

New **ICT** infrastructure and reference architecture to support **Operations** in future PI Logistics **NET**works

D2.21 ICONET PI PoC Integration Platform – Final Version

Document Summary Information

Grant Agreement No	769119	Acronym	ICONET
Full Title	New ICT infrastructure and reference architecture to support Operations in future PI Logistics NET works		
Start Date	01/09/2018	Duration	30 months
Project URL	https://www.iconetproject.eu/		
Deliverable	D2.21 ICONET PI PoC Integration Platform – Final Version		
Work Package	WP2		
Contractual due date	M26 (AMD)	Actual submission date	M26
Nature	Other	Dissemination Level	Public
Lead Beneficiary	IBM		
Responsible Author	Kieran Flynn (IBM)		
Contributions from	John Farren (IBM), CLMS, NGS		



This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 769119.

Disclaimer

The content of the publication herein is the sole responsibility of the publishers and it does not necessarily represent the views expressed by the European Commission or its services.

While the information contained in the documents is believed to be accurate, the authors(s) or any other participant in the ICONET consortium make no warranty of any kind with regard to this material including, but not limited to the implied warranties of merchantability and fitness for a particular purpose.

Neither the ICONET Consortium nor any of its members, their officers, employees or agents shall be responsible or liable in negligence or otherwise howsoever in respect of any inaccuracy or omission herein.

Without derogating from the generality of the foregoing neither the ICONET Consortium nor any of its members, their officers, employees or agents shall be liable for any direct or indirect or consequential loss or damage caused by or arising from any information advice or inaccuracy or omission herein.

Copyright message

© ICONET Consortium, 2018-2021. This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.

Table of Contents

1	<i>Executive Summary</i>	7
2	<i>Introduction</i>	8
2.1	Deliverable Overview	8
2.2	Goals of the PoC Integration Environment Task & Deliverable	8
2.3	Summary of Version 2 (D2.20) and Overview of this Deliverable	8
2.4	Leveraging Platform-as-a-Service	9
3	<i>Integration Strategies & Approaches</i>	11
3.1	Service Discovery	11
3.2	Service Mesh	12
3.3	External Integration & Access Scenarios	13
3.3.1	Simulation Access (A).....	14
3.3.2	PI Service Access (B).....	15
3.3.3	Developer Access (C)	16
3.3.4	Data Acquisition (D)	16
3.3.5	Node-to-Node Communication & Integration with Legacy Systems	18
4	<i>Deployment Strategies, Methodologies and Procedures</i>	19
4.1	Pipelines	19
4.2	Recommended Pipelined Testing Strategies	21
4.2.1	Unit Tests	21
4.2.2	API Tests	21
4.2.3	Integration Tests	21
4.2.4	Performance Tests	21
4.3	Security Audit	21
4.4	Deployments	23
5	<i>Machine Learning Deployments</i>	26
5.1	Requirements for new Machine Learning Deployment	26
5.2	Microservices Approach	27
5.3	Model Training as a Service	27
5.4	Machine Learning as a Service	28
6	<i>Future Trends in PI Service Deployment Models</i>	30
6.1	Challenges for Multi-Cloud PI Services	30
6.2	A Centralised Multi-Cloud Management System	32
6.2.1	Public & Private Cloud APIs and CLIs	32
6.2.2	Redhat Ansible.....	32
6.2.3	Redhat Cloudforms	33
6.2.4	Multi/Hybrid Cloud Networks.....	34
6.2.5	Combining SDN, Ansible, CloudForms for a Multi-cloud PI Scenario.....	35
7	<i>PoC Integration Environment Blueprint – Full Context</i>	37

8 Conclusion 41
Annex I – Accessing the PoC Environment for Evaluation 42
Annex II –PoC Integration Environment Screenshots 43

List of Figures

Figure 1 - PaaS, SaaS & IaaS 9
 Figure 2 - AWS Cloud Map Namespaces 11
 Figure 3 - LL2 Routing Service DNS Entry 12
 Figure 4 - Service Mesh Flexible Routing..... 13
 Figure 5 - External Integrations 14
 Figure 6 - VPC links to Service Load Balancers 16
 Figure 7 - Living Lab Input Streams 17
 Figure 8 - Data Stream Monitoring 17
 Figure 9 - Living Lab Data Delivery 18
 Figure 10 - CI/CD Pipeline..... 19
 Figure 11 - Routing Pipeline 20
 Figure 12 - Vulnerability Notification 22
 Figure 13 - Vulnerability Report 22
 Figure 14 - CVE Information 23
 Figure 15 - Deployment Group..... 24
 Figure 16 - AWS Beanstalk Configuration 25
 Figure 17 - Sagemaker training job in progress..... 27
 Figure 18 - Saved Model..... 28
 Figure 19 - Machine Learning Endpoint - PaaS Powered 28
 Figure 20 - Multi Cloud PI Example 31
 Figure 21 - Redhat Cloudforms 33
 Figure 22 - Multi-cloud PI Network Scenario 35
 Figure 23 - PoC Blueprint 37

List of Tables

Table 1: PoC Integration Platform Goals 38

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
Dev-ops	Development and Operations
DNS	Domain Name System
DoA	Description of Action
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
P&G	Proctor & Gamble
PaaS	Platform as a Service
PI	Physical Internet
PoA	Port of Antwerp
PoC	Proof of Concept
QoS	Quality of Service
R&D	Research & Development
RAM	Random Access Memory

RON	Resilient Overlay Node
SB	StockBooking
SCN	Scenario
SDN	Software Defined Networking
SLA	Service Level Agreement
SoTA	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
VXLAN	Virtual Extensible Local Area Network
WP	Work Package

1 Executive Summary

IBM addressed the majority of the goals for the PoC Integration during work documented in Deliverable D2.20. This involved the creation and configuration of the virtual networks, infrastructure, deployment of PI Services and both service-to-service integration and service-to-simulation integration. In this document IBM has furthered the achievements of D2.20 by addressing the remaining goals and then enhancing the flexibility, security and robustness of the PI Service networks, communication, node-to-node communication and PI Service deployments by ensuring they are cost effective and valid in real world scenarios. The defined approach allows PI Service developers to focus on PI related functionality for logistics operations and offloads non-functional requirements (security, config etc.) to the PaaS services and dev-ops personnel. The use of PaaS systems allowed IBM to extend the scope of work beyond the DoA requirements and ensure the PoC Integration Environment and the PI Services residing within adopted real world dev-ops based production strategies and technologies to ensure PI Service deployments would be instant, automated, secure, robust etc.

IBM identified machine learning based PI Services as having a unique set of challenges and requirements and also likely to play an increasingly large role in the PI Service suite. Strategies and technologies were adapted to address the needs of ML Services while ensuring they were compatible with the existing frameworks supporting standard PI Services. Strategies around data acquisition and quality, ensuring machine learning based services are recent and up to date, that new models are validated before deployed and paradigms that see analytics outside the cloud, at the edge of PI networks.

IBM identified key emerging trends in cloud technology and deployments that are likely to both mirror trends in PI Service deployments but also impact them too. IBM researched leading technologies striving to address these trends and applied them to the existing PoC environment created in Version 2, while expanding beyond that to a likely future PI scenario that relies on multiple cloud vendors in combination with local, legacy and private cloud and software technologies used by logistics organisations. These new concepts of multi-cloud and hybrid-cloud will expand the reach of the cloud beyond the datacentre and out to edge and low power devices in the fog. As IoT becomes more prevalent in the transport and logistics sector, the ability for the PI to leverage these new technologies while adapting to future trends is critical and the research in this document lays out recommendations to take advantage of new cloud and fog computing trends. This allows for PI Services to be deployed and managed in a distributed yet orchestrated manner across heterogenous device types. All of which looks to the future and aids in increasing the adoption potential, and subsequently increasing the future disruptive potential, of PI Services.

This deliverable ends with a visualised blueprint containing all major components, services, systems and ICONET technical assets that were created, configured, deployed and developed as part of all three PoC Integration Platform deliverables. This ties together the work across all three documents and presents the work in a unified manner.

The work done and documented in this deliverable and the previous deliverable versions show that the PoC Integration Environment successfully met the requirements as laid out by the DoA, successfully met the requirements of the ICONET WP2 and Living Lab participants, and in fact went beyond the initial DoA to adapt to future requirements such as security, integration and modern deployments. It shows that a design methodology was adopted to ensure that the cloud and edge computing paradigm requirements of the future were forefront in design considerations.

2 Introduction

This deliverable describes the methodology, design decisions and work done under **Task 2.7 PoC Integration Platform** (previously known as **Control and Management Platform**). 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 substantive work towards these goals were achieved and described in D.20, the intermediate version of this deliverable. The research and effort for this deliverable focused on how the approaches already taken were refined and improved upon and how they can be applied and adapted for the PI vision with regards to current and emerging technical trends.

2.1 Deliverable Overview

- **Executive Summary** – An overview of this document’s contents
- **Introduction** – Introducing the context of this document to the previous version and the DoA requirements
- **Integration Strategies & Approaches** – Discussing the different connectivity scenarios and approaches as they were applied to the PoC integration platform and PI Nodes
- **Deployment** – Discussing the technologies and approaches used for a modern, production level strategy deployment framework
- **Machine Learning Deployments** – Discussing the unique challenges and requirements of machine learning based PI Services and the approaches used to address them and combine them with existing PI Services management frameworks.
- **Future Trends in PI Service Deployment Models** – Discussing how emerging trends in cloud deployments will apply to the PI in the near future, and how the PoC Integration created for ICONET is adapted for modern hybrid/multi cloud environments of the future, discussing specific technologies and approaches.
- **Conclusion** – Overview of work and lessons learned

2.2 Goals of the PoC Integration Environment Task & Deliverable

As discussed in Version 1 and Version 2 of this deliverable, the goals of the PoC Integration Platform are to provide a flexible, cloud enabled digital workbench which will facilitate the technical research actions in the project. The platform provides a secure enclave where PI Services can be deployed, developed, tested and integrated. As the development moves to the final stages, the focus shifts more to integration actions and ensuring that the living lab scenarios are digitally represented within the PoC environment. This spans from ensuring access to the relevant datasets to multiple deployments of single services as each has been configured for a particular scenario.

With these core goals met, IBM elected to expand the scope of the requirements and has investigated the potential usage paradigms from a cloud and deployment perspective that will likely exist by the time the PI becomes a global reality. Deployment paradigms based on hybrid cloud, multi cloud, edge and fog computing were considered within the PI context to provide a roadmap to potential real-world implementations.

2.3 Summary of Version 2 (D2.20) and Overview of this Deliverable

The intermediate (2nd) version of this deliverable (D2.20) discussed in detail the following topics

- The Integration Environment Architecture
- Dataflow and interconnectivity scenarios for the PI Services
- The goals and objectives of the PoC Environment

- The Infrastructure and Network Topology of the PoC Environment along with the deployment and configuration of compute and storage systems
- The strategies, tools and systems used to support the simulation systems
- PI Services Deployment & Integrations

The work undertaken for deliverable D2.20 addressed much of the goals and requirements of the associated tasks. The approach taken for this deliverable was to enhance the existing infrastructure and frameworks in terms of features – both functional and non-functional and to expand the scope of the research in a means which would further the PI. This was done by integrating dev-ops strategies and approaches to PI Service deployment, providing specific technological approaches for machine learning based PI Services and by addressing secure connectivity from a cloud engineering perspective and allowing PI Service developers to focus on core PI Service functionality. In addition, future trends in cloud service deployments were researched and emerging trends were applied to known PI scenarios. Emerging technologies addressing these trends were evaluated in respect of the existing PoC environment and recommendations for future PI networks were made.

2.4 Leveraging Platform-as-a-Service

Platform-as-a-Service (PaaS) is specifically addressed in the ICONET DoW as the preferred approach for engaging cloud services in the ICONET project. As shown in Figure 1, there are three major paradigms – SaaS (Software-as-a-Service), PaaS (Platform-as-a-Service) and IaaS (Infrastructure-as-a-Service). SaaS is a user facing model, example of which are gmail, slack, google docs etc. PaaS is used primarily by software developers or dev-ops to enhance or support the application they wish to present to users. IaaS is primarily used by administrators and is the underlying compute and similar resources that support the platform and applications above, which is ultimately powered by the actual bare metal hardware of servers, switches, hard disks etc.

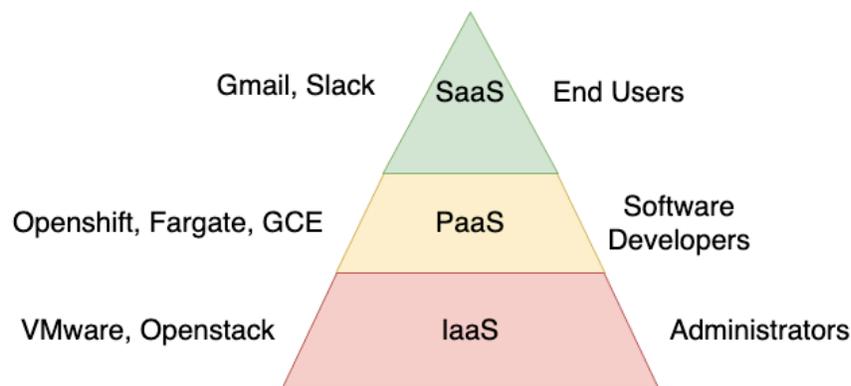


Figure 1 - PaaS, SaaS & IaaS

There are many different systems that can fall under the PaaS umbrella. These could be computed or similar systems to support applications running in the cloud such as GCE (Google Cloud Engine), Kubernetes, Red Hat Openshift or AWS Fargate but can also encompass non-functional frameworks such as security systems, firewalls, load balancers, SDN (Software Defined Networking). Anything that a software developer or dev-ops engineer can engage as a pre-existing service without having to develop, install or construct themselves can be considered a PaaS service.

The benefits of PaaS to ICONET, and why it was specified in the DoW as the approach to take, is that less time and effort needs to be spent on developing and building non-functional systems. The bulk of the effort

can be instead focussed on the core task at hand – namely the research and development of the PI Services and systems and the integration of those systems in a manner to address the Living Labs requirements. In this project Amazon Web Services (AWS) was chosen as the cloud vendor (in a selection process that also included Google Cloud Engine and IBM Cloud as candidates). AWS was chosen as it had a particularly rich catalogue of PaaS options, some specifically identified as useful for ICONET such as Lambda and the API gateway. ICONET also engaged the use of Virtual Private Clouds, Software Defined Network Infrastructure, Elastic IP Addresses, Elastic Load balancers, Elastic Container Registries, Elastic Container Services, Identity and Access Management, Deployment Pipelines and more services beyond. All of these services have commercial or open source alternatives, but a significant amount of time and effort was saved by choosing PaaS alternatives, and at a very low financial cost. This allowed ICONET to hit the ground running so to speak in terms of focusing efforts immediately and directly at the core high value development and research objectives of the project.

IBM has taken initial steps to ensure the viability of the PI Services in alternative cloud models. This is discussed in more detail in the **Future Trends in PI Service Deployment Models** section.

3 Integration Strategies & Approaches

In version 2 of this deliverable the pilot integration of the optimisation service and the simulation service were discussed. The core mechanic of the integration is the ability for the Anylogic simulation service to make REST API calls to the optimisation service.

3.1 Service Discovery

At this stage of ICONET, all services and associated APIs are now complete and published to the ECR (Elastic Container Registry) where multiple versions and iterations of each service are stored as docker image templates. Each image has been deployed and a number of services have been deployed multiple times to allow the integration and testing across various living lab scenarios. In addition, services are no longer deployed as isolated entities, as WP2 explores and configures the interdependencies of certain services on other services. For example, the routing service will make requests to the networking service and the shipping service makes requests to almost all other services. As individual services are redeployed (after an update or a bugfix) their local IP address may change. The results of this new stage of development and integration is a larger and ever-changing number of services, bound to changing IP addresses with different purposes, integrations and configurations. It is impractical to manage the publication and communication of these variations manually and in a real world environment this process needs to be managed programmatically. Therefore, for the PoC environment, Service Discovery is enabled for all services. Service Discovery is facilitated by DNS (Domain Name Servers) and configured by an internal AWS service called Cloud Map.

