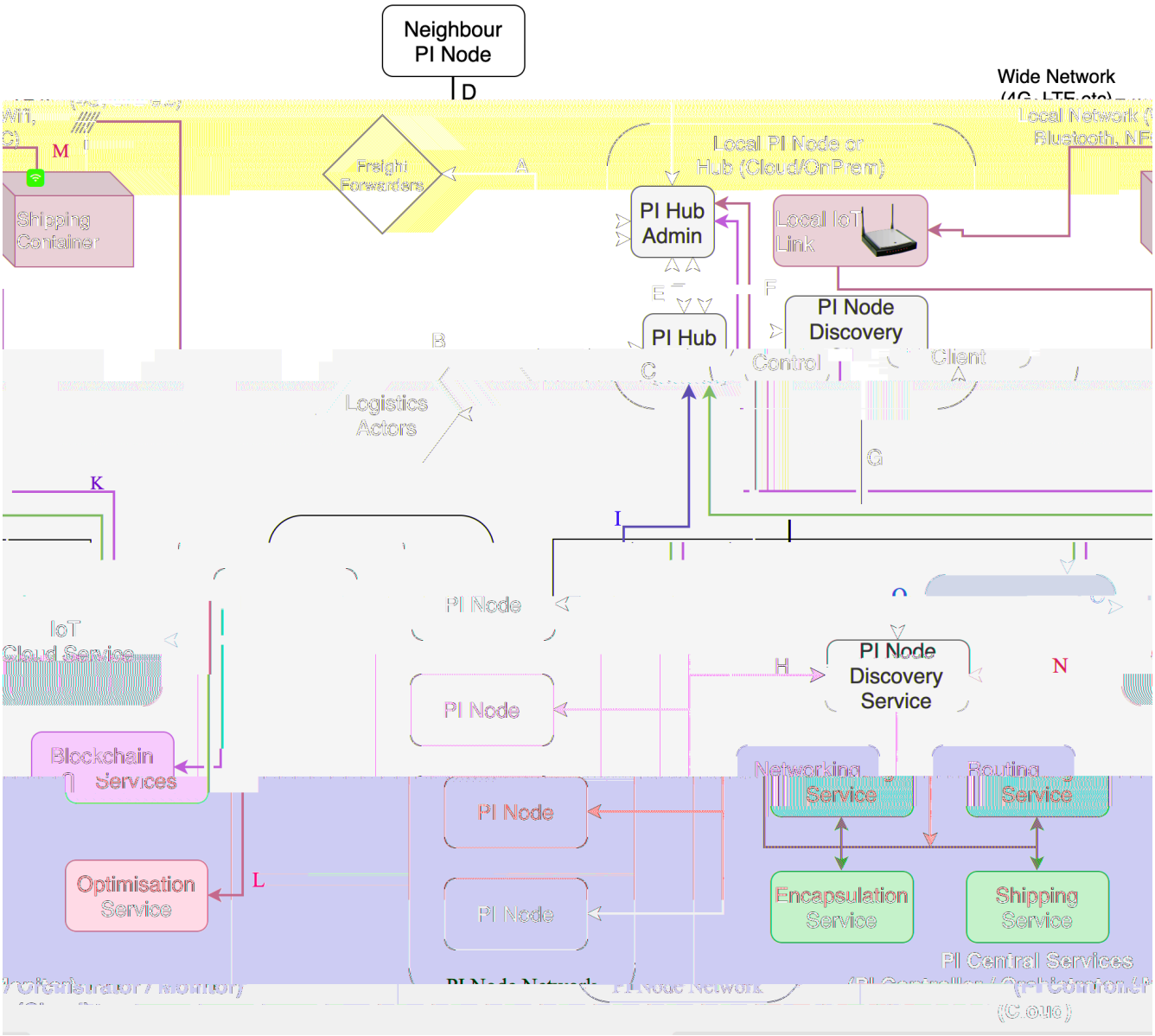
2.3! Goals of the PoC Integration Platform Task & Deliverable

The goals of the ICONET PoC integration platform were discussed in detail in **C** , the precursor to this document. However, they will be briefly restated here for convenience. The purpose of the work in **C** is to provide a technical testbed or "work-bench" that allows technical assets developed by WP2 partners to be deployed and tested in a single location that will aid the research and development of PI assets and IoT elements in a PI simulated environment. Beyond that, and once single-system development and testing is complete, the platform will facilitate multiple integrations between the various components that are deployed within. This will allow for multiple variations and integrations that will test a variety of theories, strategies and designs which will ultimately feed into the PI reference architecture and actual deployment recipes and technical specifications. In addition to these goals are the use cases provided by the Living Labs that need to be reflected and related to in the PoC platform environment. The

3 Integration Strategy

3.1! Architecture

As stated previously in this document, the purpose of the PoC integration platform is not to define a PI architecture in or off itself, but rather to facilitate the evolution of a new PI reference architecture by ensuring technical collaboration and experimentation which will lead to insights that in turn feed into the development of a new PI architecture based on the activities in ICONET. That said, there does need to be a starting point, otherwise it is difficult to define specific steps and goals that cover all PI assets and services within WP2. For that reason, a recent WP2 technical workshop in Antwerp on 2nd of December 2019, devoted significant time to defining an "integration diagram". There are a lot of similarities that can be drawn between this and a final reference architecture, but the integration diagram should be viewed as a proposal, and the architecture would be the agreement of that proposal after adjustments are made and detail added. To initiate discussions IBM proposed a centralised approach as shown in *Figure 1.*



The key concept in this diagram is that the more complex PI Services would be hosted as a cloud-based services with a flexible model. This service could be a standard commercial SaaS offering or it could be a shared resource between common stakeholders – for example the Port of Antwerp could provide access in a similar manner to their nxtPort [1] system. The proposed idea is that local logistics operators within a PI Node or PI Hub would deploy local PI services to manage the interface between their IT systems and operations and the PI systems. PI Hub Admin would handle incoming orders from neighbouring nodes as well as the creation of new orders by Freight Forwarders or other entities. PI Hub Control would feed these orders and any other relevant info to the various PI services in the cloud, which would return a set of instructions and steps to be carried out (i.e. Load Container X onto Truck Y and send Truck Y to Node Z").

With relation to the IoT systems and services, the requirements at the local node or hub were limited – simply provide (if available) the network infrastructure that allows IoT devices to connect to the internet and stream data to the central service. The local hub or node did not need to see, access or process this info in the same manner by which the telecom operator does not see the traffic from a mobile phone.

While this proposed integration approach had some merits, it was universally agreed by WP2 participants to be not viable. This approach was evaluated against both the LL2 (PI Corridor with Proctor and Gamble) and Generic

frameworks by building and adding custom implementations (that may be based on open source generic versions) of some or all of the services within the PI service stack. This serves to ensure that logistics actors can define their own level of engagement with the PI, but it does increase the reliance on a common PI reference and data model to ensure that all stakeholders undergoing this process will ultimately be able to interface across this PI concept. This follows the same adoption approach of the digital internet which was simply "In order to join this network, you must adopt this protocol, at least at the point of connection".

The peer-to-peer approach also has some implications regarding the PI Network, adjacent node discovery and for high level network visibility. There will no longer be a central "source of truth" regarding the PI network. Therefore, each node or hub is responsible for establishing agreements and connections with its neighbour nodes and hubs or hubs and nodes from neighbour regions. This information can then be accessed via the networking service by neighbouring nodes and ultimately will be used by the routing service to calculate an optimal route to, from or via a node.