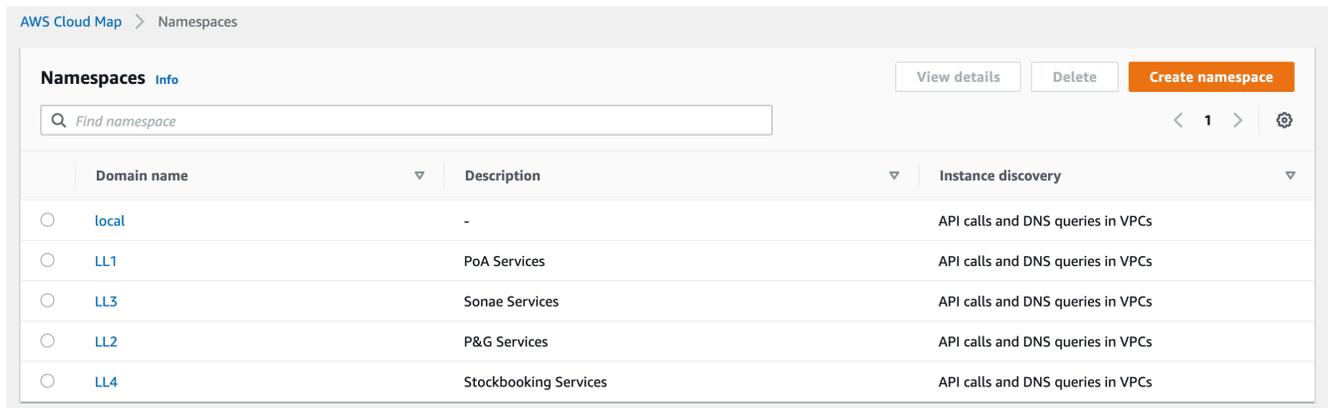


Figure 2 - AWS Cloud Map Namespaces

As shown in *Figure 2*, Cloud Map allows for the creation of namespaces and service names. A namespace is simply a designated collection of entries under a common category. In WP2 there are five namespaces, one for each living lab and an additional namespace for more broad integration and testing deployments. Within each namespace any number of service names are defined and then associated with a given service. For example, the namespace for Living Lab 2 has a service within it for routing simply called “routing”. This service is associated with the actual deployed routing service which is specifically deployed for Living Lab 2 which is labelled as “ll2-routing”. When the developers of another application wish to interact with this specific service, they need only point the service at the “routing.ll2” endpoint and the DNS servers managed by AWS will point requests at the IP address of the “ll2-routing” service.

AWS Cloud Map > Namespaces > LL2 > routing-ll2

Service: routing-ll2 [Info](#)

Service information

Name	routing-ll2	Service ID	srv-2qowxu7745bwrp4
Namespace name	LL2	Description	-
Namespace ID	ns-ug4vv5b7ik5r65zi		

DNS configuration

DNS routing policy	Record type	TTL
Multivalue answer routing	A	60

Custom health check configuration

Health check type	Custom health check configured
Failure threshold	1

Figure 3 - LL2 Routing Service DNS Entry

This process only works however if the developers already know the endpoint and know what that specific function the service at that endpoint has or does. In order to provide a programmatical way to acquire this information it is possible to associate any number of “tags” to a given endpoint. The tags can describe relevant information about the service and can include, but are not limited to:

- Port number the service is listening on
- Version number of the service
- Living Lab the service is operating for
- Dataset the service is using.

The tags are managed by Cloud Map and a developer can access these tags as well as request full listings of all namespaces and all services within specific namespaces by making API requests against the Cloud Map service itself. While Cloud Map is a proprietary service offered by AWS all major cloud vendors will have similar systems for managing Service Discovery or indeed 3rd party implementation can be used. Having developers tailor each service to be able to interact with each particular implementation is not practical so it is desirable for a common interface to be designed. In the case of the ICONET PoC, a simple API service can be developed that relays the information from Cloud Map to the service. This still needs to be updated to match the given deployment scenario, but it is a single point of configuration change, as opposed to multiple points of change that would be needed without the API relay.

3.2 Service Mesh

The next evolution of Service Discovery is the Service Mesh. This is a concept that allows developers and administrators to see and understand the relationship between all services and view them as a single entity in terms of logs, metrics, connectivity, troubleshooting etc. Of particular relevance to the ICONET project is the ability to divide requests to a single service into multiple lanes and send those lanes to specific service instances.

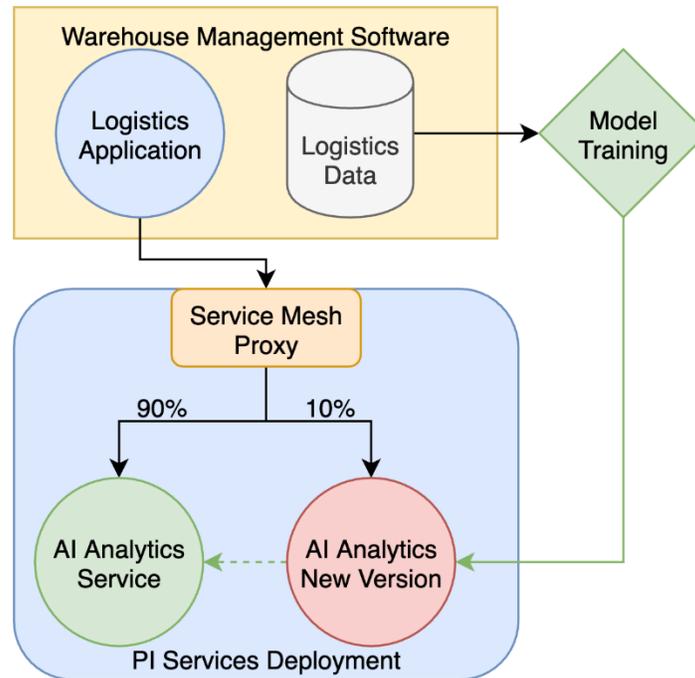


Figure 4 - Service Mesh Flexible Routing

In Figure 4 we see an example. In this scenario a warehouse management software system is using a machine learning powered analytics service to help optimize operations within the warehouse. Data regarding the warehouse operations is stored by the management software and is used to train new machine learning models based on recent activity ensuring the analytics and predictions are up to date and as accurate as possible. New models are deployed on a regular schedule. The logistics application then makes requests directly to the analytics service. However, when a new model is ready to be deployed it is not good practice to push it to a live deployment without first undergoing a verification phase. The use of a service mesh allows a single endpoint proxy that the logistics application will interface with. The application is not aware of nor need it be aware of any changes that occur behind the proxy. The proxy is configured to route a percentage of traffic to the new model where it can be verified while the majority of traffic is routed to the current model. Once confirmation is received the new model is performing in line with the old model, the new model can be configured to accept 100% of the traffic with no interruption to service.

In the ICONET PoC AWS provides a mesh service known as App Mesh as it allows full integration with existing AWS services. All major cloud vendors offer comparable services and in case of an on-prem or self-managed cloud deployments and these approaches can be easily replicated across any vendor or using open source private cloud approaches.

3.3 External Integration & Access Scenarios

The ICONET PoC Environment has three major external integration points:

- Data acquisition from non-PI legacy logistics systems (such as a warehouse activity database)
- PI Service access and use by non-PI logistics systems (such as warehouse management software)
- Simulation access for interested parties to view, use and evaluate simulations.

In addition, PI Service developers access the PI services and simulation services via the secure VPN server. In Figure 5 below we can see a visual representation of these integration points.

- Point A (top left) represents Anylogic users accessing the ICONET PoC environment. This is unique to the ICONET PoC and allows industry stakeholders to access, use and trial the simulations that utilize the PI Services and demonstrate the KPIs against which they will be evaluated.
- Point B (middle left) represents a legacy logistics service utilizing and communicating with a PI Service
- Point C (middle right) represents an ICONET WP2 PI Service developer accessing the PoC environment via secure VPN
- Point D (bottom left) represents a legacy logistics system passing info and datasets to datastores used by the PI Services

These 4 scenarios are discussed in more detail in the following sections.

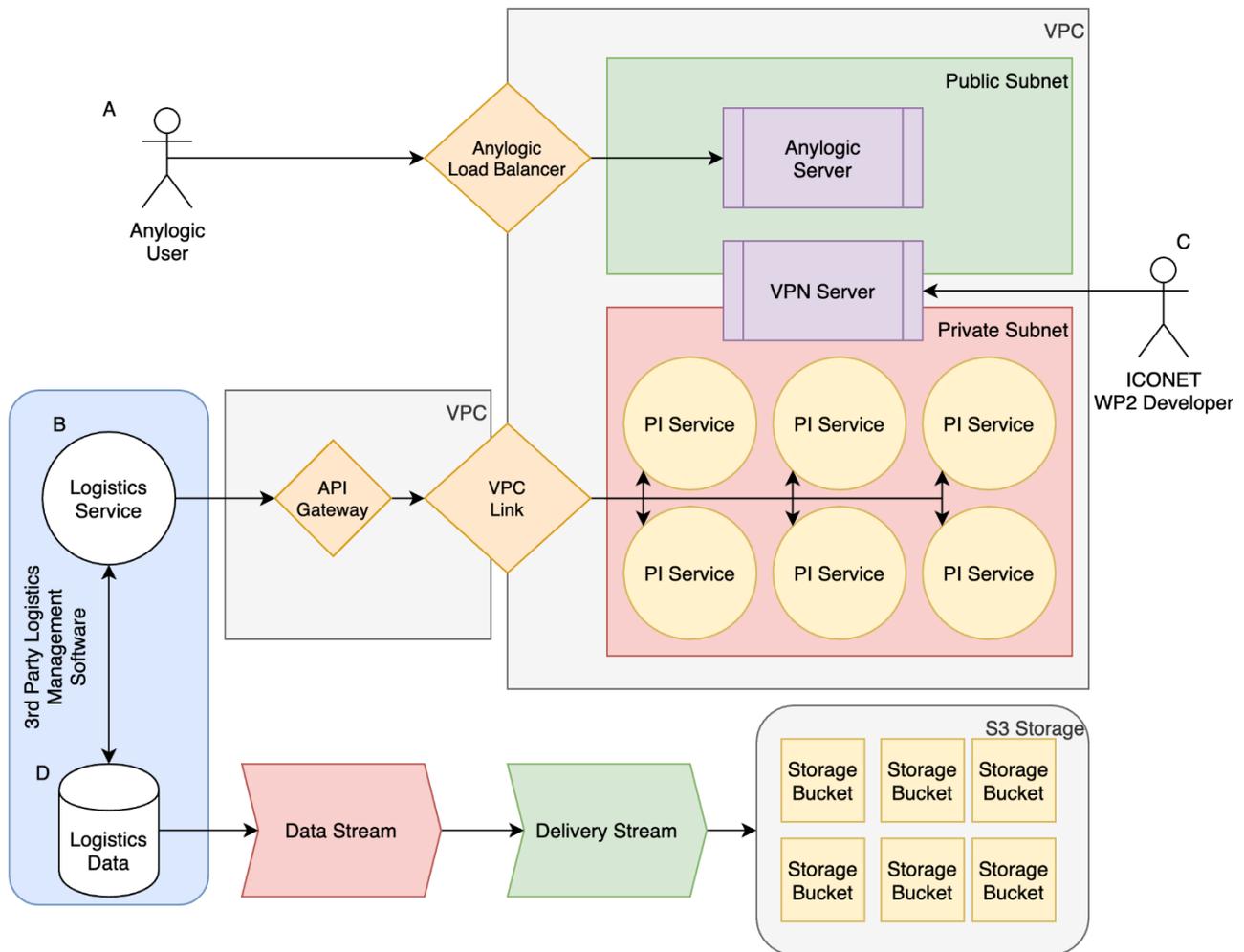


Figure 5 - External Integrations

3.3.1 Simulation Access (A)

The simulation service is deployed on a high resource virtual machine running Ubuntu Linux. The simulation service is actually comprised of a large number of distinct microservices, each communicating with each other. The internal connectivity requirements have been discussed in Version 2 of this deliverable and can be summarised as simply requiring the ability to access the various PI Services that are deployed in the PoC. However, in addition to this there are two further connectivity scenarios that are relevant in the final stages of

the project. The Anylogic Simulation allows for users to create (or alter) simulations on the client on a local machine or laptop and then push these new simulation scenarios to the cloud. Users can then use the Anylogic user interface to observe the results of the simulations as they run. Initial requirements were that Itainnova had the ability to do these tasks as they are the core simulation model designers and simulation experts of the project. They did this by connecting to the secure VPN (as discussed in Version 2). However, at this stage ICONET looks to make these offerings available to a wider audience. However, allowing external users to connect via the VPN is neither practical or secure as it is desirable to prevent public access to the back-end services and systems. To allow users to access the Anylogic systems an Internet facing load balancer is configured. This is an ELB (Elastic Load Balancer) that is software-based load balancer available from AWS. This was configured as an Application Load Balancer which means traffic can be filtered or blocked based on the traffic type (HTTP, HTTPS, SSH, etc.). ELB is a distributed system backed by AWS, so it has resistance to some common attacks and also distributes connectivity across multiple geographical zones. This means that in the event of a catastrophic failure at a specific datacentre, the service will remain live and available. The ELB provides a static public address that users can use to connect to the Anylogic UI. Anylogic provides its own user management system so there is no need to configure authentication, users, passwords, access control lists etc. The load balancer ensures that requests made via HTTP/S are forwarded to the Anylogic services, and all other requests are denied as intrusions. The load balancer also performs period health checks against the Anylogic service, a simple request to ensure the service is still responsive. For the PoC a failed health check will generate a notification to the administrator but for production level environments automated mitigating action can be configured.

3.3.2 PI Service Access (B)

Although this is a research project, it is good design practise to consider security and data protection at the outset of a development process. All the PI Services are developed in a microservices ecosystem where each service is supplied as a docker image. Each image has implemented the core functionality of that service and has not implemented any schema's related to networking, security or data protection. There are two reasons for this. The first is that this allows developers to focus on the core solution they are building and doesn't require expertise in other areas or time spent developing in other areas. The second is that by ensuring only the core functionality is addressed the PI Services are universally adaptable to an external solution that address the non-functional requirements.

Focussing on security and authorisation, we can see in *Figure 5* above that external systems will access the PI services through an API gateway. In the case of the ICONET PoC the API gateway is a service provided by AWS but there are many off the shelf tools that can be customised. The API gateway is essentially the "first point of contact" with an API from external systems. The API Gateway resides in its own private network, which is public internet facing, and appropriately firewalled and protected for this reason. The API gateway accepts incoming requests and based on the specific parameters is routed through the VPC link to the front facing load balancer of a given PI Service. The VPC link is a dedicated network route that connects the private VPC where PI Services reside to the public facing VPC where the API Gateway is but can only be used by the API Gateway itself. *Figure 6* shows some of the VPC links used in the ICONET PoC and the Routing VPC Link points at the Routing Load balancer.

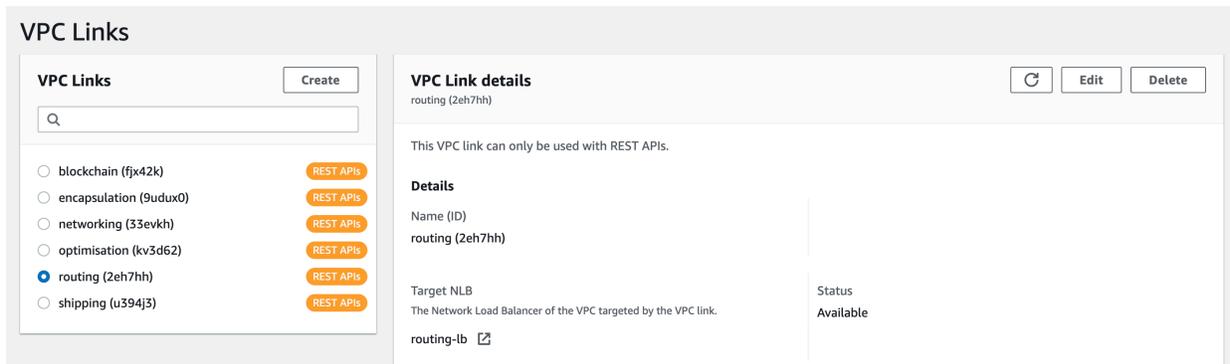


Figure 6 - VPC links to Service Load Balancers

There are many possible actions the API gateway can perform before but the main focus in this implementation is that the API gateway is configured with an authoriser. The authoriser can be configured to work with existing identity management systems to allow only requests that contain authorised tokens and secret keys to access a specific PI Service. This allows authorisation and access to all PI Services to be managed in a centralised manner while offloading this responsibility away from the PI Service developers.

3.3.3 Developer Access (C)

This integration point is for development and administrator access only. By design only interfaces that require external access are made accessible on public networks. There are a number of additional interfaces and components that are in the PoC environment that are not accessible publicly for security reasons. To allow the PI Services developers access for deployment, test, verification etc. a VPN (Virtual Private Network) service was configured. This allows developers to access the internal networks of the PoC environment from their local laptop or workstation as if the laptop was actually part of the PoC environment network. It does so in a highly secure manner ensuring no unwanted access can occur. This scenario is discussed in more detail in the intermediate version of this deliverable (D2.20).

3.3.4 Data Acquisition (D)

One of the core drivers of the PI Services in ICONET is the data provided by the Living Labs. This is reflective of a real-world implementation of the PI and PI Services scenario. The PI Services themselves provide abstraction, common protocols and standards, analysis and prediction among other functions, but they are all powered by the data provided by logistics organisations. This data can be day-to-day operational data such as the exchanging of orders or it can be large scale data for analytics processing, such as a month's worth of data related to late deliveries. The daily operations of data exchange are handled by the Logistics Web Service and the Shipping Service, but large-scale datasets (or data streams) for analytics need a dedicated consumption strategy.

Data streams (4) Info						
<input type="text" value="Find data streams"/>						
<input type="checkbox"/>	Data stream name	▲	Open shards	▼	Status	▼
<input type="checkbox"/>	ll1		1		Active	
<input type="checkbox"/>	ll2		1		Active	
<input type="checkbox"/>	ll3		1		Active	
<input type="checkbox"/>	ll4		1		Active	

Figure 7 - Living Lab Input Streams

There are a large number of possible strategies for large scale data consumption, but many are based on similar approaches. As shown in Figure 7, within the ICONET PoC a data stream was created for each Living Lab. A data stream is a dedicated API endpoint where data can be submitted via API requests. Data is forwarded to the data stream by an agent which is installed and run at the data source, making API requests to the stream. The agent is simple and light weight and usually works by simply capturing log files as they are generated. Each stream has some configuration options and allows for monitoring of the incoming data to ensure the stream is active and functional. This means that each Living Lab or real-world data source can be monitored independently with regards frequency of data and is an early warning mechanism if data streams fail. Figure 8 below shows the monitoring dashboard for the Living Lab 1 data stream.

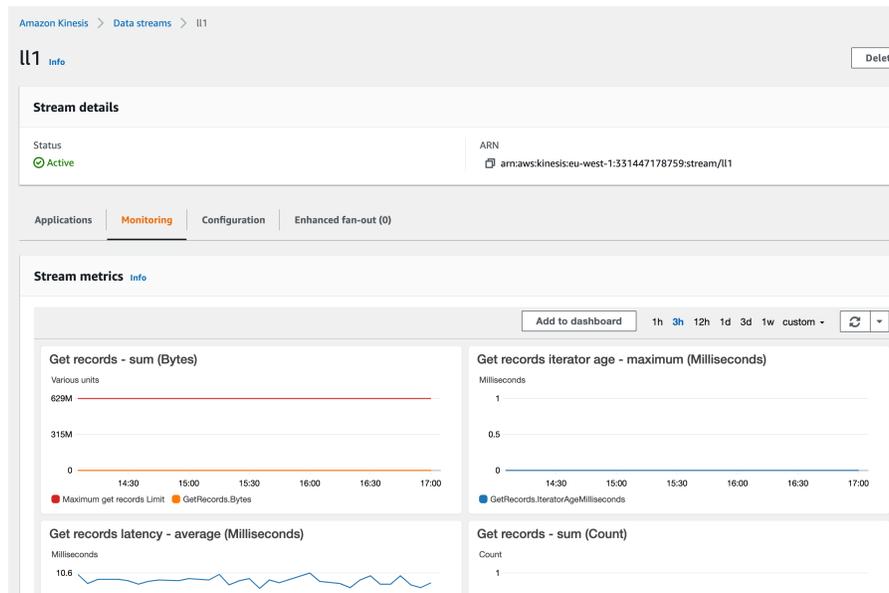
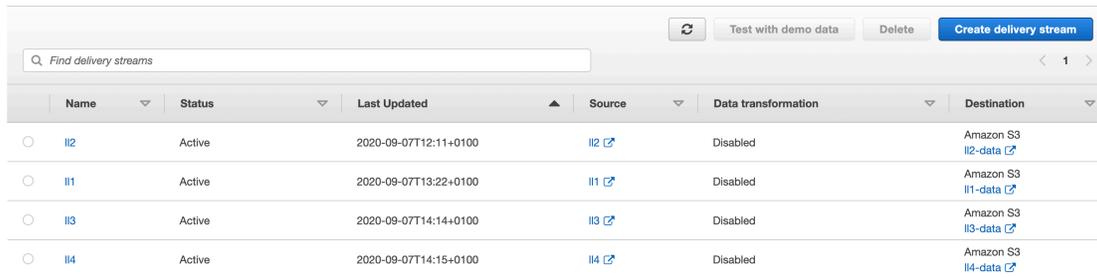


Figure 8 - Data Stream Monitoring

The data stream passes the data to a delivery stream. This is essentially a dedicated agent that consumes data coming out of the stream, processes it and sends it to its final destination. The delivery stream has the ability to convert the data into a common format, encrypt it and save it in a dedicated storage slot for this data steam. Each Living Lab has a dedicated data stream, delivery stream and storage space. Figure 9 shows the delivery streams for each Living Lab in the management console and Figure 5 (Point D, bottom left) shows the data ingestion point and the components discussed here. In the case of the ICONET PoC AWS Firehouse and S3

services were used for this. These systems are based on industry standards used in other major tools such as ELK (Elasticsearch Logstash Kibana) and the strategies employed can be reused in these tools if needed.



The screenshot shows a web interface for managing data delivery streams. At the top, there is a search bar with the placeholder text "Find delivery streams". To the right of the search bar are three buttons: "Test with demo data", "Delete", and "Create delivery stream". Below the search bar is a table with the following columns: Name, Status, Last Updated, Source, Data transformation, and Destination. The table contains four rows of data, each with a radio button in the Name column.

Name	Status	Last Updated	Source	Data transformation	Destination
<input type="radio"/> II2	Active	2020-09-07T12:11+0100	II2 ↗	Disabled	Amazon S3 II2-data ↗
<input type="radio"/> II1	Active	2020-09-07T13:22+0100	II1 ↗	Disabled	Amazon S3 II1-data ↗
<input type="radio"/> II3	Active	2020-09-07T14:14+0100	II3 ↗	Disabled	Amazon S3 II3-data ↗
<input type="radio"/> II4	Active	2020-09-07T14:15+0100	II4 ↗	Disabled	Amazon S3 II4-data ↗

Figure 9 - Living Lab Data Delivery

3.3.5 Node-to-Node Communication & Integration with Legacy Systems

Node-to-Node communication is a critical and key element to the PI concept. In the above scenarios, section PI Service Access (B) discusses the manner in which legacy systems will communicate and access PI Services. The same technologies, approaches and policies will govern how two PI Nodes will communicate. The Shipping PI Service serves as the interface for PI Node to another node, and therefore two Shipping services will exchange information about PI Orders across API gateways. As before, the API gateway will manage the implementation of the authorisation and access, security tokens etc. but this must be managed across multiple nodes. In early PI pilot programs, the PI will essentially be a “network of networks” where two transport and logistics organisations will agree to connect their networks using PI protocols. In these circumstances the management of tokens and authorisation can be handled manually. As the networks grow in numbers of participants it will become impractical to manage authorisation, keys and identity manually and will require central management, likely owned by a local trusted transport and logistics entity such as a port authority.

Integration with legacy systems such as ERP, WMS, TMS is also a critical element in terms of PI Integration. PI systems, in particular for early adoption of PI and PI pilot programs. PI Services will initially be unable to function in isolation and will require significant integration with legacy systems in order for the PI to be leveraged. The exact manner and means of integration falls into distinct layers:

- The business level – this would be where organisations have agreements to connect their networks together. This is a practical and legal process, not a technical one.
- The data level – the PI is the common data model, so that has been already define by GPICS and PI Architecture, however the interface between existing legacy systems and PI Services will be required, and this is a service provided by PI Service developers and IT staff in logistics organisations in a coordinated effort. There can be no “universal adapter” as there are many different legacy systems with their own data models. Although rare currently, standardized data models for logistics are becoming more prevalent in research and industry and should be utilized for PI pilot programs.
- Access level – this is concerned not with what is communicated – but how is relevant to this document. Regardless of the data model the manner and means is likely to be a REST API based communication in line with existing PI Services. Authentication will likely be via authorised token or VPN or a combination of both and these scenarios both align with existing external connectivity scenarios for the PoC Integration Platform. It is possible that an intermediate application such as a “data adapter” is required but this is simply to change the format of the data and something like AWS Lambda using serverless functions can be utilized to do this. Therefore, the combination of API gateways and AWS lambda provide and out-of-the-box approach for legacy integrations with PI Services in a manner and means described in section 3.3.2.

4 Deployment Strategies, Methodologies and Procedures

While ICONET is a research and development action, it is worth considering how the technical assets produced by the project can be adopted in a real-world production engagement. In modern cloud deployments the CI/CD (Continuous Integration/Continuous Deployment) approach is widely adopted and suits the needs of this project well. CI/CD is based on the idea that as a developer commits new code and new changes, these are integrated and deployed immediately as long as they pass a series of tests and verifications. This means that new features and fixes can be made available to users immediately, in opposition to the “scheduled release” approach which requires all changes and updates to be packaged together and all services and components to release these changes in sync. This means updates will be complete but unused while other updates are in development.

4.1 Pipelines

The key mechanic that powers CI/CD is the concept of a pipeline. A pipeline is simply a series of steps that the service or application passes through from code commit through to deployment.

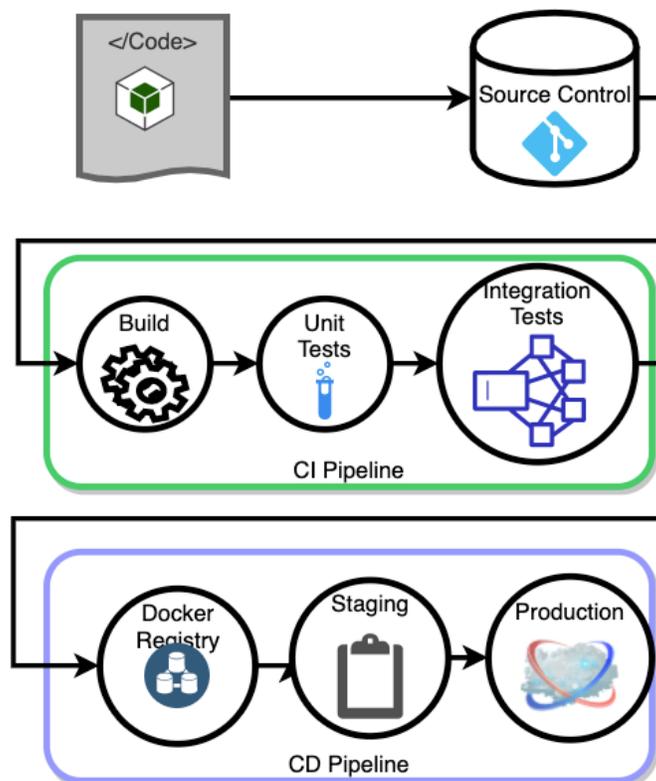


Figure 10 - CI/CD Pipeline

As we can see in Figure 10 a developer will commit a new codebase to the source control repo for that component or service and this will initiate the build process. In the above example the integration phase will build the component, run the unit tests (self-tests) and the integration tests (testing the components behaviour in relation to other components) before moving to the deployment phase. In the deployment phase the service is committed to the docker registry for future re-use if needed, deployed into the staging environment and then deployed into production. All PI Services in the ICONET PoC are deployed into AWS Fargate clusters. Fargate is a serverless compute engine that the Elastic Container Service uses to run containerised applications. Fargate has been discussed in more detail in the intermediate version of this deliverable (D2.20). It should be noted however this is just one potential pipeline configuration, it is possible to add any number of stages in the pipeline

depending on requirements. For example, a security audit and evaluation is another common stage that pipelines will include. As already stated, this project was to focus on the use of PaaS as much as possible, so the pipeline systems provided by AWS were leveraged. However major pipeline systems available from other cloud vendors or open source options such as Jenkins are very similar in their approach and use in that they accept assets and resources from external sources or previous steps, perform an action and based on the results perform one or more subsequent steps. On AWS the pipeline is provided as part of the AWS developer tools:

1. CodeCommit: This allows you to create source control repos. ICONET has 17 different source code repositories for various versions and sub-components of the services developed for WP2.
2. CodeArtifact: This allows storage and version control of artifacts created as part of a build process (for example a java library jar file)
3. CodeBuild: This allows for the building of applications. In the case of ICONET all services are containerised, so the build phase builds docker images using the Dockerfile in the source repository and packages in the source code and dependencies required for the image.
4. CodeDeploy: This tool allows the creation of “applications” to define how multiple components and services might make up a single application. Then this application can be deployed using images from the build phase into either a staging environment or straight into a production environment. In the case of ICONET services are deployed into ECS (Elastic Container Service) Fargate clusters.
5. Pipeline: This is the tool that allows the creation of an actual pipeline composed of the previous 4 tools. As we can see in Figure 11, the codebase and docker registry make up the source phase of the pipeline, then there is a test phase and finally a deployment phase, but it’s possible to add an intermediary stage at any point in the pipeline such as a security audit.

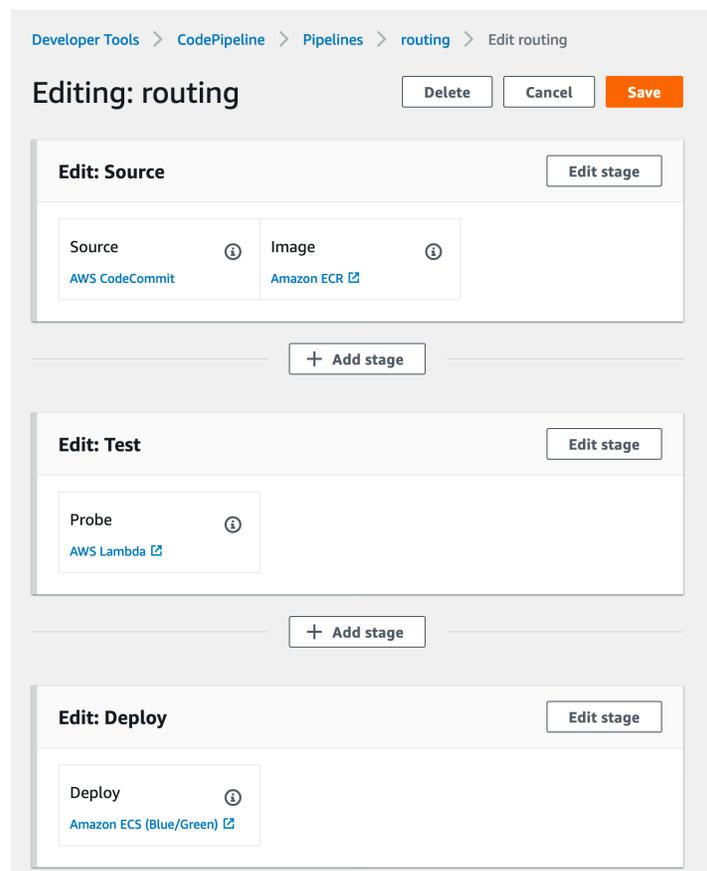


Figure 11 - Routing Pipeline

4.2 Recommended Pipelined Testing Strategies

There are a number of potential approaches for testing the PI services before deployment. If a PI Service developer has included a set of unit tests, then it's simply a matter of running these tests. The creation of Unit, API and Integration tests are part of the development process and outside the scope of this deliverable. However, IBM did build out the frameworks and pipelines to support testing, as well as implement the security audit process as discussed below.