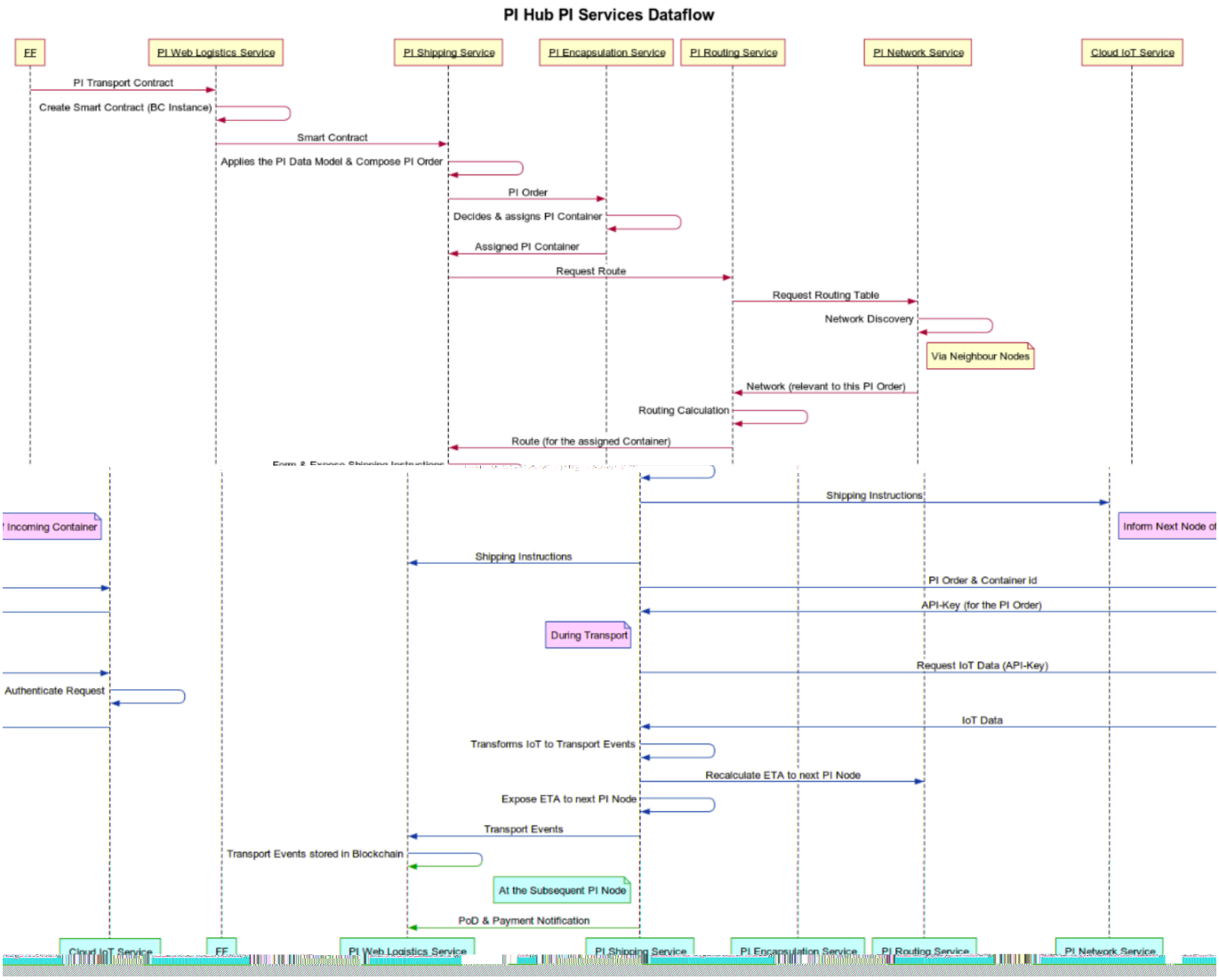
This approach has some direct implications with regards to how the PoC Integration Platform is designed and configured. For the originally proposed approach (centralised management), there would be a requirement for a separate Virtual Private Cloud (VPC) (discussed in detail in the **C** section) to host the centralised management and services and then multiple VPCs configured to connect to this central location representing many nodes. The complexity of the services in the centralised management deployment, combined with the large number of use cases it would be required to address, would require a much more complex development framework allowing for multiple workstreams in parallel and rapid deployment of new versions. A single centralised management entity also requires all services and assets to be highly integrated as a group at a very early stage.

With the chosen decentralised approach integration can progress in smaller increments. For example, verifying integration between two services before moving on to adding more. In addition, there is no longer a need for a large central VPC, and many satellite VPCs but rather simply a place to verify the integration that is currently being tested. In short WP2 moved from a large complex framework to a smaller simpler one and this benefits not only the PI concept and architecture overall but also the development effort itself.

3.2! PI Services: Dataflow and Interconnectivity Scenarios

With a comprehensive and overarching integration approach defined, the next steps were to define the integration points for the specific PI services. This was done via discussion and debate among WP2 members at the workshop in Antwerp. Again, the discussion was driven by the GPICS and LL2 use cases to gain an understanding of the practical steps and actions to be taken at each step in the PI stack. The OLI model was also used to prompt potential relationships and interactions.

The first steps were to add detail to the methodology and diagram described in the **A** section. The next logical step was to develop a sequence diagram to add in critical concepts such as sequence and order of connection. The approach chosen by IBM and ILS was to define a sequence diagram that outlined the broad steps for moving a container through the PI as described by the LL2 and GPICS use cases. The sequence diagram can be seen in *Figure 3* and it should be noted that the diagram has been updated on numerous instances as new insights, designs and ideas are incorporated.



Since *Figure 3* above is an overview of the end-to-end process of moving a PI container through a PI node, there are a significant number of steps within. To aid following the diagram, there is a full outline text sequence of this diagram located in **A** There is also a larger version of this diagram **A** for readability. A summary of the activity described, from a PI Service perspective is as follows:

1. PI Web Logistics Service
    a. Accepts incoming Orders
    b. Creates Smart Contracts
    c. Passes Smart Contract to Shipping Service
    d. Receives Shipping instructions from Shipping Service
    e. Receives Transport Events (delays, sensor triggers) from Shipping Service
    f. Stores Transport Events in blockchain
    g. Records payment notifications in blockchain
2. Shipping Service
    a. Receives Smart Contract from Web Logistics Service

   b. Creates PI order using defined Data Model
   c. Requests & Receives Encapsulation Instructions from Encapsulation Services
   d. Requests and Receives Route from Routing Service
   e. Notifies neighbour nodes of incoming transport via Network Service
   f. Use IoT Service to get an API and key for a given container
   g. Uses API & Key to track container
   h. Translates IoT Info, Route Info into transport events, sends to blockchain (Web Logistics Service)
3. PI Encapsulation Service
   a. Receives a PI order
   b. Allocates PI Containers, sends allocation to shipping service
4. Routing Service
   a. Receives request for routes
   b. Requests info about available network from Networking Service
   c. Provides Route to Shipping Service
5. Network Service
   a. Accepts requests regarding local and neighbour node networks
   b. Communicates with neighbour nodes to establish status and links to neighbour networks
   c. Returns Network Info
6. Cloud IoT Service
   a. Accepts Container ID as a request
   b. Returns unique API & key for specific container
   c. Provides access via API & key to container info & event log

The benefits of this diagram are numerous. While it does provide an excellent overview of the overall dataflow and sequence for the PI service stack it also allows each individual service to be assessed on its own merits as above. Each service has a number of clear inputs and outputs, a clear set of other services with which it will integrate and finally a succinct summary of the functionality it must perform. This sequence diagram, in combination with the data model defined by CLMS provides an excellent road map for future development of the services themselves.

Following the workshop this sequence diagram led to the initial API definitions for the services and those in turn can lead to more formalised definitions and API publication.

While *Figure 3* above is quite complex, it has spawned a number of sub-diagrams that focus on specific service integrations. Of major relevance to this deliverable is the integration of the simulation models developed by Itainnova into the sequence diagram and thus into the overall PI stack that WP2 is building. This allows for practical evaluation of the services in a “with PI vs without PI” comparison. This is discussed in more detail in the section.

4 Infrastructure to Support PoC Environment

4.1 Goals

The goal of the PoC environment is to facilitate technical development and collaboration within WP2 and also to support the work occurring between WP2 and WP3. While the PoC environment not only acts as a landing area for assets developed through the project, but as an area where multiple deployment methods and scenarios can be trialled and experimented. This applies not only to individual components and services but to the integration between multiple services. For example, the developer of a single service may wish to try deployment on a standard VM (Virtual Machine) through classical software installation means, via a docker container into the AWS ECS (Elastic Container Service), via AWS Lambda (discussed in detail D.19, Lambda allows for serverless code execution) or another model defined by the developer. Developers also have the option to experiment with various supporting frameworks, for example different database types offered by AWS, block versus file storage etc.