4.2.1 Unit Tests

Unit tests are a set of functions provided by the developers that ensure that for a given set of inputs, expected outputs are received for the various internal components of a service. To avail of this one can either launch the docker image with some specific command line variables passed, to initiate the tests, or one can check out the codebase and run the tests directly. Which approach will depend on the type of approach chosen by the developer. Both are supported by the AWS Developer toolkit.

4.2.2 API Tests

API Tests will simulate real requests to the API service and verify that responses returned are accurate. These tests are not always applicable as some services will have multiple downstream dependencies that are required for the API responses to be delivered. However, some services do not have such a reliance or can be adapted to not need one. For example, the routing service depends on the networking service, but the routing service has been designed to allow the static upload of a PI network. This allows the service to be tested by ensuring the correct routes are returned. To perform these tests AWS Lambda can be used. As discussed in D2.20, version two of this deliverable, AWS Lambda is a serverless code execution engine. This allows relatively simple scripts or code to be written on run on a schedule or as needed. In this instance AWS lambda can make a HTTP API REST request to the service being tested, verify the result and return a PASS or a FAIL.

4.2.3 Integration Tests

Integration Testing verifies the components will work together as designed. Using routing as an example once more, routing requires the networking service to operate. To perform an integration test for routing, the pipeline will deploy the new version of the routing service to be tested to a temporary staging cluster along with an already known valid version of the networking service for the routing service to make requests to. The routing service will be again tested by verifying the API requests using AWS Lambda or similar. If the single API test passes but the integration test fails, this highlights a likely issue in how the services are communicating, but not how they function.

4.2.4 Performance Tests

Performance testing is outside the scope of the ICONET project as any use of the PI services is taking place in isolated pilot programs with the living labs. Performance testing is needed to address the limits of a single PI Service instance with regards the number of concurrent requests it can accept and also to understand the manner and means of failure that will be observed as the service begins to reach that limit and then actually hit this limit. These kinds of KPIs are relevant where services are experiencing high usage in the form of multiple requests per second. While the volume of requests in the ICONET PoC is low enough not to require performance testing, this kind of testing and evaluation will be critical in any scaled real-world pilot programs for the PI.

4.3 Security Audit

The nature of microservices deployments with Docker as their backbone requires a new approach to security audits and validations. Docker images are built by inheritance from a base image. These are usually operating systems as Docker images (Ubuntu, CentOS etc.). Some additional tools may be added to this base image (for example nodejs) to give it new functionality and this now becomes a new image. Then finally the PI Services developer will pick an image to be the basis for the PI service. This means that any given docker image may have

a number of inherited images that it is built upon. Each with a vast array of potential security audit points. In addition to that almost all PI Services are built using python. As part of the process of building a docker image the python dependencies are installed. These are libraries that the PI Service application will need in order to function correctly. The developer has full control to add as many dependencies as they see fit from any number of potential sources. The resulting docker image will contain the PI Service code but will also contain a large number of “unknowns” in terms of base image components, libraries etc.

Image URI	Pushed at	Digest	Size (MB)	Scan status	Vulnerabilities
331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet-router-ll3:latest	09/18/20, 03:27:22 PM	sha256:506ada3b8...	454.58	Complete	2 Medium + 941 others (details)

Figure 12 - Vulnerability Notification

It’s not feasible to audit these by hand so a docker image audit tool is used to perform this. As seen in *Figure 12*, In the case of ICONET, the audit tool allows for images to be scanned as they are pushed to the registry, and a notification generated if the resulting report requires attention. An example report is shown in *Figure 13*. The report indicates two medium severity CVEs (Common Vulnerability or Exposure). However, there are a number of open source and commercial alternatives such as Anchore or Clair.

ECR > Repositories > iconet-router-ll3 > sha256:506ada3b803501d1ae5be617d07d3e6ac8677db38e0fa26a7a6334a8d81bb58

Overview

Critical	High	Medium	Low	Informational	Undefined
0	0	2	184	348	409

Vulnerabilities (943)

Find vulnerabilities

Name	Package	Severity	Description
CVE-2009-3546	libwmf:0.2.8.4-10.6	MEDIUM	The _gdGetColors function in gd_gd.c in PHP 5.2.11 and 5.3.x before 5.3.1, and the GD Graphics Library 2.x, does not properly verify a certain colorsTotal structure member, which might allow remote attackers to conduct buffer overflow or buffer over-read attacks via a crafted GD file, a different vulnerability than CVE-2009-3293. NOTE: some of these details are obtained from third party information.
CVE-2007-3996	libwmf:0.2.8.4-10.6	MEDIUM	Multiple integer overflows in libgd in PHP before 5.2.4 allow remote attackers to cause a denial of service (application crash) and possibly execute arbitrary code via a large (1) srcW or (2) srcH value to the (a) gdImageCopyResized function, or a large (3) sy (height) or (4) sx (width) value to the (b) gdImageCreate or the (c) gdImageCreateTrueColor function.
CVE-2017-13613	apr:1.5.2-5	LOW	When apr_time_exp*() or apr_os_exp_time*() functions are invoked with an invalid month field value in Apache Portable Runtime APR 1.6.2 and prior, out of bounds memory may be accessed in converting this value to an apr_time_exp_t value, potentially revealing the contents of a different static heap value or resulting in program

Figure 13 - Vulnerability Report

Clicking the CSV link on the left will bring more detail on the CVE and how to resolve as shown in the detailed report in *Figure 14*. In this instance, we can see three versions of this library that are vulnerable to attack, so

changing to any of the known fixed versions will resolve the issue. In this circumstance the issue was with the base docker image, not the developers code, so to update it will require rebuilding the base image with the libraries updated.

Vulnerable and fixed packages

The table below lists information on source packages.

Source Package	Release	Version	Status
libgd2 (PTS)	stretch	2.2.4-2+deb9u5	fixed
	stretch (security)	2.2.4-2+deb9u4	fixed
	buster	2.2.5-5.2	fixed
	bullseye, sid	2.3.0-2	fixed
libwmf (PTS)	stretch	0.2.8.4-10.6	vulnerable
	buster	0.2.8.4-14	vulnerable
	bullseye, sid	0.2.8.4-17	vulnerable
racket (PTS)	stretch	6.7-3	fixed
	buster	7.2+dfsg1-2	fixed
	bullseye, sid	7.8+dfsg1-1	fixed

The information below is based on the following data on fixed versions.

Package	Type	Release	Fixed Version	Urgency	Origin	Debian Bugs
libgd2	source	etch	2.0.33-5.2etch2		DSA-1936-1	
libgd2	source	lenny	2.0.36-rc1~dfsg-3+lenny1		DSA-1936-1	
libgd2	source	(unstable)	2.0.36-rc1~dfsg-3.1	medium		552534
libwmf	source	(unstable)	(unfixed)	unimportant		
php5	source	(unstable)	(not affected)			
racket	source	(unstable)	5.0.2-1	unimportant		601525

Notes

Figure 14 - CVE Information

4.4 Deployments

The final phase of the pipeline is deployment. There are two deployment paradigms that have been considered. In the case of a less substantial update – for example a minor bug in the UI (User Interface) of a web service - a typo. To fix this a single service needs to be updated and it has minimal or no bearing on other services. Services are packaged within the AWS dev-ops tools as applications. An application may comprise of one or more services that are linked closely together. For example, the routing service operates by itself so an application definition comprises of a single service. However, the IoT service is made of multiple services, so the IoT Container tracking application is comprised of all these services. The application also manages the historical revisions of the application and components that were deployed in the past. This facilitates rolling back to known functional versions in the event of a faulty deployment.

Figure 15 - Deployment Group

Within an application, a deployment group is configured for each PI Service. As seen in Figure 15, the deployment group allows for the definition of where the PI Service will be deployed to, specifying which Fargate cluster and which service definition within that cluster to use. The deployment group also defines the load balancer and port assignments of the PI Service and finally the traffic rerouting strategy.

The traffic rerouting strategy allows for the setting of a deployment schedule by specifying a specific time to remove the old service and deploy the new one. An approach here could be to engage the deployment at night when there are little or no users if the service is only used during the day. The traffic rerouting strategy can also set rate of rerouting. For example, every hour redirect and additional 10% of the traffic to the new deployment until the old deployment is receiving no traffic. This allows for careful monitoring of the new service or deployment and ensures issues are highlighted in time to reverse the deployment.

An alternate strategy for deployments is to use a staging environment. This is more applicable for deploying much more significant changes that encompass multiple PI Services acting together, such as a new standard or policy. In this situation there are two production environments available. One is accepting traffic from users, while the other is used to deploy the latest service versions. The components have already been tested via the pipeline, but some validation and testing can take place of the environment overall. Once this is done, the endpoints that users are using to access the environment is switched from the existing live environment to the staged. This is instantaneous and there is no downtime from the user's perspective. Once again, if any issues are observed, the endpoints are simply swapped back to the known good environment.

Configure Iconet-env-staging

Presets
Start from a preset that matches your use case or choose *Custom configuration* to unset recommended values and use the service's default values.

Configuration presets

- Single instance (*Free Tier eligible*)
- Single instance (using Spot instance)
- High availability
- High availability (using Spot and On-Demand instances)
- Custom configuration

Platform

Multi-container Docker running on 64bit Amazon Linux/2.22.0

[Change platform version](#)

<p>Software</p> <p>Rotate logs: enabled Log streaming: enabled Environment properties: 1 env</p> <p>Edit</p>	<p>Instances</p> <p>Root volume type: container default Root volume size (GB): container default Root volume IOPS: container default Security groups: sg-0bf43e9e12fab1619</p> <p>Edit</p>	<p>Capacity</p> <p>Environment type: load balancing, auto scaling Availability Zones: Any Fleet composition: On-Demand instances EC2 instance type: t2.large EC2 image ID:ami-09c44e3b186e27e52 Instances: 1-4</p> <p>Edit</p>
<p>Load balancer</p> <p>Load balancer type: classic Listeners: 1 Session stickiness: disabled Cross-zone load balancing: enabled Connection draining: enabled</p>	<p>Rolling updates and deployments</p> <p>Deployment policy: All at once Rolling updates: disabled Health check: enabled</p>	<p>Security</p> <p>Service role: arn:aws:siam::331447178759:role/aws-elasticbeanstalk-ec2-role Virtual machine key pair: rancher_rebuild Virtual machine instance profile: ec2power</p>

Figure 16 - AWS Beanstalk Configuration

AWS abstracts this process out using a tool called beanstalk. Beanstalk allows you to clone an environment once it has been deployed once and manages the public endpoints to that environment. This then allows for multiple versions of an environment to be easily recreated and multiple deployment pipelines configured as requirements need. For example, there may be a live environment, a staging environment and a demo environment (which allows critical customers to view potential new features and give feedback while development is still underway). Beanstalk allows for centralised management of all environments and some of the specifics within those environments also, as shown in Figure 16.

5 Machine Learning Deployments

A number of PI Services either currently or will in the future leverage machine learning technology. These services will still be leveraged via REST APIs as the other services are, but with a different analytical approach behind the API interface. As seen in the **Deployment** section, the cycle for other services is straightforward:

- Developer checks in new code
- Code is built into a service
- Service is tested, audited, verified
- Service is deployed

However, services that are based on machine learning approaches require a different approach to pipelines and deployments. The basic process for the development and creation of a machine learning API service covers the following:

1. Data acquisition – All machine learning research requires significant datasets to work with for training and validation.
2. Data Curation – Removing Data that is not relevant. For example, if analysing traffic patterns for commuting rush hour, weekend data can be removed completely.
3. Data pre-processing – the data may not be in a desirable format and will need to be converted depending on the tools being used and the formats they will accept
4. Definition of a machine learning approach or algorithm - This is the main task of the developer, ensuring they pick an approach that matches the datasets and goals of the research
5. Training the machine learning model – this is the most computationally (and monetarily) expensive and time-consuming part of the process. The time required will vary vastly depending on the hardware available for training. CUDA GPUs will increase the speed at which training can take place but these components are extremely expensive to purchase and equally expensive to rent through a cloud provider.
6. Trained model is verified against the dataset – The dataset is usually split into two parts, one to train and one to validate the model. Once the model has been validated to some chosen metrics (such as accuracy or rate of false positives, false negatives etc.)
7. The model is deployed to an API service that accepts requests, passes them to the model, captures the results and returns them to the requestor.

5.1 Requirements for new Machine Learning Deployment

As can be observed, there are a number of additional steps and there are now more than one places where a change could initiate the requirement for a new service deployment:

1. The quality of the datasets for training changes
2. The format of the datasets for training changes
3. The developer changes the algorithm or approach (or alters the existing approach slightly)
4. A new dataset is provided

The last option, number 4 is of course the most common and most frequent. A machine learning service is only as good as the data that it has been trained on. Therefore, any analytics being done on data that changes periodically, seasonally or in line with other trends (financial, or consumer for example) will require new models trained on data that represents these latest trends. A simple example of this would be any analysis of the product choices consumers make for delivery during the summer would be meaningless during the months leading up to Christmas. Similarly, analytics noting an increase in Pumpkin sales during the month of October will likely not be relevant during the rest of the year.

5.2 Microservices Approach

The initial development process combines the actions of curation, pre-processing, model definition and training into a single application or script. This works well at early development as the developer can make multiple changes in parallel however as the application approaches production, we see issues with this. A change to the pre-processing method should not require us to retrain the image. Therefore, an emerging trend is to split the actions into individual microservices, each dedicated to a specific task. This allows updates to one service without interference with another. In the case of ICONET these services are docker based (as are all components in ICONET) which allows them to be pipelined and deployed in the same manner as the other PI Services and using approaches discussed in the **Deployment** section.

5.3 Model Training as a Service

However, there is an exception to this approach – the model training service will require access to specific hardware. This is both expensive and complicated as most PaaS offerings do not easily allow access to such hardware in their container compute services. One alternative is to purchase (or reserve) some dedicated hardware with a GPU installed, but this is expensive and also not very cost effective as models are only trained periodically.

PaaS offerings that are specifically design for machine learning are the best approach here. They will utilise industry standard tools (such as Jupyter Notebooks) to allow developers to migrate their algorithmic approaches to model training into the cloud. For the ICONET PoC AWS Sagemaker was used but all major cloud vendors have similar services such as IBMs Watson Machine Learning or Google’s AI Platform. This allows models to be trained as needed in the cloud. This brings a number of advantages, the first of which is cost. Since the hardware is only engaged for the time needed to train the model, there is no lost cost to hardware sitting idle. Additionally, there is no overhead in terms of administration or setup for dedicated hardware and because the models are trained using a PaaS service, they accessible by other PaaS systems such as the pipeline tools and deployment tools.

The screenshot shows the AWS SageMaker console for a training job named 'ICONET-LL4-Opt'. The job is in a 'Starting' state. The console displays the following details:

Job settings			
Job name	Status	SageMaker metrics time series	IAM role ARN
ICONET-LL4-Opt	Starting	Disabled	arn:aws:iam::331447178759:role/service-role/AmazonSageMaker-ExecutionRole-20200730T201804
ARN	Creation time	Training time (seconds)	
arn:aws:sagemaker:eu-west-1:331447178759:training-job/iconet-ll4-opt	Oct 08, 2020 10:58 UTC	-	
	Last modified time	Billable time (seconds)	
	Oct 08, 2020 10:58 UTC	-	
		Managed spot training savings	
		-	
		Tuning job source/parent	
		-	
Algorithm			
Algorithm ARN	Instance type	Additional volume size (GB)	Volume encryption key
-	ml.m4.xlarge	30	-
Training image	Instance count	Maximum runtime (s)	
685385470294.dkr.ecr.eu-west-1.amazonaws.com/xgboost:latest	1	86400	
Input mode		Maximum wait time for managed spot training(s)	
File		-	
		Managed spot training	
		Disabled	

Figure 17 - Sagemaker training job in progress

Once the training and algorithmic approach is defined, this can be created as a training job as shown in Figure 17. Combined with new data either being streamed or manually uploaded to a dedicated storage location, this allows training jobs to be run on a predefined schedule, occurring frequently enough to account for any trends in the dataset (weekly, monthly, etc.). The resulting models are then stored in a dedicated storage location within the cloud and are accessible for further use as shown in Figure 18

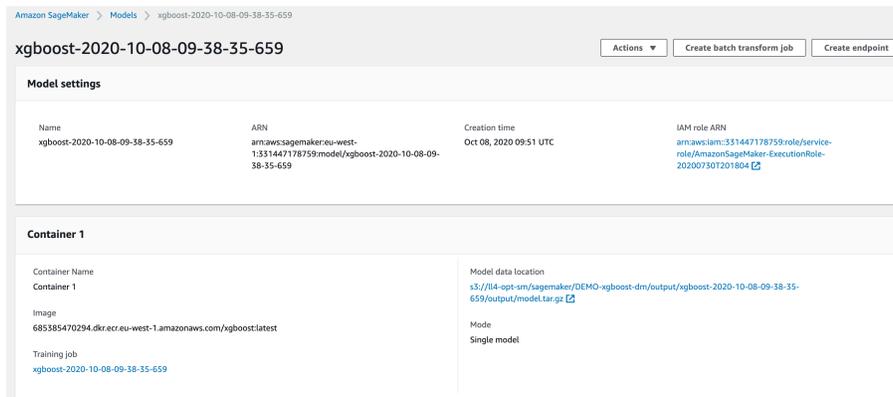


Figure 18 - Saved Model

5.4 Machine Learning as a Service

At this point, the model has been trained as is available in an S3 storage bucket (AWS refers to storage locations as “S3 buckets”). There are two available approaches from this point forward. It is possible to use the arrival of new model in S3 to trigger a pipeline build similar to the pipeline described in the **Pipelines** section. In fact, much of the pipeline can be reused, aside from the triggering incident (a new model instead of new code. A machine learning based service is built into a docker image in the same manner as any other PI service, however as part of the build process it will import the model from S3.

The alternative approach is to avoid the use of docker images entirely. Usually, the bulk of the processing work in a machine learning based service is performed by the model itself, and the docker image and other code within are simply there as frameworks to support the use of this model, usually to simply accept requests via REST API and pass the data provided to the model, capture the result and return it to the requestor. Since this functionality is both simple and likely to be similar for any service that relies on a machine learning model, we can implement it using PaaS in a much more efficient, secure and cost-effective manner.

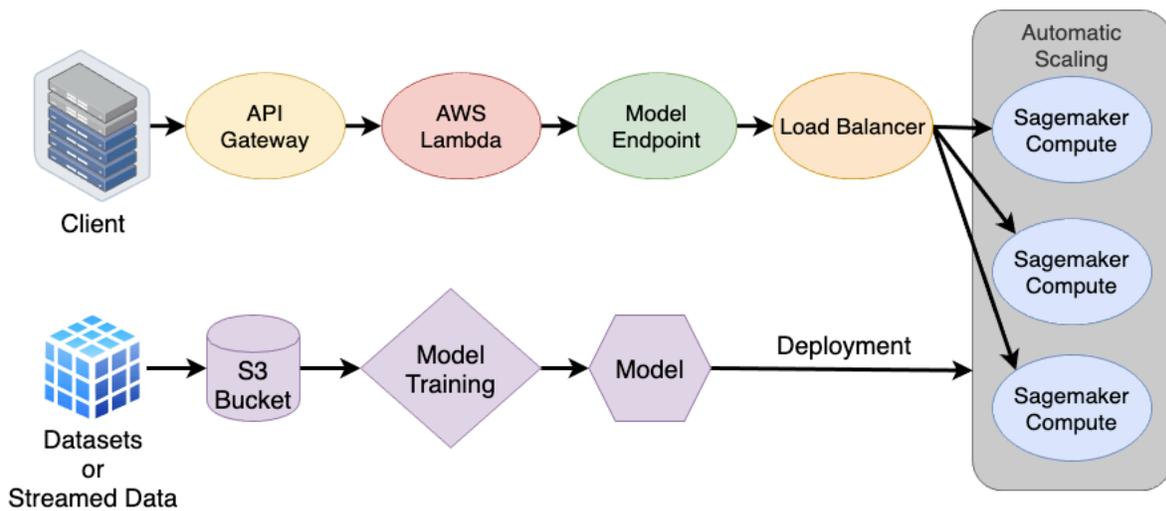


Figure 19 - Machine Learning Endpoint - PaaS Powered

Figure 19 shows the various components involved in providing a machine learning powered service using only PaaS services. On the top row we can observe:

1. The client – this is likely another PI Service or a 3rd party logistics management tool that is availing of machine learning analytics.
2. The API Gateway – Also seen in **the Integration Strategies & Approaches** section, this tool provides a public facing interface to the API. Through a simple configuration process the API Gateway can handle some non-functional mechanics such as authentication keys, authorisation, certificate management and so forth. Requests are made directly to the API Gateway and once the authentication & authorisation elements are validated, the request is passed on (in this case) to AWS Lambda.
3. Lambda is a serverless execution engine provided by AWS. Serverless execution allows for relatively simple applications or scripts to be run without the overhead of managing the libraries or underlying compute such as docker, Kubernetes, virtual machines etc. In this case the code that captures the request data from the API request and passes it to the machine learning model is implemented as a lambda function because it is quite straightforward.
4. AWS Sagemaker provides an endpoint for a given machine learning model, as a single point of access for entry. This can take some configuration in terms of the compute resources allocated to the endpoint and also serves as a point of monitoring where some metrics such as rate of request can be captured.
5. The endpoint sends requests to a load balancer which distributes them across however many compute resources are available. It is possible to configure the endpoint compute nodes to scale automatically as needed – adding additional nodes as the rate of requests or complexity of requests increases ensures there is no degradation in performance but ensures there is no cost for hardware sitting idle either.

On the lower half of Figure 19 we see the general steps for the creation of the models that the endpoint is using. Data is either stream or uploaded (more details in **the Integration Strategies & Approaches** section) and is stored in S3 storage buckets. The data is used to train new models which are deployed by Sagemaker to the model endpoint. What is noteworthy here is that aside from the core work of designing an algorithmic approach that the model will use, there is no further overhead from a development perspective. A data scientist can do effective and useful work without the support of additional developers and with minimal training in the AWS services that support the service.

Looking to the future, low power edge devices operating in a fog network will be utilized for inference based on machine learning models. This can also be combined with the concept of distributed learning, which spreads the analytics across a variety of devices in a fog network with dispirit data sources, allowing for more accurate analytics. This is discussed in more detail in the following section.

6 Future Trends in PI Service Deployment Models

The core goals of this deliverable and associated tasks were to provide a digital workbench that facilitated the deployment, development and integration of the PI Services and other services and systems utilized in WP2. Beyond that research was done to explore the non-functional considerations regarding the deployment, security and access scenarios that the PI Services may need to be coupled with. However, it is valuable to consider the potential deployment models that will be prominent as the PI begins to be introduced to real world logistics operations in early pilot programs.

The EU has already identified emerging trends in cloud computing and has directed H2020 funding towards projects exploring these future-looking computing paradigms. One of the major themes is to explore and facilitate the “Cloud to Thing Continuum”.

In the last decade, more and more major organisations are moving away from traditional “On Premise” models (where they deploy and manage their own data centres, equipment etc.) and moving their IT operations to the cloud, usually engaging a major cloud vendor such as Amazons’ AWS, Google’s GCE, Microsoft, Azure or IBM Cloud. However, in more recent years industry leaders like IBM are exploring trends moving towards the concept of a hybrid approach. Combining the cost effectiveness of the cloud with the security and control of a private datacentre (or private cloud). This has then further led to the concept of multi-cloud – where an organisation should be able to engage multiple cloud vendors as well as their own datacentres, to leverage the strengths of each that match their requirements and avoid being locked to any one approach.

The challenge in multi/hybrid cloud model is the overhead and complexity of trying to manage systems and services that are spread across a myriad of providers and even physical locations. Each cloud vendor has their own services, APIs, protocols, tools and private clouds will rely on open source tools or licensed tools. While there is often overlap in the use of these tools between vendors, there is not always, and there can be version and implementation mismatches.

The final emerging trend that requires consideration is the concept of edge and fog computing. Edge computing refers to placing the compute power and other resources and resource management systems closer to the consumers of said resources. A simple example would be a medium spec server that resides in a PI Mover (Truck, barge, ship or train) and has the capability to consume sensor data coming out of PI Containers. Rather than wait for connectivity to the cloud, this edge device will do its own analytics of the sensor data and forward the results of these analytics to the cloud. By placing the device near the edge, the need to forward huge volumes of data streams is dramatically reduced. The concept of fog computing expands this concept out further, to the idea that there are multiple edge devices, in tiered or clustered sets that need to be managed in tandem with the cloud services. Edge devices don’t always need to be resource heavy servers as in the example above, it could be as simple as a raspberry pi (a low powered, small compute device), monitoring the rate of trucks entering a port with a simple light sensor. These devices can use models that were trained in the high spec systems within the cloud. The increasing trend is that any system or device that can be connected to a network can now be considered part of a fog network. This applies to the transport and logistics domain as there are many systems and devices currently in use today that will very soon (or already) have cost effective alternatives that have enhanced processing and networking capabilities. Systems like handheld PDAs for delivery drivers, forklifts and sensors on forklifts, trucks and GPS tracking, automated warehouse robotics and intelligent loading systems in ports are all on the verge of a technological revolution that will see all of these systems joining the fog network.

6.1 Challenges for Multi-Cloud PI Services

So, with these trends in mind, how may they impact the PI, and how will the PI operate and exist in a world where these trends have become more prominent?

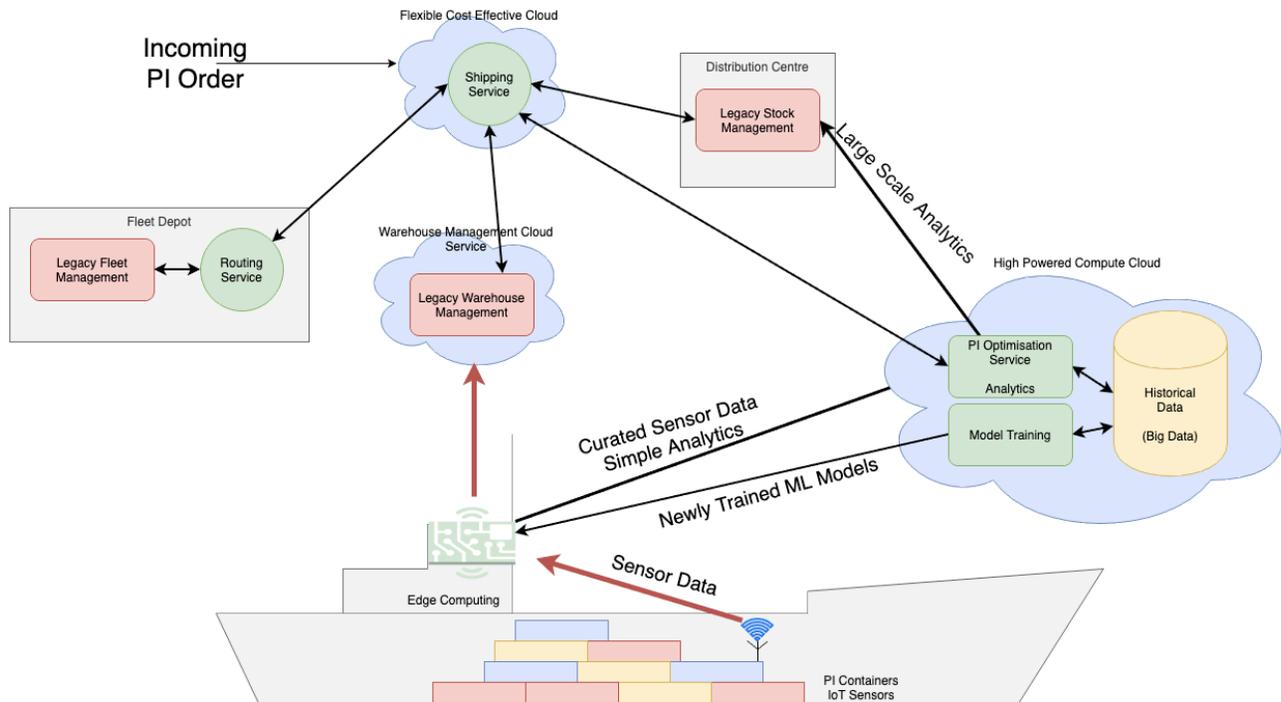


Figure 20 - Multi Cloud PI Example

In Figure 20 we propose a possible scenario in the future of PI and how it relates to current trends in cloud and edge computing. This example focuses on a logistics entity or node with a number of operational requirements. During the initial adoption of PI it is likely that existing legacy IT systems will still play a major role in the logistics operations of this organisation. In this scenario the logistics organisation uses three in-house systems – a Warehouse Management System, a Fleet Management System and a Stock Management System. The three systems share information and coordinate as needed and these systems are often deployed (but not always deployed) at the premises where they are used – for example the fleet management system is running on a server at the depot where the fleet operates out of but the warehouse management system is a cloud based software solution. In addition, the logistics organisation is leveraging the PI and has deployed its own tailor customised Shipping Service which serves as the entry point of this organisation into the PI world. They have also deployed the routing service and are engaging the PI Optimisation analytics service. From Figure 20 there are some points to note:

- The shipping service has been deployed into a simple, cost effective cloud vendor as the compute requirements and resources needed for the shipping service are low. It should be easy to update and manage it and require little overhead and be as cost effective as possible
- The Routing Service was deployed in an on-premise setting because the fleet management system is deployed at the depot and the two services are closely coupled and integrated.
- The Warehouse management system is an out-of-the-box SaaS offering that operates entirely in the cloud and is provided by a 3rd party.
- The PI Optimisation Service performs much more compute intensive and expensive operations, so is deployed in a cloud vendor suited to this
- There are IoT Sensors on shipping containers that report a number of metrics at all times, but they have no memory for resource optimisation reasons.
- However, many modern trucks, trains, ships etc have edge devices to receive sensor readings and some initial analysis and data curation of the sensor data takes place here.

- The PI Optimisation service consumes large scale data from multiple edge devices across multiple PI movers which fuel analytics

This example highlights the many and varied likely communication lines in this scenario – there is almost every conceivable data path occurring. Cloud vendors to other cloud vendors, cloud vendors to private clouds, legacy systems to cloud systems, edge devices to cloud devices, sensors to edge devices and so on. Centralised management of the systems, communication and orchestration of the applications within these systems is an extremely complex challenge to overcome.