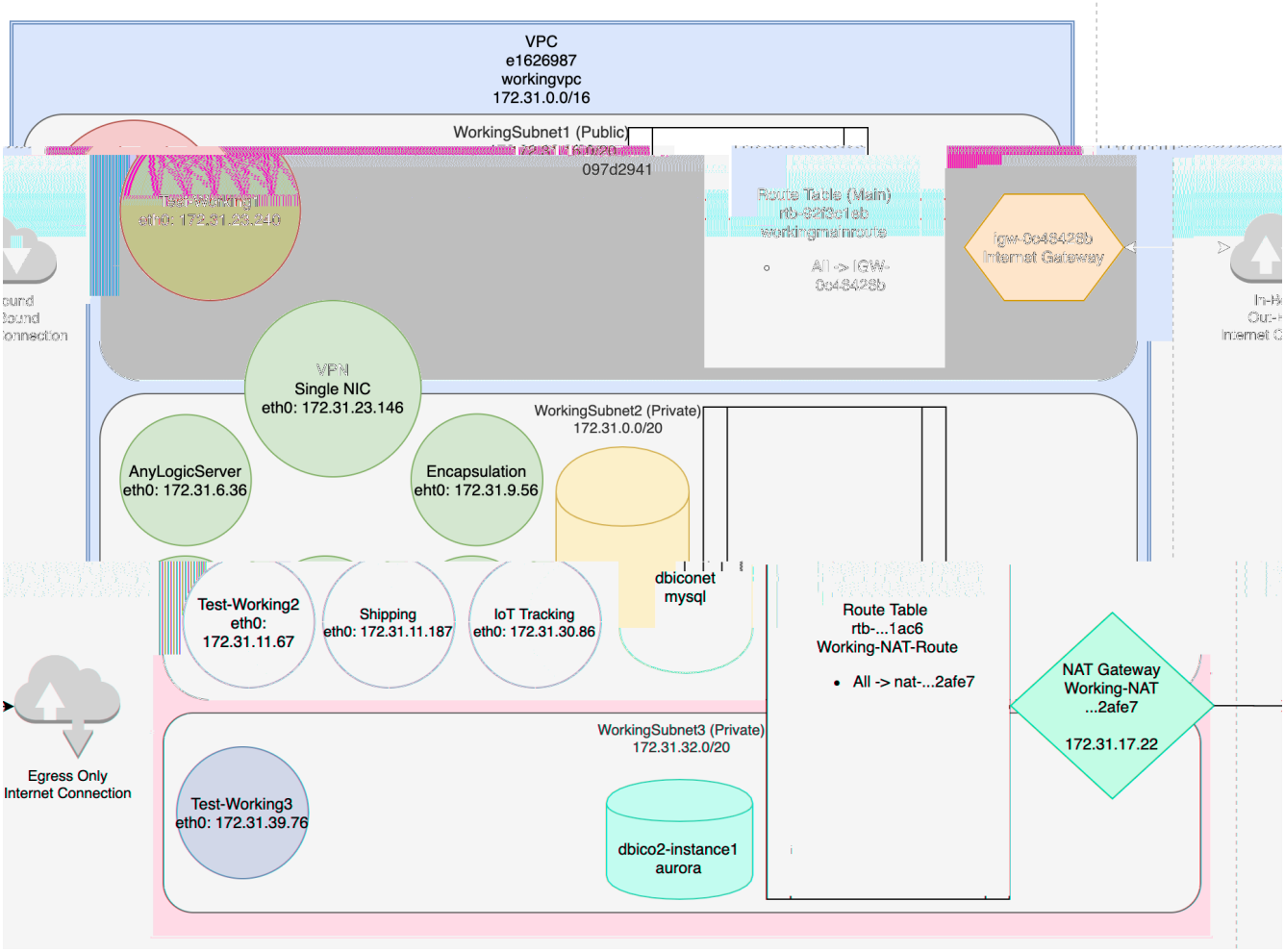
From a multi-component integration perspective, the PoC allows for multiple integration scenarios to be facilitated. For example, integration components on the same local network versus (simulated) connections through the internet.

The requirements gather process for the PoC environment approach involved a number of cross collaborations. The general approach at a high level is that the Living Labs provide their requirements to the service owners and then the service owners derive a technology approach based on these requirements. The service owners will then pass their requirements for that technology approach to IBM. However, in practice the process was significantly more collaborative with Living Lab representatives, service developers and IBM representatives discussing the overall approach in a unified manner. While some Living Labs had functional requirements such as IoT data stream access on a public network for P&G (discussed in the **C C** section), all had non-functional requirements such as security and data protection. With the potential for sensitive data to be shared in this platform, there was a strong requirement for the PoC to be secure (discussed in the section).

The final output of the PoC should be a number of integrations, deployment and usability experiments that provide insight and education that can be streamed into the T2.1 Architecture task.

4.2 Virtual Private Cloud Network Security & Topology

AWS allows for the deployment of Virtual Private Clouds (VPCs) [2]. These are essentially private networks that can be configured as required and into which a variety of networking components and configurations can be deployed. It is possible to have many VPCs in a single AWS account and although they are secure and separate networks it is possible to facilitate inter-VPC network communication through appropriate configuration. *Figure 4* below shows the internal layout of the first VPC that WP2 is using for asset deployment. There are other VPCs created within AWS for WP2 and they each have specific goals and configurations. The first VPC is noteworthy as it adopts a broad range possible configurations, assets and frameworks and is thus a useful example to discuss. The VPC provides a broad subnet of IP addresses which give a range of approximately 65,000 IP addresses available for use.



A

For network security purposes this was further divided into three subnets – one public subnet and two private subnets. Each subnet has approximately 4,000 IP addresses available in its range. A public subnet is so defined by its ability to connect to the internet and receive incoming connections from the internet. This is configured by creating an Internet Gateway (provided by AWS) and adding the gateway to the route table for the public subnet. In addition, a public IP address is assigned to a VM within the public subnet. In this instance an AWS elastic IP address has been assigned.

The two private subnets are configured to use a NAT gateway. The gateway is created and also assigned an elastic public IP and the internal address is added to the private subnet route tables. A NAT gateway allows for network traffic to leave (i.e. for any VMs or services within to access the internet) but does not allow any external traffic inbound.

The purpose of creating both private and public subnets is to provide strong security hardening by default. All WP2 developed assets will be placed in private subnets so no external attacks will have a route to these assets. In order to facilitate developer access, as VPN (Virtual Private Network) has been configured. The VPN service is installed on a VM that resides in both the public and private subnets (using multiple network adapters). Using a VPN client installed on their local laptops or desktops, developers create a secure connection to the VPN server, which in turn bridges their local network with the private subnets in AWS. This allows developers to interact with their own and other services deployed within those subnets.

The VPN server is running OpenVPN server and has been configured with multiple unique userIDs for each WP2 participant. Multiple users can connect simultaneously. Figure 5 shows the VPN Client & Server status interfaces.

4.3 Security Configuration Strategies

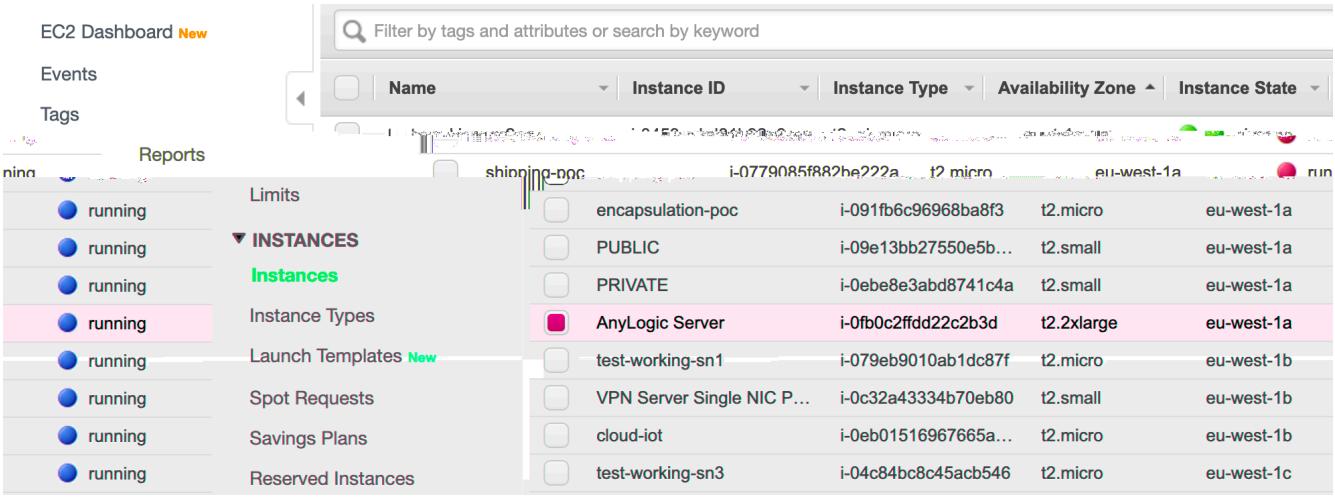
AWS provides a number of services that complement the network security design described above [3]. The AWS IAM (Identity and Access Management) Service allows for the creation of multiple users and multiple groups. In the case of ICONET, each technical partner organisation (such as IBM) has an associated group and then users in that organisation are added to that group.

4.4 Compute

The compute services are the major resource engaged by WP2 for ICONET. There are three main approaches to utilizing the compute resources in AWS that were evaluated and utilized.

4.4.1 Virtual Machines

A virtual machine (VM) is simple a virtualised instance of a traditional hardware-based server. AWS offers a wide variety of VM template in its EC2 (Elastic Compute 2) service [4]. and configurations to suit a large number of scenarios. AWS denotes categorises VMs by the hardware resources that will be allocated to a VM. For example, “t2.small” instance allocates 1 vCPU (Virtual CPU), 2GB of RAM, shared storage resources, and low network performance. Whereas, a “m5ad.2xlarge” instance allocated 8 vCPU, 32GB of RAM, a dedicated SSD, and up to 5 Gigabit network speeds. The more resources, the more cost associated with the instance. Therefore large, resource heavy VMs are only engaged when strictly necessary. Aside from the resources allocated, there are a number of choices available for operating systems that come pre-installed and configured. For ICONET Ubuntu 18.04 Server edition has been chosen as a default image.
At deployment time, a number of configuration choices must be made. Of note, is to ensure the VM is deployed to the correct VPC and correct subnet (See) and that an appropriate security group is assigned to the VM (See) Once created access to the VM is performed via SSH connection using a private key already configured.

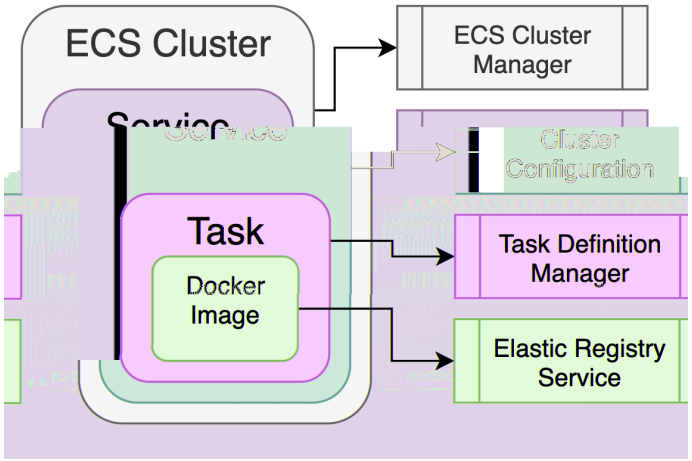


From an application and service perspective, software is deployed into a VM manually by the developer or engineer using whatever manner or means they have chosen for their component. For example, the shipping service is developed using node.js, which was manually installed.
AWS also provides AMIs (Amazon Machine Images) which are VM instances that not only come pre-configured with their operating system but also have some specific services or software pre-installed. The VPN server (See) was deployed using an AMI from the AWS marketplace. Some AMIs in the marketplace have associated licensing costs depending on the software and its licensing rules. The VPN server came with OpenVPN Access server pre-installed with a free tier license suitable for WP2 needs for now but may require a license upgrade at a later point as development intensifies.
In addition to being used for application and service deployment and the VPN a number of “micro” VMs (free cost tier) were deployed for network administration purposes. These VMs are used to test and verify the various networking routes and restrictions to both ensure the VPC is configured correctly before service VMs are deployed within and also to test and verify any configuration changes without having to interfere with VMs

running PI services. These VMs have no associated cost, so there is no limit on the number that can be created. The VMs currently deployed in the WorkingVPC (See) can be seen in *Figure 6.*

Containerisation

Containerisation is a lightweight virtualization technology that allows relatively small images to be built that contain the required operating system, libraries and tools to run a service or application, and no superfluous components. The most prevalent technology supporting this approach is Docker. AWS provides a number of approaches for supporting containerised applications. AWS ECR (Elastic Container Registry) is a simple service that provides a storage location for all container images that are available to be deployed. Any WP2 partners using the containerisation model for deployment will push their images to a dedicated ECR cluster from where it can be deployed. The ECR cluster allows for the storage of iterative versions and for experimental versions to be stored.



For deploying containers, ECS allows for entities known as “tasks” to be defined. A task specifies one or more container images that will make up your application, the image version that should be deployed, the port numbers in use, storage solutions needed and networking configurations to be applied. It is also possible to allocate resource usage restrictions and security groups, policies and roles. Other attributes such as monitoring, and logging can also be configured here.

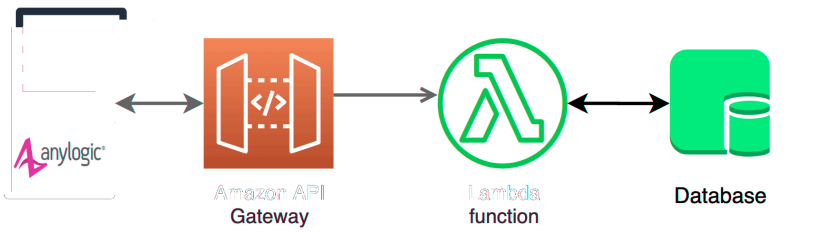
Once a task is defined it can be deployed into a cluster – a cluster is the entity responsible for running the task. It is known as a cluster as it is possible that the task definition specified some scaling rules meaning there are multiple copies of multiple container images that need to be run, and the cluster meets the hardware requirements of the task defined. The nested relationship of clusters, services tasks and docker images and their associated control mechanisms or sources are shown in *Figure 7.*

There are two cluster types – EC2 or Fargate. EC2 (as mentioned in) in the context of an ECS cluster is simply a standard VM running an ECS containerisation agent. This means it’s possible to have fine grain control over the actual VMs the container will run within. Fargate abstracts the user away from the underlying VMs. The user simply specifies their task definition and some basic info about how the cluster should be configured and AWS handles the compute behind the cluster. This gives less control but is simpler to manage and set up. The optimisation, routing and blockchain services are currently running in an ECS cluster running Fargate.

AWS does also offer Kubernetes Cluster Service, but this is unlikely to be engaged at this phase of the project as Kubernetes is designed to allow applications and services to scale to very significant size and load. This is not required as part of ICONET.

4.4.3 Lambda

AWS Lambda abstracts the user even further away from the underlying hardware or virtualisation frameworks needed to support an application. One of the core tenants of Lambda services is that they only exist for the exact moment they are needed. This leads to the phrase “Serverless Execution” which is often associated with Lambda. The general approach is that a Lambda Function is defined as pure code. This is the core execution and functionality of your task. This model is known as “Faas” or Function-as-a-Service. This suits Research and Development actions as it allows developers and researchers to focus on the core problem, and not have to spend time on hardware or orchestration configuration and setup. In the case of Lambda, It also allows easy integration with other services provided by AWS such as storage or network resources. This also eliminates the need for custom configuration or setup and also places the responsibility for issues like security or patching onto the cloud provider. FaaS is also well suited to microservices based designs which is a strategy that underpins the ICONET development approach. Lambda supports a number of languages such as Python, C++, Ruby etc.