6.2 A Centralised Multi-Cloud Management System

As the multi-cloud paradigm is still somewhat recent there are a relatively small number of emerging multi-cloud management solutions. Because this is a recent trend it's difficult to identify a single industry leader. Indeed, there are solutions provided by IBM, Redhat, Cisco, Dell, Citrix Rackware and many more. For this deliverable we elected to focus on Redhat as Redhat technologies also underpin a lot of existing open source and PaaS frameworks discussed in this document. Redhat provide a number of tools that each play a role in addressing the issues above. It should be noted that the Redhat (or any other) multi-cloud management tools should be considered as tools to work in tandem with existing cloud vendor PaaS systems. However, before discussing multi-cloud management solutions it is necessary to re-examine cloud vendors and some private cloud tools from a new angle.

6.2.1 Public & Private Cloud APIs and CLIs

In this document so far, we have seen a number of screenshots showing the management interfaces for AWS tools and services. However, the GUI (Graphical User Interface) is not the only means to control and configure these services. AWS also provides a CLI (Command Line Interface) and API access. The CLI allows individual users to make changes and configurations, create and destroy assets and more via a text interface only. This is sometimes more desirable than using the GUI as it maybe be faster or easier if there is a lot of configuration text to be entered. However more significantly the CLI tool can be used by automation scripts or tools. Dev-ops engineers can write scripts in a language of their choice to automate certain actions using the CLI tool.

API access performs much the same role, except it is designed only to be used programmatically or by automation tools, not manually by a user. Both approaches use secure authentication as standard (username and password for the CLI, tokens for the API) and are widely used and well documented. These tools are provided by all the leading cloud vendors (Google, Microsoft, IBM, Amazon, etc.) but they are also often provided by private cloud frameworks such as Openstack, VMware, Kubernetes, etc. Many tools and products that perform other functions also provide CLI or API access (or both) such as networking systems (firewalls, switches), load balancers (F5), storage clusters (Ceph) and more.

With so many interfaces for so many different products and services it is possible to create complex overarching automation but there is a heavy overhead to creating, managing and maintaining these frameworks. However, much of the work has already been done by Redhat Ansible.

6.2.2 Redhat Ansible

Ansible is an automation framework that operates through the command line using “playbooks”. A playbook is simply a series of steps to be completed in a specific order. Playbooks are text files that indicate requirement before a step is taken, how to take that step and then a way to validate the step was successful before moving on. Ansible has a vast catalogue of libraries that allow it to interface with a huge number of products and services. Everything from local systems to cloud vendors are available. An automation task could be as simple as creating

a folder and moving a file to as complex as creating a virtual private cloud and associated networks in AWS and provisioning 100 VMs (Virtual Machines) into it.

There are many pre-existing playbooks to perform commonly performed actions (such as the ones mentioned above) but if a playbook does not exist it can be created by a dev-ops engineer or adapted from an existing playbook.

While Ansible is excellent for performing automated and complex tasks at scale, it is not designed for ongoing maintenance and visibility. This role is fulfilled by RedHat CloudForms.

6.2.3 Redhat Cloudforms

CloudForms makes use of the existing API and CLI access routes to public and private cloud tools as well as the automation frameworks provided by Ansible to give a centralised management platform for a multi/hybrid cloud environment. This platform gives centralised visibility of all resources in use across all resource providers as well as facilitating creation, control and configuration of PaaS services and compute resources.

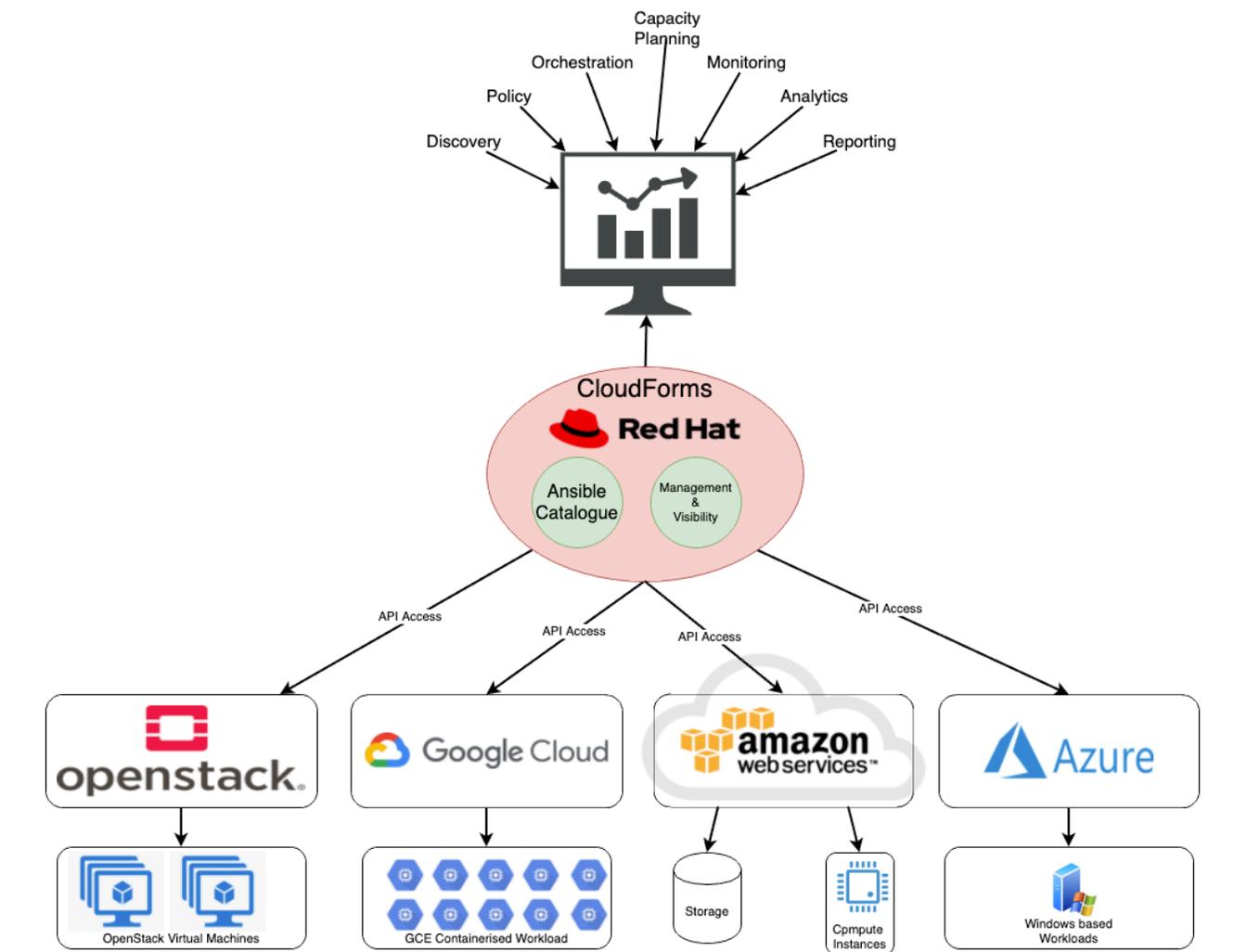


Figure 21 - Redhat Cloudforms

As we can see in Figure 21 CloudForms provides an interface to a number of different public and private cloud providers through API access, which allows for a flexible range of different workload types and resource types. In this instance CloudForms is managing traditional VMs from Openstack, a containerised workload from GCE,

pay-per-use dynamic services from AWS and MS Windows based workloads from Azure. The management interface for CloudForms allows for a number of features:

- Insight – This gathers information about resources across all providers to give a single point view of the status of workloads, and offers recommendations based on status
- Control – This allows for orchestration of workloads across multiple providers. Each cloud vendor will of course provide its own solution for workload orchestration (this allows for automatic scaling, healing etc.) but CloudForms enables orchestration across providers, allowing workloads to be moved towards or away from certain providers depending on the workload requirements.
- Automate – Complex automated workflows can be created that leverage multiple providers in cooperation to create, configure or destroy resources and design workloads
- Integration – CloudForms can be integrated with a number of existing industry standard automation tools and API interfaces for significant customisation

6.2.4 Multi/Hybrid Cloud Networks

While CloudForms will provides a centralised manner in which to configure resources across multiple resource providers, it is worth noting that it will not automatically allow for those providers to communicate with each other, and without this communication path the power of CloudForms or any multi-cloud environment is highly limited. Therefore, it is necessary to formulate secure communication paths between environments on different cloud providers.

SDN (Software Defined Networking) provides a universally adoptable approach for this. As we've seen in the Integration Strategies & Approaches section in this deliverable and also in the intermediate version of this document (D2.20), Software Defined Networks have been utilised in the creation of the Proof of Concept (PoC) environment for ICONET. This is based largely on the concept of the Virtual Private Cloud (VPC) which is a private network space that administrators can create to contain whichever resources and assets they wish to deploy. The VPC can be further configured into different zones and virtual networking functions are used to create subnets, switches, routers, NAT gateways, public gateways, DNS servers and so on. VPCs are highly secure and both public and private access routes can be carefully configured to maintain this security.

For two private cloud environments to communicate directly there are a number of options available but there are two common approaches. The first being public API access. This has been discussed in the **Integration Strategies & Approaches** section, and involves APIs that are open to the public internet that provide access to functions or resources within the environment. The advantage to this is that there is minimal configuration required, but the disadvantage is that any public facing interface comes with some security risks. Therefore, sturdy security and authentication measures are needed to secure any public facing API. Most cloud vendors will provide these functionalities, private cloud environments may need more manual configuration. The alternate, more secure option to use VPN (Virtual Private Network) connections. In the intermediate version of this deliverable (D2.20) VPNs were discussed as the manner in which developers created secure connections to the VPC for the PoC from their laptops or desktops. However, VPNs can also be used for Site to Site connections, allowing for a number of VPCs across multiple cloud vendors to be linked in a secure manner. Depending on the configuration the connection between networks will appear seamless to the resources and assets within these networks, and this can operate in tandem with management systems such as CloudForms to ensure true cooperative multi-cloud workloads.

6.2.5 Combining SDN, Ansible, CloudForms for a Multi-cloud PI Scenario

Returning to the example in Figure 20 above but focussing on the IoT data streams and processing elements we can examine the creation and configuration of the multi-cloud approach needed. We can break the setup into some distinct steps the first of which mirrors the PoC environment creation and configuration. This entails creating the VPC and associated network functions and configuring them, deploying the services and then configuring access through public interfaces and VPNs. Ansible playbooks already exist for many of these steps, but some do not and will need to be written. However, this is not complex, as Code Snippet 1 below shows.

```
- name: create VPC
  ec2_vpc_net:
    name: "{{ vpc_name }}"
    cidr_block: "{{ vpc_cidr }}"
    region: "{{ region }}"
    state: present
    aws_access_key: "{{ aws_access_key }}"
    aws_secret_key: "{{ aws_secret_key }}"
  register: vpc
```

Code Snippet 1 - Ansible VPC Example

Ansible uses the AWS API to create the VPC as configured, the user does not have to use the API themselves, simplifying the approach significantly. Other systems and deployments are created and configured using similar approaches to above. CloudForms allows for the creation and maintenance of a catalogue of Ansible playbooks that will perform commonly required actions and can manage these in a centralised manner.

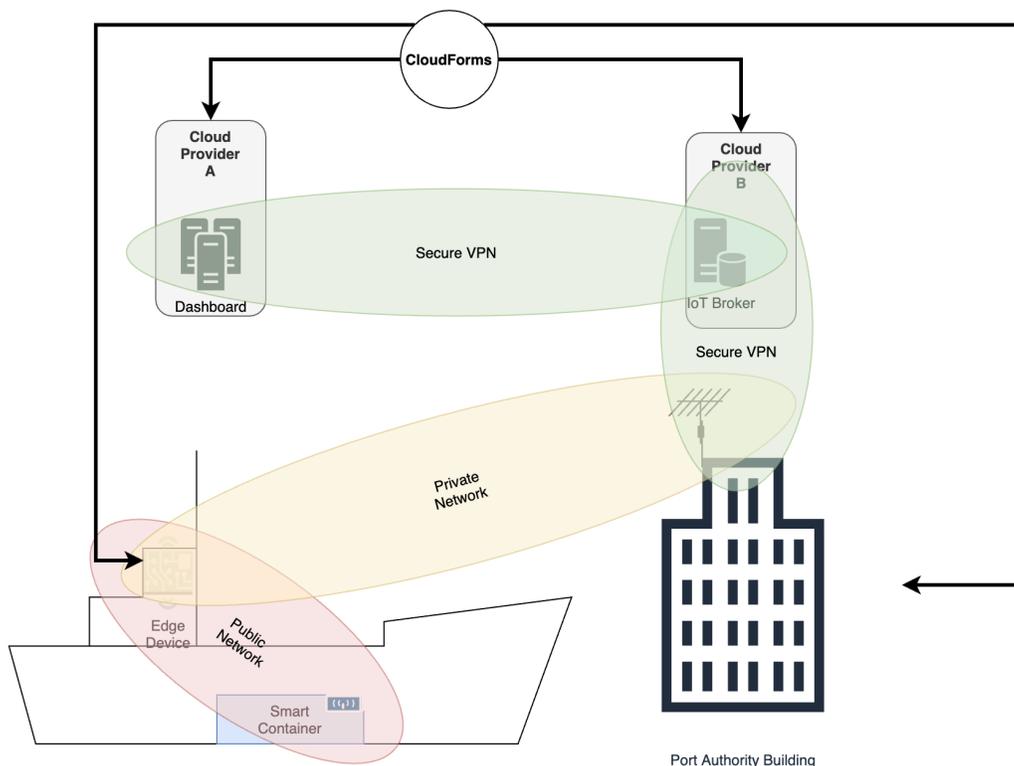


Figure 22 - Multi-cloud PI Network Scenario

Figure 22 shows the scenario from Figure 20 with a focus on the IoT data streams. In this scenario the IoT sensors in smart containers will connect to a local network if it exists – in this case the one on the ship. This connection is a public network, as all containers will connect to the same network. The network on the ship is maintained by an edge device which captures the sensor data for initial pre-processing, analysis or curation. This data can be forwarded via a private network through local connectivity (for example the port authority). This is a private network as the edge devices will only communicate with port authority networks through prior arrangement. The IoT data is forwarded to cloud provider B, chosen for their competitive rates for IoT brokerage. This is done via a secure VPN and finally the data can be accessed and visualised by an application deployed in Cloud Provider A as they have competitive rates for simple front-end application hosting. Both cloud provider networks are joined by a VPN also. CloudForms manages each entity through its own independent interface, and each of the 4 networks were created and are managed by CloudForms (using Ansible) but are used as communication pathways between cloud environments.