The function is assigned some roles (if it needs to access other AWS resources, and a trigger which will cause the function to be run. Of interest to the ICONET project is the ability to create an API endpoint trigger. As can be seen in *Figure 8 - Lambda Example*, a sample PI service called by Anylogic can be implemented in Lambda by using an API endpoint to trigger an algorithm which may use data from a database to calculate a result and return the answer to Anylogic.

Although no services or components have yet been deployed via Lambda, it is planned to use Lambda to create the “intermediate” assets needed to successfully integrate and test the core PI services. Examples will be services to create sample PI orders based on information in a database, to generate simulated events (dropped container for example) or to simulate external nodes that need only emulate limited functionality.

4.5 Storage

There are a three major types of storage needs across the varying types of assets and components in WP2. Any VMs deployed will require storage in the form of disks in the VM. AWS allows for either dedicated SSD (Solid State Drive) or EBS (Elastic Block Storage). SSDs are for high performance, high end VMs where disk speed is a requirement and EBS is acceptable for standard VMs of the type being used in ICONET. EBS is highly scalable, reliable and redundant and is available on a pay-as-you-use model. EBS volumes appear as block storage in the VM operating system and therefore can be configured in the same manner as any standard physical drive. This allows for applications that operate on the block level to use the disk and allow user configurations such as non-standard volume formats. All VMs in the ICONET PoC environment are using EBS storage.

For Containerised applications or Lambda applications not deployed into an actual VM, but via AWS ECS only those deployed via EC2 have configurable storage options as Fargate does not support persistent storage. Therefore, containers deployed via the ECS EC2 option avail of the same storage services as EC2 itself – EBS.

The most utilized storage type in ICONET WP2 currently is the RDS (Relational Database Service). This allows for the creation and configuration of a wide range of SQL based databases. There are a number of engine options including Aurora, MySQL, PostGreSQL, MariaDB, Oracle and MS SQL Server. Living Lab 3 and Living Lab 4 (Stockbooking and Sonae) in particular are heavily data driven and may be able to supply large amounts of data to enhance the simulation services. This will require further integration effort to ensure this data is stored centrally in the PoC environment but accessible via the simulation and other services.

4.5.1 SDN and Network Overlay

The DoA indicates that these technologies should be utilized where possible. While the ICONET PoC Integration Platform does utilize these services, it does so via AWS services as opposed to a dedicated implementation. The mechanism whereby VPCs, Internet Gateways, NAT Gateways, Routing Tables and other networking frameworks are provided is though SDN and network overlay technologies. However, these are abstracted away from the WP2 developers and users. There are advantages and disadvantages to this approach. The advantage is that there is minimal configuration or management required, but the disadvantage is there is very little control or customisation possible.

It was decided that this was the optimal approach after careful consideration of the development goals of ICONET. Custom SDN and Network overlay management tools are likely to benefit the next evolution of PI research. SDN allows for large scale simulation of containers, nodes, locations, users etc. while still ensuring it can all remain in a secure closed simulation environment. ICONET is at a stage before this scale of testing is required. The current testing phases (to be addressed in version three of WP2 deliverables) will address functionality, PI protocol robustness, and inter-service communication and this is facilitated via the Anylogic Simulation suite. This allows for a reasonable scale of PI objects but does not require complex networking configurations to support it.

As mentioned above, Kubernetes as a Service is offered by AWS as a tool for managing containerised applications and this tool is a production level, industry standard tool that allows services and networks to scale massively. Kubernetes has inherent SDN and Network Overlay functionality, but it can also be heavily customised or even enhanced with additional tools. This is likely the next evolution of PI Service deployment and configuration to allow validation on an international scale.

A

To access the Anylogic Cloud service a user connects to the private IP (via VPN) on port 80 in a browser. This presents a login screen where users can authenticate. Once logged in, there is a catalogue of simulations that can be browsed and launched. Models are created and configured on a local client and pushed to the cloud where they can be used. While the Anylogic server does reside inside the VPN there is a requirement to make the simulations available outside the scope of WP2 participants. To facilitate this an external IP address is associated with the Anylogic server with firewall rules allowing only access via port 80 and 443 for http and https traffic. User accounts are managed by ITAINNOVA as they are the administrators of the Anylogic System itself.

5.2! PI Services

5.2.1 ECS Fargate: Optimisation, Routing & Blockchain

The Optimisation service has been built in PYTHON using FLASK as the underlying web frameworks. The source code for the service comes as a directory structure with different folders for the docker, web and algorithmic elements. The docker elements reside in the root directory and consist of a Dockerfile which describes how to build the docker image, what dependencies will be needed, where the core application resides, and how to launch the software that is built within. A requirements' file also specifies some python libraries needed for the application to run. The application is divided into the web elements which are found under "API" and the algorithmic elements found under "PoA". This source code directory structure is saved in a github repository. To build (currently) the user clones the repository, to a local destination and runs the docker "build" command using the appropriate labels and the AWS CLI (Command Line Interface) tool.

There is an AWS ECR (Elastic Container Registry) repository that has been configured for the Optimisation Service located at The newly labelled and built docker image is then pushed to this address thereby placing the docker image inside the AWS PoC for further use. The repository has automatic version control and the most recent version can always be found at

optimisation service is iconet-opt:1 referring to version 1 of the task definition. It is set to use VPC networking, 0.25 CPU and 0.5GB RAM as a baseline of dedicated resources, maps port 5000 for access to the web API, and the latest docker image.

The next phase involves running the task that has been just defined in a cluster. The basic purpose of the cluster is to put compute and other resources to the task that has been defined, however there are additional attributes worth considering. While it is possible to create a cluster and simply have it run a task, it is far better to define a service within the cluster and have the service run the task. Clusters do not have any significant configuration; they are more of a logical way to divide up workloads and applications. A service gives further control over how the task is run within the cluster. It allows the user to choose a task to run, refine the launch type (ECS vs FARGATE, see), configure a replication strategy, boundaries on resources to limit cost, deployment strategies to avoid downtime on upgrades, load balancing, specific networking configurations, autoscaling policies and security groups.

For the optimisation PI service, a cluster named was created and inside that a FARGATE serviced called was created. The services were placed into the WorkingVPC along with all other assets, connected to both private subnets. A security group was applied allowing access to port 5000 and finally the previously defined task was attached. In this instance there is not a need for scaling or replication rules as there will not be significant user or test load at this time. The application became live at and the application logs can be accessed from the AWS console. Functionality was verified by connecting a local laptop to the PoC with a VPN, accessing the URL via a browser and observing the received request in the application logs.
The blockchain and routing services were developed and deployed in the same manner. The applications run on python and flask and is deployed as a docker image. There is a repository to hold iterative versions of the image and these are accessed by a task definition, which runs in a service, which runs in a cluster. The blockchain service is available at and the routing service is available at

5.2.2 EC2 Virtual Machines: IoT, Shipping & Encapsulation

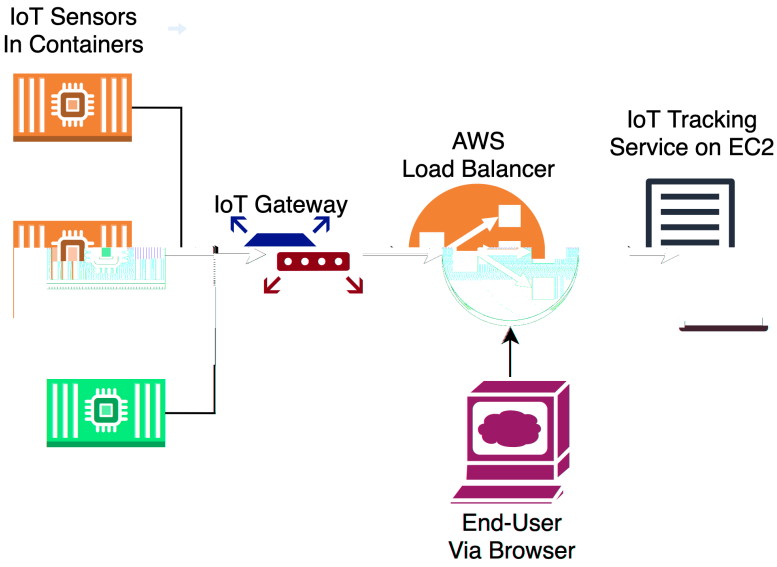
The Shipping, Encapsulation and IoT Services are all deployed on EC2 Virtual Machines with the configurations shown in *Table 1.*

• T • CPU • RAM • OS A L • D E • VPC VPC • S P • P IP • P IP	• T • CPU • RAM • OS A L • D E • VPC VPC • S P • P IP • P IP	• T • CPU • RAM • OS U • D E • VPC VPC • S P • P IP • P IP

While the IoT service is containerized, it is running in a classic VM running docker for development purposes. The Shipping and Encapsulation services are not containerized at this time but through consultation with CLMS it has been agreed that they will be containerized in the future. Ultimately, it is planned that these three services will migrate to ECS in the next phase of development work.

5.3! Connectivity: IoT Container Tracking Service

At this phase of development, the majority of the PI services in WP2 need to connect internally with each other and with the simulation services but have limited requirements to connect externally. However, the IoT communication systems do have a requirement for external communication. The IoT tracking service is in trial with P&G from Living Lab 2, therefore there is a requirement for both communication between IoT devices via routers and gateways to the cloud services and for users to interact with the cloud services directly.



As discussed in the IoT service is a Dockerised service deployed on an EC2 VM in a private VPN inside a VPC. This means that the IoT service can access the internet, but there is no means for direct access from external sources such as IoT gateways or end-users. To address this restriction an AWS Load Balancer was configured. This was accomplished by deploying the Load Balancer to the VPC that also contains the IoT Service. Then two subnets were added (1 public, 1 private where the IoT Service resides) and a security group was configured to allow access on the relevant ports from external sources. Finally, the EC2 instance where the IoT Service is deployed is registered with the load balancer. The load balancer also has a built-in health check that can verify responses from the server at specified intervals and create a notification to the administrator if the service is not functioning correctly. As shown in *Figure 10* - IoT Connectivity Scenario traffic from IoT sensors in P&G containers is routed via the gateway to the load balancer and from there to the EC2 service that hosts the IoT service. End-Users will also connect via the same load balancer on a separate port.

6 Development Asset Integrations

6.1! Simulation Integration Approach

As covered before, one of the core goals of this deliverable is to facilitate and drive the integration of the individual development assets, services and algorithms. One integration milestone that has always been considered of key importance is the integration of the simulation service with each of the other services and assets. Therefore, an integration strategy applying specifically to simulation has been devised. The approach is to pick a “pilot integration” as a means to identifying insights, approaches, problems and their solutions to the simulation integration process. Due to IBMs key role across the activities and ownership of the Optimisation service it was decided among WP2 members that the Optimisation service would be the pilot integration.

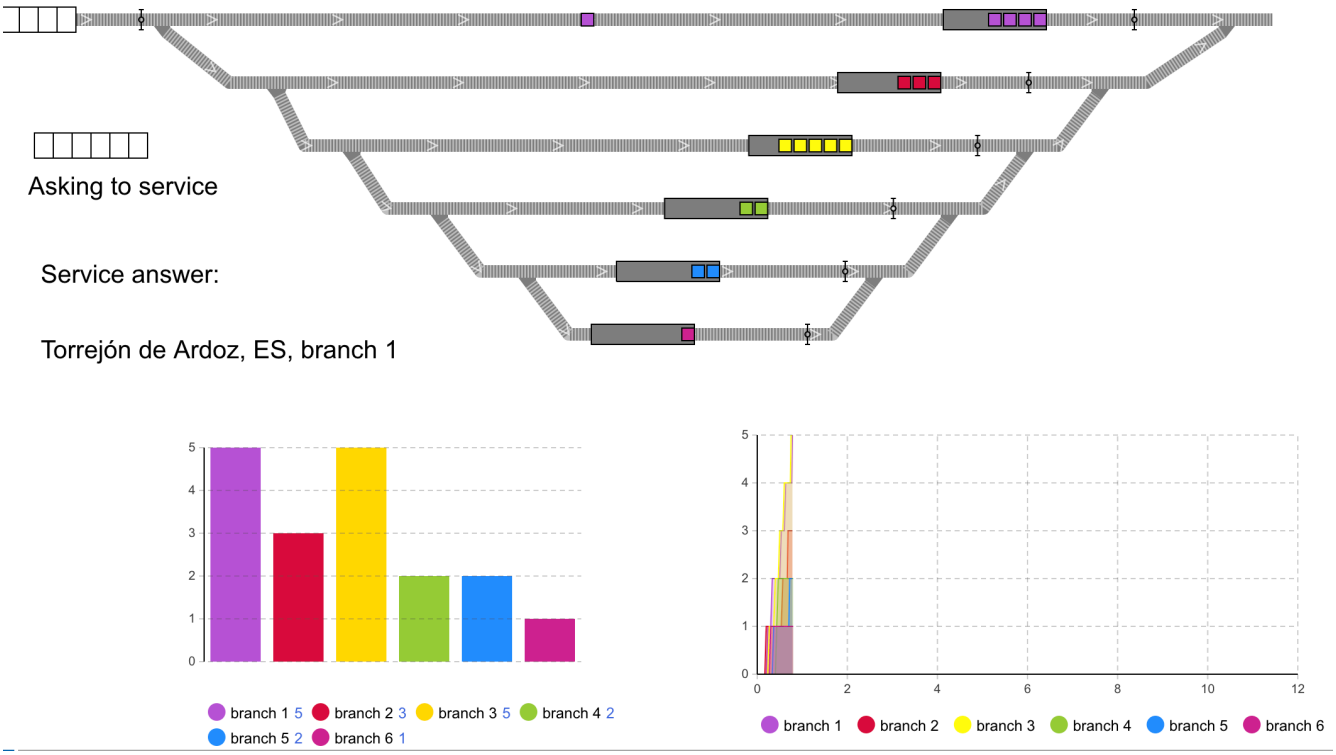
Following the Pilot Integration effort, the remaining services will be integrated with simulation services based on the approaches and insights gathered from the Pilot effort.

Finally, once all services are integrated with simulation on a one-to-one basis, WP2 partners will proceed to complex integrations involving multiple services and multiple simulations as required to advance the GPICs use case and specific Living Lab use cases.