7 PoC Integration Environment Blueprint – Full Context

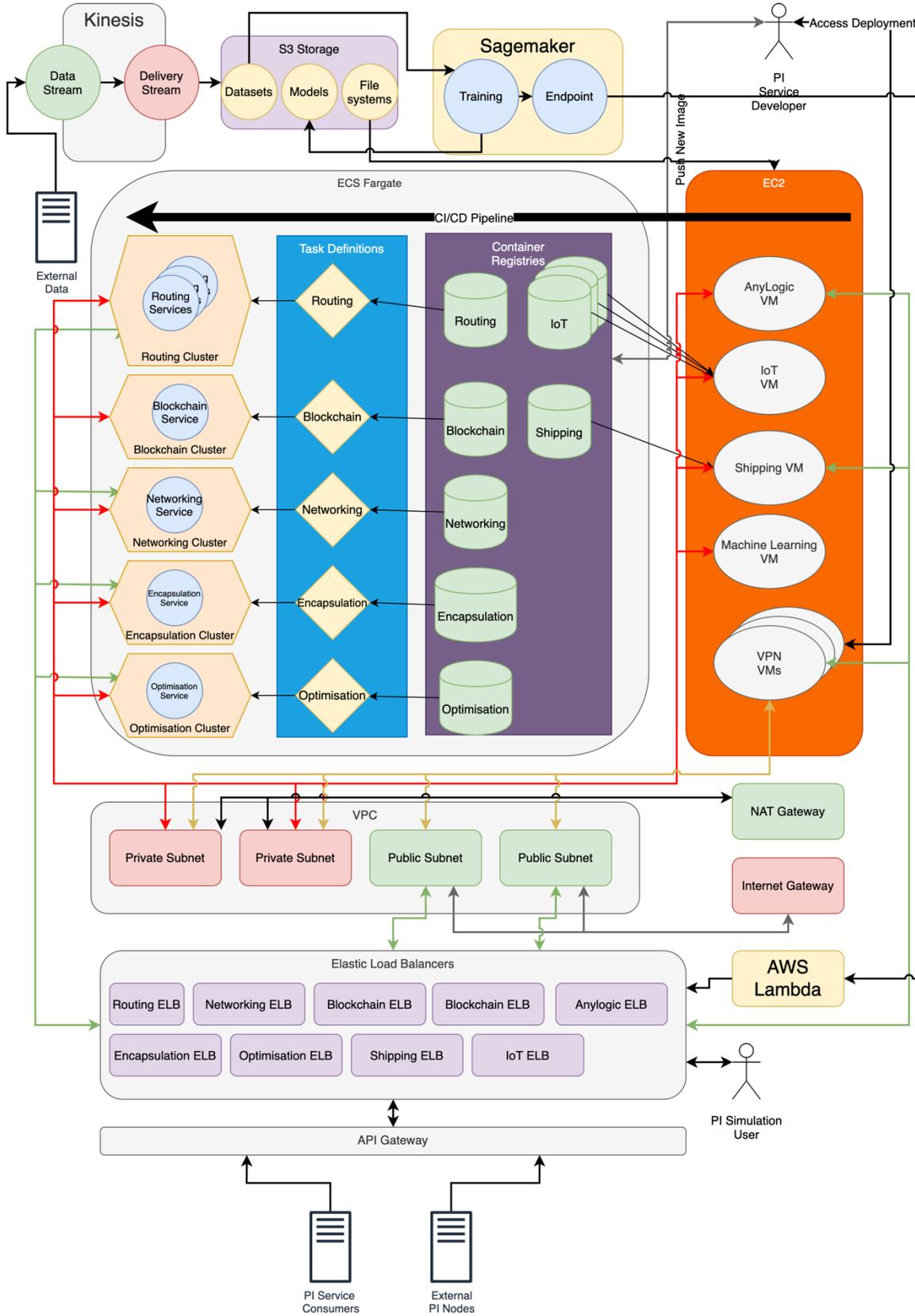


Figure 23 - PoC Blueprint

Figure 23 captures the technical components and component relationships in the PoC Integration Environment in the full context of versions one, two and three of this deliverable in a single diagram. All major technical assets (some elements were left out for clarity of the diagram) and services are represented in Figure 23 but have also been addressed and discussed in detail in the three documents. Figure 23 serves to show the “bigger picture” of how all the technical approaches and systems come together to form the PoC Integration Platform.

D2.19 (Version one) discussed:

- Based on the DoA, Defining Goals & Function of the PoC Environment
- The merits of PaaS as a service
- The approach for choosing a cloud vendor
- The variables and constraints needing consideration for the choice of cloud vendor
- Cloud vendor choice made & justifications for that choice
- Initial PoC Environment creation steps
- Potential Connectivity Scenarios for the PoC Environment

The outputs of that deliverable manifest in Figure 23 and this document by the choice of AWS as the cloud vendor, the substantial use of PaaS systems in the PoC, the external connectivity points to developers, external systems, external PI Nodes, and simulation users. Furthermore, the initial goes as set out in section 3.1 of D2.19 were addressed successfully as shown in Table 1 mapping the goals from D2.19 to the achievements as seen in D2.21.

Table 1: PoC Integration Platform Goals

	PoC Integration Platform Goals (D2.19, 3.1, Table 2)	Goals Addressed
1	Support the research and development efforts in WP2 by providing any tools, frameworks, network connectivity, user access and IaaS/PaaS infrastructure as required.	User access and secure connectivity provided, security policies created, code repositories created for source-controlled development
2	Provide an integration platform upon which PI services can be developed, deployed and tested.	All PI Services were deployed, tested and redeployed with subsequent versions within the PoC environment, with increasing use of automation as the project progressed
3	Support the interconnectivity between the PI services. Ensure that all communication ports and protocols are supported and implemented as needed by the technical requirements of the project.	VPCs, subnets, routing tables and firewall policies ensured that all PI Services communicated with each other and the simulation service in a controlled secure environment. When required, specific access was granted externally via load balancers facilitating Living Lab partners access, external PI Node access, simulation user access.
4	Support the interconnection between the PI simulation environment and the PI services, IoT elements, and Blockchain as necessary.	Similar to above, the VPCs, subnets, routing tables, firewalls, load balancers etc granted connectivity to these systems in the same means as the PI Services

5	Provide integrated and controlled remote user access to PoC integration platform.	Remote access for developers was provided via secure VPN to the VPC, console access via the AWS web UI, command line access via the AWS CLI tool.
6	The integration platform should be flexible and elastic when responding to the technical demands of the project.	The project facilitated and carried rapid and significant redesign based on the needs of the project and all requirements from the PI Service developers were met in a timely manner
7	The integration platform should always be available.	The PoC Integration platform experienced no outages or downtime during the course of the project.

D2.20 (Version two) discussed:

- Restating the goals of the PoC Integration Environment
- The interdependencies of the PI Architecture and the PoC Integration Environment
- Integration Strategies and practical approaches
- The main resource and services (such as compute and storage) that would support the PI Services & other services within the PoC environment.
- Deployment methods and approaches
- Reports on initial assets deployed
- Reports on pilot integrations
- Report on Simulation Service Deployment
- PI Services Dataflow

The outputs of D2.20 can be identified in Figure 23 by noting the integrations and data flows across multiple PI Services, simulations, external actors, data sources and external systems. In addition, compute and storage systems such as S3, Fargate, ECS and EC2 play a central role in the deployment PI Services and other services.

D2.21 (Version three & this document) discusses:

- More complex integration structures such as
 - Elastic Load Balancers
 - Static Addresses
 - Public Access to PI Service APIs
 - API Gateways, authentication and security
 - Deployment pipelines
 - Machine Learning
 - Data acquisition
 - Training
 - Model management
 - API Gateways & AWS Lambda
 - Machine learning based APIs

The outputs of this document, D2.21, can be identified in Figure 23 by noting the deployment pipeline across all ECS systems and deployed services, the load balancers that serve as public interfaces to PI Services where legacy systems, external PI nodes and testing and evaluation can take place, the API gateway that fronts these load balancers for authentication purposes and their relationship to existing SDN structures such as public and private subnets in the VPC and NAT and Internet Gateways. In addition the machine learning pipeline can be identified

near the top of the diagram showing data acquisition, model training and storage and finally AWS lambda and API Gateways to allow machine learning services to be available using PaaS.

8 Conclusion

The research and technical implementations undertaken for these deliverables and tasks have successfully met the current needs of the ICONET project. In addition, this document has begun to address the needs of future PI engagements including anything from further research, to pilot PI programs to full production level PI engagements. The containerisation approach is considered vastly superior to other approaches for service development due to the homogenous nature of development work and flexibility provided. Containerisation allows for a reusable series of deployment requirements across all PI Services. The PI Services stack clearly aligns with the current trend in microservices based architectures which allows the PI Services to benefit from orchestration technologies and emerging serverless compute approaches.

The use of PaaS as the supporting cloud and deployment frameworks in conjunction with containerisation and microservices approaches matched well with the PI Service stack and should be an approach that is followed for future PI research efforts. By limiting containerised services to core PI Service functionality this both offloads non-functional requirements away from PI Service developers, but also allows for a unified approach to non-functional issues (security, deployment etc.). Given emerging trends in cloud technology, the ability for all services to be centrally supported and integrated across multiple cloud vendors is critical.

The use of machine learning technologies is recognised to be an approach that will be extensively adopted in the PI and logistics domain given the huge benefits that could be derived from it. Strategies and technologies such as robust data streams, cloud side pre-processing, the use of PaaS systems, service mesh, and CI/CD pipeline integrations must be adopted to make efficient use of these technologies in harmony with existing technological approaches.

The newest trend in cloud computing paradigms is to adopt a multi-cloud or hybrid cloud approach seeing workloads spread across any number of private and public clouds across a number of technology providers. Given the diverse array of actors, assets and systems in a PI scenario it is likely the bleeding edge of the PI will intersect with the bleeding edge of cloud computing. It is critical to understand these trends now, adopt technologies and practices early to integrate these two concepts to encourage PI adoption at the greatest possible rate. Emerging paradigms such as multi-cloud and fog computing will change how organisations approach their IT solutions, and not only will the PI adapt to these paradigms, but it will excel because of them.

The PoC Integration task and associated deliverables aimed to provide a strategy for collaborative research as well as a technological approach to facilitate research and integration for PI Services and other assets. The PoC platform supported research in the PI domain, aided in evolving the PI reference architecture, development of PI services, supported deployment and integration of all PI Services and simulation services and provided practical approaches for the present as well as state of the art approaches for the future which will ensure the PI Services meet both functional and non-functional requirements.

Annex I – Accessing the PoC Environment for Evaluation

Any authorised party outside the consortium who wishes to access the PoC integration environment hosted in AWS can do so by making contact with ILS, the project coordinator, to arrange secure access via VPN or AWS login credentials. Anyone accessing the PoC environment will need some dev-ops or cloud computing skills to successfully navigate the assets, services and systems within as this is technical platform not designed for end users.

The Anylogic Simulation service can be accessed from a browser at <http://172.31.6.36/auth/login> . Access requires a login name and password which INV will provide and this interface does not require an IT, dev-ops or cloud engineering background.

Annex II –PoC Integration Environment Screenshots

The screenshot shows the AWS CodePipeline console. The left sidebar is titled 'CodePipeline' and includes sections for 'Source', 'Artifacts', 'Build', 'Deploy', 'Pipeline', 'Getting started', 'Pipelines', and 'Settings'. The main content area is titled 'Pipelines' and contains a search bar, a refresh button, a 'Notify' button, and buttons for 'View history', 'Release change', 'Delete pipeline', and 'Create pipeline'. Below these is a table listing various pipelines.

Name	Most recent execution	Latest source revisions	Last executed
routing	Succeeded	Image – sha256:e: Source – 484447f: Added task definition files	1 month ago
iot-parser-api	Succeeded	Source – bdd6f77e: Optimization [atomic operations]	2 months ago
iot-device-parser	Succeeded	Source – 35cc78a8: Add dependencies from amazon	6 months ago
iot-service-storage	Succeeded	Source – 6baa2649: Updated proto package	2 months ago
iot-service-proto	Succeeded	Source – 960df78f: Adding timeout parameter	2 months ago
shipment-service-api	Succeeded	Source – 9f236dbf: Fixing metadata message format	15 days ago
shipment-service-storage	Succeeded	Source – 33127d74: Fixing metadata message format	28 days ago
shipment-service-proto	Succeeded	Source – 7dcc6e29: Fixing metadata message format	24 days ago
iconet-database-driver	Succeeded	Source – c9628c92: Update aggregation test	29 days ago
iconet-python-poetry-image	Succeeded	Source – 6b0aa8c9: Initial commit	6 months ago

The screenshot shows the AWS CodeCommit console. The left sidebar is titled 'CodeCommit' and includes sections for 'Source', 'Getting started', 'Repositories', 'Approval rule templates', 'Artifacts', 'Build', 'Deploy', 'Pipeline', and 'Settings'. The main content area is titled 'Repositories' and contains a search bar, a refresh button, a 'Notify' button, and buttons for 'Clone URL', 'View repository', 'Delete repository', and 'Create repository'. Below these is a table listing various repositories.

Name	Description	Last modified	Clone URL
ICONET-CLMS-shipping	This repo contains the implementation of the Shipment Service API.	1 day ago	HTTPS SSH HTTPS (GRC)
ICONET-ILS-Blockchain	This repository contains the ICONET Blockchain component which is based on the Tendermint DLT and the REST API for interacting with the Blockchain component	5 days ago	HTTPS SSH HTTPS (GRC)
ICONET-NGS-shipment-service-api	This repo contains the implementation of the Shipment Service REST API.	15 days ago	HTTPS SSH HTTPS (GRC)
ICONET-NGS-shipment-service-proto	This repo contains the implementation of the interaction between the Shipment Service API and the Shipment Storage Server.	24 days ago	HTTPS SSH HTTPS (GRC)
ICONET-NGS-shipment-storage-service	Contains the implementation of the Storage Service for the Shipment Service interaction. It offers the API that the Shipment Service can use to manages their informations: Shipments and Data belonging to those shipments.	28 days ago	HTTPS SSH HTTPS (GRC)
ICONET-NGS-database-driver	This repo contains the interconnecting layer between all the system components and the database.	29 days ago	HTTPS SSH HTTPS (GRC)
route-pipe	-	1 month ago	HTTPS SSH HTTPS (GRC)
ICONET-CLMS-encapsulation	This repo contains the implementation of the Encapsulation Service API.	2 months ago	HTTPS SSH HTTPS (GRC)
ICONET-NGS-database-model	Contains all the classes representing the DB collections' entries of the ICONET project	2 months ago	HTTPS SSH HTTPS (GRC)
ICONET-NGS-iot-parser-api	This repo contains the implementation of the Tracker Parser API. It is a Tornado based REST API for parsing the messages coming from the trackers.	2 months ago	HTTPS SSH HTTPS (GRC)
ICONET-NGS-iot-storage-service	This repo contains the implementation of the Storage Service used by the Device Manager component of the Iconet project. It offers the API that the Device Manager can use to manage its informations: Write TrackerData and retrieve	2 months ago	HTTPS SSH HTTPS (GRC)

EC2 > Target groups

Target groups (16)

Filter resources by property or value

<input type="checkbox"/>	Name	ARN	Port	Protocol	Target ...	Load b...	VPC ID
<input type="checkbox"/>	anylogic-grp	arn:aws:elasticload...	80	HTTP	Instance	anylogic-lb1	vpc-e1626987
<input type="checkbox"/>	bca-lb	arn:aws:elasticload...	7777	HTTP	IP	-	vpc-e1626987
<input type="checkbox"/>	blockchain-svc	arn:aws:elasticload...	80	HTTP	IP	blockchai...	vpc-e1626987
<input type="checkbox"/>	ecs-blocke-blockchain-svc	arn:aws:elasticload...	80	HTTP	IP	-	vpc-e1626987
<input type="checkbox"/>	ecs-networ-networking-svc-lb	arn:aws:elasticload...	80	TCP	IP	networkin...	vpc-e1626987
<input type="checkbox"/>	ecs-routin-routing-lb-test	arn:aws:elasticload...	80	TCP	IP	routing-lb	vpc-e1626987
<input type="checkbox"/>	encap-tgt	arn:aws:elasticload...	80	TCP	Instance	encapsula...	vpc-e1626987
<input type="checkbox"/>	nginx	arn:aws:elasticload...	80	HTTP	IP	-	vpc-e1626987
<input type="checkbox"/>	opt-tgt	arn:aws:elasticload...	80	TCP	Instance	optimisati...	vpc-e1626987
<input type="checkbox"/>	routing-ili3-public	arn:aws:elasticload...	80	HTTP	IP	routing-ll...	vpc-e1626987
<input type="checkbox"/>	routing-ili3-tgt	arn:aws:elasticload...	80	HTTP	IP	ili3-routin...	vpc-e1626987
<input type="checkbox"/>	routing-pipe-2	arn:aws:elasticload...	5000	HTTP	IP	routing-pi...	vpc-e1626987
<input type="checkbox"/>	routing-pipelined-tgt	arn:aws:elasticload...	5000	HTTP	IP	routing-pi...	vpc-e1626987
<input type="checkbox"/>	shipping-tgt	arn:aws:elasticload...	80	TCP	Instance	shipping-lb	vpc-e1626987
<input type="checkbox"/>	tmp	arn:aws:elasticload...	80	HTTP	Instance	-	vpc-e1626987
<input type="checkbox"/>	tmp-target	arn:aws:elasticload...	80	HTTP	Instance	routing-pi...	vpc-e1626987