6.2! Pilot Integration: Simulation and Optimisation Service

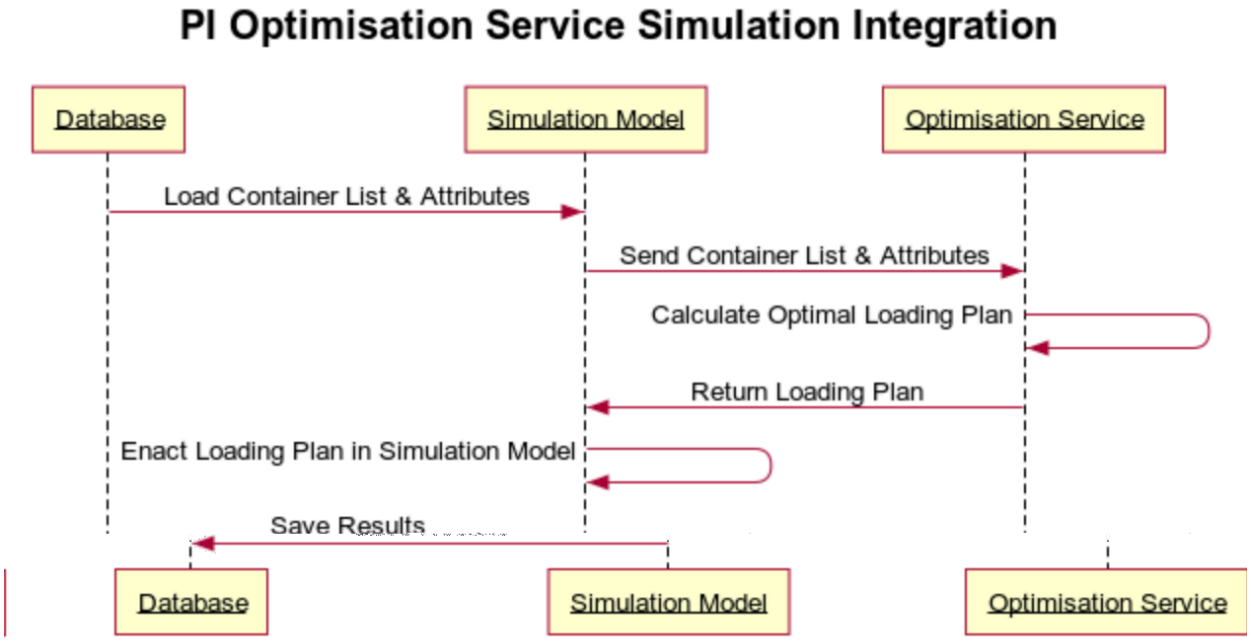
The Optimisation service is covered in detail in **C**

In very brief summary, the service works to optimize the loading of containers onto train wagons in the Port of Antwerp, and then the combination of different train wagons into train sets. The optimisation is based on container size, weight and destination with the Port of Antwerp. The service itself works by accepting a list of containers as input and then returning a loading plan as output.



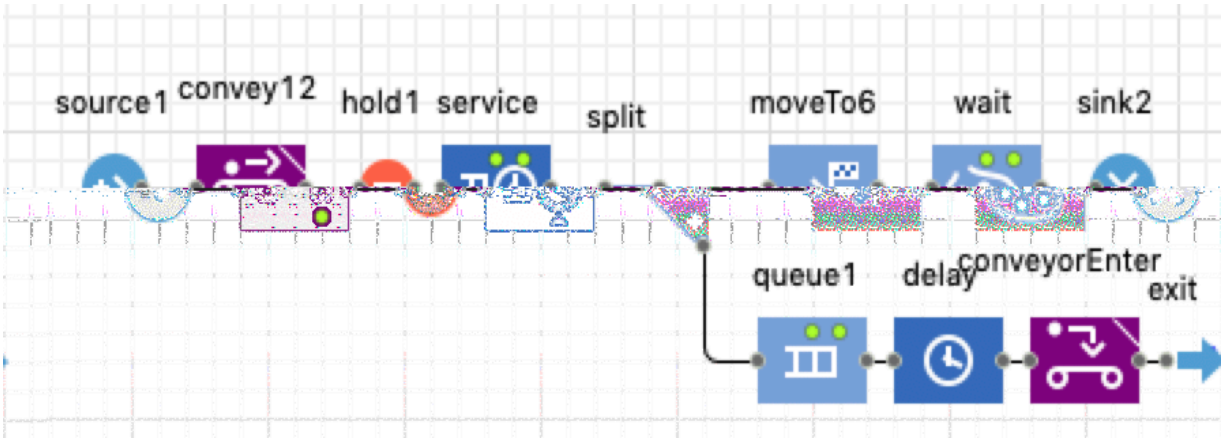
A

Shown in *Figure 11* is the model that ITAINNOVA have developed in Anylogic that simulates the shunting yard operations as defined by the Port of Antwerp use case. The simulation models the layout of the shunting yard from a physical perspective (number of tracks, how they are connected etc.), the movement of containers onto trains and the movement of trains themselves. The grey lines represent the train tracks themselves and the layout is similar in design (although at a smaller scale) to the North Shunting Yard in the Port of Antwerp. In this yard, incoming trains from outside the port arrive and the individual wagons (trains cars) are disconnected and queued on the left. The individual tracks in the centre each hold a train destined for a specific terminal in the port. Individual wagons are then moved onto a track that holds a train going to the destination of that wagon. The wagons are represented by coloured blocks and they are placed into a "train" (the larger grey rectangle). The colour represents the destination within the port – for example purple could represent all wagons destined for the deep-sea terminal. The graphs at the bottom summarise total wagons per destination and wagons processed over time.



As shown in *Figure 12*, The simulation will ingest data (in spreadsheet or database format) that defines the behaviour of the simulation such as how many containers, how often, and for what destination and then it will simulate behaviour based on these actions. Initial versions of the models use either simple decision-making strategies for how to load the containers, or just load them manually but subsequent versions make use of the Optimisation Service to define the loading order. The simulation sends the container list to the optimisation service, the optimisation service returns a loading plan and the simulation enacts this loading plan. Simulations using the Optimisation service will be compared against random or simple loading plans to quantify impact.

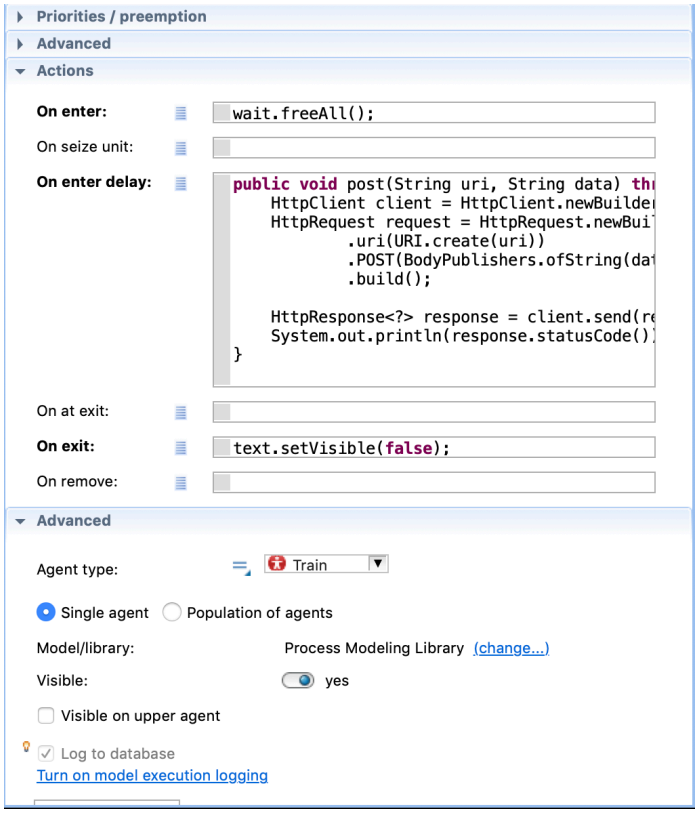
From an integration perspective, the three elements of concern are how the service is made available to the Anylogic system, how the Anylogic system communicates with the service and the common network infrastructure that supports this communication. The Anylogic models are complex and are covered in more detail in but for demonstrative purposes a simplified partial view of the shunting yard model design is shown in *Figure 13*.



A A

The Anylogic simulations are represented both visually and in code. The visual element allows simple diagrams that describe the interaction of the internal components of the simulation. This is shown in Figure 13, where we can see a source of objects (in this case containers) on the left and then a manner in which they are conveyed to a holding area, followed by the hold action itself. Following this is “service”. This is the element that makes the API request to the optimization service to gain info on what steps to take next. Once the info is acquired from the service, a decision is made, and the following steps can proceed. In this case the next step is to queue the container in an existing line so one must be chosen as recommended by the service.

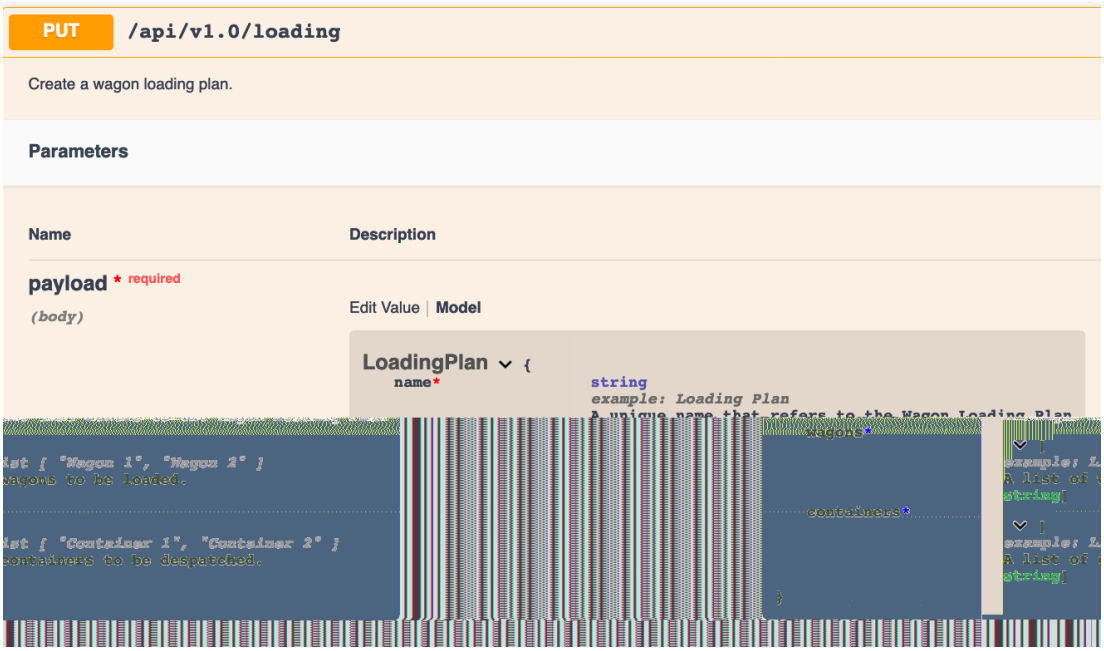
The “service” element is configured in the Anylogic development tool and has a number of standard configuration options, however the most significant is the ability to run java code on three key triggers: On Enter, On Enter Delay and On Exit, as shown in *Figure 14*.



A

On Enter allows custom java code that will pause the simulation while the service API call is completed. On Enter Delay allows for custom java code that actually makes the API call and retrieves the result. On Exit can restart simulation now that the call is complete. The code to make the service API call can utilize any Java http client library that allows for synchronous or asynchronous HTTP call and response.

A



A

For the Optimisation Service to be used by the simulation it must be available on a standard REST interface. The Optimisation Service was developed with the Swagger plugin which provides technical specs for the REST API itself. An example of the POST operation for creating a new loading plan and data format required are included in Figure 15 and Figure 16. To facilitate the interaction between these two entities, both are placed within the same VPC and on the same private subnet. Within this configuration no further firewall rules or configuration is required as security steps have been taken elsewhere to ensure no outside traffic penetrates within. Therefore, the simulation and optimisation services are free to communicate as needed.

7 Conclusions & Next Steps

The approach of IBM in WP2 and for this task and associated deliverables spans three disciplines – Leadership, Strategy and Technology.

From a leadership and strategic perspective IBM has driven the discussion and design around integration plans, pre-architecture designs, data models and service data flow. These designs are key building blocks that not only helped to define the goals of WP2, but also helped define a collaborative integration strategy and lay the groundwork for workplans for individual components and development efforts. Future efforts in this regard will take insights and lessons learned from problems already solved in order to drive the next iterations of the design specifications and PoC integration platform evolution.

From a technological perspective IBM has pro-actively sought and acquired the development requirements from WP2 partners through collaboration with WP1, WP2 and WP3 partners. These requirements have led to the creation and development of a flexible and elastic PoC environment that can support a number of deployment types at various stages of development and can support multiple integrations and experiments. In addition to meeting the functional requirements of the project the PoC environment provides safe and secure access to WP2 users while blocking any unwanted access and addressing security risks. The next steps in this area will aim to enhance the existing deployment options by creating streamlined deployment pipelines to allow developers to go straight from code check-in to deployment in a single step. Additional focus will be placed on serverless deployments using Lambda and new resources will be deployed and configured to address the increasing needs of the project.

The work described in this deliverable has direct impact on all other tasks within WP2 and has significant impact on work taking place in WP3. The PoC platform supports not only the simulation services but facilitates the design, development, deployment and integration of all PI Services developed in Task T2.2, the IoT tracking and Cloud Services developed in T2.3, the PI Node Optimisation Service developed in T2.5 and the Blockchain services developed in T2.4. Furthermore, the PoC platform serves as the deployment and “workbench” for furthering the application of these PI Services to the Living Lab Use Cases, specifically facilitating the application of the PI Node Optimisation Service to LL1 (Port of Antwerp – Infrastructure Optimisation), IoT Tracking Service to LL2 (P&G – PI Corridor), PI Routing Service to LL3 (Sonae - e-commerce) and Routing and Encapsulation Services to LL4 (Stockbooking – Warehouse-as-a-Service).

By providing a flexible and dynamic environment facilitating multiple deployments of each PI service or supporting service, the work done in D2.20 will directly benefit the further enhancement and development of the PI Services, their internal and external integrations and the PI architecture overall as captured in T2.1 all of which will serve to accelerate the PI roadmap.

8 Bibliography

[1] “nxtPort - About,” [Online]. Available: https://www.nxtport.com/about.

[2] “What is Amazon VPC,” [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html.

[3] “Internetwork Traffic Privacy in Amazon VPC,” [Online]. Available: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Security.html.

[4] “What is Amazon EC2?,” [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

Annex I: PI Services Dataflow – Sequence Text

1. FF -> PI Web Logistics Service: PI Transport Contract
2. PI Web Logistics Service -> PI Web Logistics Service: Create Smart Contract (BC Instance)
3. PI Web Logistics Service -> PI Shipping Service: Smart Contract
4. PI Shipping Service -> PI Shipping Service: Applies the PI Data Model & Compose PI Order
5. PI Shipping Service -> PI Encapsulation Service: PI Order
6. PI Encapsulation Service -> PI Encapsulation Service: Decides & assigns PI Container
7. PI Encapsulation Service -> PI Shipping Service: Assigned PI Container
8. PI Shipping Service -> PI Routing Service: Request Route
9. PI Routing Service -> PI Network Service: Request Routing Table
10. PI Network Service -> PI Network Service: Network Discovery
11. note right of PI Network Service: Via Neighbour Nodes
12. PI Network Service -> PI Routing Service: Network (relevant to this PI Order)
13. PI Routing Service -> PI Routing Service: Routing Calculation
14. PI Routing Service -> PI Shipping Service: Route (for the assigned Container)
15. PI Shipping Service -> PI Shipping Service: Form & Expose Shipping Instructions
16. PI Shipping Service -> PI Network Service: Shipping Instructions
17. note right of PI Network Service: Inform Next Node of Incoming Container
18. PI Shipping Service -> PI Web Logistics Service: Shipping Instructions
19. PI Shipping Service -> Cloud IoT Service: PI Order & Container id
20. Cloud IoT Service -> PI Shipping Service: API-Key (for the PI Order)
21. note left of PI Shipping Service: During Transport
22. PI Shipping Service -> Cloud IoT Service: Request IoT Data (API-Key)
23. Cloud IoT Service -> Cloud IoT Service: Authenticate Request
24. Cloud IoT Service -> PI Shipping Service: IoT Data
25. PI Shipping Service -> PI Shipping Service: Transforms IoT to Transport Events
26. PI Shipping Service -> PI Routing Service: Recalculate ETA to next PI Node
27. PI Shipping Service -> PI Shipping Service: Expose ETA to next PI Node
28. PI Shipping Service -> PI Web Logistics Service: Transport Events
29. PI Web Logistics Service -> PI Web Logistics Service: Transport Events stored in Blockchain
30. note left of PI Shipping Service: At the Subsequent PI Node
31. PI Shipping Service -> PI Web Logistics Service: PoD & Payment Notification

Annex II: PI Services Dataflow – Sequence Diagram