Create Load Balancer Actions

Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Name	DNS name	State	VPC ID	Availability Zones	Type	Created At	Monitoring
<input checked="" type="checkbox"/>	anylogic-lb1	anylogic-lb1-866382980.eu...	active	vpc-e1626987	eu-west-1a, eu-west-1b	application	July 8, 2020 at 4:15:09 PM ...	<input checked="" type="checkbox"/>
<input type="checkbox"/>	blockchain-lb-pub	blockchain-lb-pub-84972495...	active	vpc-e1626987	eu-west-1a, eu-west-1b...	application	October 18, 2020 at 1:19:22 ...	<input type="checkbox"/>
<input type="checkbox"/>	encapsulation-lb	encapsulation-lb-71c5aa610f...	active	vpc-e1626987	eu-west-1a, eu-west-1c	network	September 11, 2020 at 4:56:...	<input type="checkbox"/>
<input type="checkbox"/>	ili3-routing-lb	internal-ili3-routing-lb-189762...	active	vpc-e1626987	eu-west-1a, eu-west-1c	application	September 18, 2020 at 4:32:...	<input type="checkbox"/>
<input type="checkbox"/>	networking-lb	networking-lb-a73065bc2f58...	active	vpc-e1626987	eu-west-1a, eu-west-1c	network	September 9, 2020 at 11:07:...	<input type="checkbox"/>
<input type="checkbox"/>	optimisation	optimisation-764b95959d92...	active	vpc-e1626987	eu-west-1c, eu-west-1a	network	September 11, 2020 at 4:58:...	<input type="checkbox"/>
<input type="checkbox"/>	routing-lb	routing-lb-6b4025ed7d5a19a...	active	vpc-e1626987	eu-west-1a, eu-west-1c	network	September 8, 2020 at 11:15:...	<input type="checkbox"/>
<input type="checkbox"/>	routing-ili3-public	routing-ili3-public-103521713...	active	vpc-e1626987	eu-west-1a, eu-west-1b	application	October 14, 2020 at 12:37:5...	<input type="checkbox"/>
<input type="checkbox"/>	routing-pipelined	internal-routing-pipelined-17...	active	vpc-e1626987	eu-west-1a, eu-west-1c	application	September 15, 2020 at 10:5...	<input type="checkbox"/>
<input type="checkbox"/>	routing-pipelined-IP	internal-routing-pipelined-IP...	active	vpc-e1626987	eu-west-1a, eu-west-1c	application	September 15, 2020 at 11:0...	<input type="checkbox"/>
<input type="checkbox"/>	shipping-lb	shipping-lb-4812eb69e88e7f...	active	vpc-e1626987	eu-west-1a, eu-west-1c	network	September 11, 2020 at 4:55:...	<input type="checkbox"/>

Load balancer: anylogic-lb1

Description | Listeners | Monitoring | Integrated services | Tags

Basic Configuration

Name: anylogic-lb1

ARN: arn:aws:elasticloadbalancing:eu-west-1:331447178759:loadbalancer/app/anylogic-lb1/d43e7c53f1730e07

DNS name: anylogic-lb1-866382980.eu-west-1.elb.amazonaws.com (A Record)

State: active

Type: application

Scheme: internet-facing

IP address type: ipv4 (Edit IP address type)

VPC: vpc-e1626987

Availability Zones: subnet-039dbaf281a56fa77 - eu-west-1a (IPv4 address: Assigned by AWS)

aws Services

New EC2 Experience

Create Volume Actions

Filter by tags and attributes or search by keyword

Name	Volume ID	Size	Volume Type	IOPS	Snapshot	Created	Availability Zone	State	Alarm Status	Attachment Information	Monitoring	Volume
	vol-0eb64e9...	30 GiB	gp2	100	snap-0b44d47...	September 17, 202...	eu-west-1b	in-use	None	i-02e10b659a3c312a...		Ok
	vol-00c096d8...	30 GiB	gp2	100	snap-0b44d47...	September 2, 2020 ...	eu-west-1b	in-use	None	i-08bcd85e1b7544e8...		Ok
	vol-0487546...	8 GiB	gp2	100	snap-043d097...	July 30, 2020 at 10:...	eu-west-1b	in-use	None	i-080d7adb9f9536997...		Ok
	vol-00d7326...	50 GiB	gp2	150	snap-0c27652...	July 29, 2020 at 1:1...	eu-west-1b	in-use	None	i-07d52b2ab61aba17...		Ok
	vol-0ce431e...	16 GiB	gp2	100	snap-043d097...	July 7, 2020 at 7:57...	eu-west-1b	in-use	None	i-002483f7a9a70ab0...		Ok
	vol-0149d33...	8 GiB	gp2	100	snap-091b097...	May 7, 2020 at 11:4...	eu-west-1a	in-use	None	i-03707a268d14093d...		Ok
	AnyLogic OS Disk (Upgraded)	200 GiB	gp2	600	snap-0e63a2c...	March 10, 2020 at 3...	eu-west-1a	in-use	None	i-0f60c2f9d22c2b3d...		Ok
	vol-01cd726...	16 GiB	gp2	100	snap-0c53d8e...	January 15, 2020 at...	eu-west-1b	in-use	None	i-0eb01516967665aa...		Ok
	vol-01b47ca...	8 GiB	gp2	100	snap-076d641...	January 14, 2020 at...	eu-west-1a	in-use	None	i-0779085f882be222...		Ok
	AnyLogic OS Disk (Original)	16 GiB	gp2	100	snap-0c53d8e...	November 25, 2019...	eu-west-1a	available	None			Ok
	vol-0bcd75f...	8 GiB	gp2	100	snap-0c53d8e...	November 25, 2019...	eu-west-1c	in-use	None	i-04c84bc8c45acb54...		Ok
	vol-0a8785e...	8 GiB	gp2	100	snap-0c53d8e...	November 25, 2019...	eu-west-1b	in-use	None	i-079eb9010ab1dc87...		Ok
	vol-06abb29...	8 GiB	gp2	100	snap-0c53d8e...	November 25, 2019...	eu-west-1a	in-use	None	i-0453aab8d1b92e9e...		Ok
	AnyLogicServerDisk	30 GiB	gp2	100	snap-0d317b8...	October 22, 2019 at...	eu-west-1b	in-use	None	i-0c32a43334b70eb8...		Ok

Launch EC2 Image Builder Actions

Owned by me Filter by tags and attributes or search by keyword

Name	AMI Name	AMI ID	Source	Owner	Visibility	Status	Creation Date	Platform	Root Device 1	Virtualization
VPN Image	ami-0f03e7debece9d45	331447178759/...	331447178759	Private	available	September 2, 2020 at 11:17:...	Other Linux	ebs	hvm	

Image: ami-0f03e7debece9d45

Details Permissions Tags

AMI ID	ami-0f03e7debece9d45	AMI Name	VPN Image
Owner	331447178759	Source	331447178759/VPN Image
Status	available	State Reason	-
Creation date	September 2, 2020 at 11:17:46 AM UTC+1	Platform details	Linux/UNIX
Architecture	x86_64	Usage operation	RunInstances
Image Type	machine	Virtualization type	hvm
Description	-	Root Device Name	/dev/sda1
Root Device Type	ebs	RAM disk ID	-
Kernel ID	-	Product Codes	marketplace: f2ew2wrz425a1jagnifd02u5t
Block Devices	/dev/sda1=snap-0b44d479bd3efffb:30:true:gp2		

aws Services

New EC2 Experience

Instances (13) Info

Filter Instances

Name	Instance ID	Instance state	Instance type	Status check	Alarm Status	Availability zone	Public IPv4 DNS	Public IPv4
test-working-sn2	i-0453aab8d1b92e9ee	Stopped	t2.micro	-	No alarms	eu-west-1a	-	-
AnyLogic Server	i-0fb0c2ffdd22c2b3d	Running	t2.xlarge	2/2 checks ...	No alarms	eu-west-1a	-	-
shipping-poc	i-0779085f882be222a	Running	t2.micro	2/2 checks ...	No alarms	eu-west-1a	ec2-18-203-98-56.eu...	18.203.98.5
encapsulation	i-03707a268d14093d8	Running	t2.micro	2/2 checks ...	No alarms	eu-west-1a	ec2-3-248-249-187.eu...	3.248.249.1
VPN Server 1	i-0c32a43334b70eb80	Running	t2.small	2/2 checks ...	No alarms	eu-west-1b	ec2-18-202-162-211.e...	18.202.162.
test-working-sn1	i-079eb9010ab1dc87f	Stopped	t2.micro	-	No alarms	eu-west-1b	-	-
cloud-iot	i-0eb01516967665aa2	Running	t2.micro	2/2 checks ...	No alarms	eu-west-1b	ec2-54-171-159-149.e...	54.171.159.
SimpleDeployments	i-002d83f7a9a70ab07	Running	t2.micro	2/2 checks ...	No alarms	eu-west-1b	ec2-176-34-220-166.e...	176.34.220.
MachineLearningG4	i-07d52b2ab61aba17c	Stopped	g4dn.xlarge	-	No alarms	eu-west-1b	ec2-63-333-180-7.eu-w...	63.333.180.7
MachineLearningG4_Blank	i-080d7adb9f95369972	Stopped	g4dn.xlarge	-	No alarms	eu-west-1b	-	-
VPN Server 2	i-08bcd85e1b7544e8b	Running	t2.small	2/2 checks ...	No alarms	eu-west-1b	ec2-54-76-69-200.eu...	54.76.69.20
VPN Server 3	i-02e10b659a3c312ae	Running	t2.small	2/2 checks ...	No alarms	eu-west-1b	ec2-34-242-85-242.eu...	34.242.85.2
test-working-sn3	i-04c84bc8c45acb546	Stopped	t2.micro	-	No alarms	eu-west-1c	-	-

Security Groups (35) Info Actions Create security group

Filter security groups

<input type="checkbox"/>	Name	Security group ID	Security group name	VPC ID	Description	Owner	Inbound rules count	Outbound rules count
<input type="checkbox"/>	-	sg-003b20b04677c271d	routin-4003	vpc-e1626987	2020-09-05T18:09:43....	331447178759	1 Permission entry	1 Permission entry
<input type="checkbox"/>	ITA_mysql	sg-00547a9d0cb94371a	ITA IP Group	vpc-e1626987	ITA IP Group - mysql	331447178759	1 Permission entry	2 Permission entries
<input type="checkbox"/>	-	sg-00bfb7b86b374672	testin-9546	vpc-e1626987	2020-09-05T15:59:39....	331447178759	2 Permission entries	1 Permission entry
<input type="checkbox"/>	-	sg-00f5233b9e5ca79b8	nginx--9685	vpc-e1626987	2020-07-02T10:18:09....	331447178759	1 Permission entry	1 Permission entry
<input type="checkbox"/>	-	sg-0126d45e3771c2b3a	blockc-5086	vpc-e1626987	2020-07-07T12:48:10....	331447178759	1 Permission entry	1 Permission entry
<input type="checkbox"/>	-	sg-02780312f25ee792d	opt-security	vpc-e1626987	2020-02-03T11:58:03....	331447178759	1 Permission entry	1 Permission entry
<input type="checkbox"/>	VPN Server Specific	sg-028380bc67e49add5	VPN_Access	vpc-e1626987	For VPN Server	331447178759	14 Permission entries	1 Permission entry
<input type="checkbox"/>	-	sg-02a2fd587d0e10870	GPUgrp	vpc-e1626987	launch-wizard-1 create...	331447178759	2 Permission entries	1 Permission entry
<input type="checkbox"/>	-	sg-034db48981669c7fb	launch-wizard-1	vpc-e1626987	launch-wizard-1 create...	331447178759	1 Permission entry	1 Permission entry
<input type="checkbox"/>	-	sg-0417da71eeb90557f	networ-4140	vpc-e1626987	2020-08-11T10:58:56....	331447178759	2 Permission entries	1 Permission entry
<input type="checkbox"/>	-	sg-04c2b2da6d83128ac	routin-5809	vpc-e1626987	2020-09-08T10:45:48....	331447178759	1 Permission entry	1 Permission entry
<input type="checkbox"/>	-	sg-0562c41c65497a499	bca7777	vpc-e1626987	load-balancer-wizard-...	331447178759	2 Permission entries	1 Permission entry

Elastic IP addresses (1/5) Actions Allocate Elastic IP address

Filter Elastic IP addresses

<input type="checkbox"/>	Name	Allocated IPv4 address	Type	Allocation ID	Associated instance ID	Private IP address	Association ID
<input type="checkbox"/>	AL	54.171.5.2	Public IP	eipalloc-0b5260284e39fb33b	-	172.31.17.22	eipassoc-0b01t
<input type="checkbox"/>	ML	63.33.180.7	Public IP	eipalloc-048b6b0019fad7d82	i-07d52b2ab61aba17c	172.31.27.68	eipassoc-03b4c
<input checked="" type="checkbox"/>	VPN 1	18.202.162.211	Public IP	eipalloc-0b2332a7f545c7905	i-0c32a43334b70eb80	172.31.23.146	eipassoc-00d6;
<input type="checkbox"/>	VPN2	54.76.69.200	Public IP	eipalloc-0878b307482b56f0b	i-08bcd85e1b7544e8b	172.31.23.57	eipassoc-0dbd
<input type="checkbox"/>	VPN3	34.242.85.242	Public IP	eipalloc-061aa7db8fa7a957b	i-02e10b659a3c312ae	172.31.20.200	eipassoc-05ebC

18.202.162.211 Summary Tags

Summary

Allocated IPv4 address 18.202.162.211	Type Public IP	Allocation ID eipalloc-0b2332a7f545c7905	Association ID eipassoc-00d67df3a7df8a7ef
Scope VPC	Associated instance ID i-0c32a43334b70eb80	Private IP address 172.31.23.146	Network interface ID eni-009dddf19c4fbc1ee
Network interface owner account ID 331447178759	Public DNS ec2-18-202-162-211.eu-west-1.compute.amazonaws.com	NAT Gateway ID -	Address pool Amazon

Create route table Actions

Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Name	Route Table ID	Explicit subnet association	Edge associations	Main	VPC ID	Owner
<input type="checkbox"/>	Working-NA...	rtb-076b476623d3c1ac6	2 subnets	-	No	vpc-e1626987 MainVPC	331447178759
<input type="checkbox"/>	WorkingMai...	rtb-92f3c1eb	subnet-097d2941	-	Yes	vpc-e1626987 MainVPC	331447178759

Create subnet Actions

Filter by tags and attributes or search by keyword

<input type="checkbox"/>	Name	Subnet ID	State	VPC	IPv4 CIDR	Available IPv4	IPv6 CIDR	Availability Zone	Availability Zone
<input type="checkbox"/>	ib_public	subnet-039dbaf281a56fa77	available	vpc-e1626987 MainVPC	172.31.48.0/24	248	-	eu-west-1a	euw1-az1
<input type="checkbox"/>	WorkingSubnet1 (Public)	subnet-097d2941	available	vpc-e1626987 MainVPC	172.31.16.0/20	4077	-	eu-west-1b	euw1-az2
<input type="checkbox"/>	WorkingSubnet2 (Private)	subnet-59c5803f	available	vpc-e1626987 MainVPC	172.31.0.0/20	4076	-	eu-west-1a	euw1-az1
<input type="checkbox"/>	WorkingSubnet3 (Private)	subnet-c1f57f9b	available	vpc-e1626987 MainVPC	172.31.32.0/20	4073	-	eu-west-1c	euw1-az3

Your VPCs (1/1) Info

Actions ▾
Create VPC

VPC ID: vpc-e1626987
Clear filters

<input checked="" type="checkbox"/>	Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR	IPv6 pool	DHCP
<input checked="" type="checkbox"/>	MainVPC	vpc-e1626987	Available	172.31.0.0/16	-	-	dopt-7

vpc-e1626987 / MainVPC

Details |
 CIDRs |
 Flow logs |
 Tags

Details

VPC ID vpc-e1626987	State Available	DNS hostnames Enabled	DNS resolution Enabled
Tenancy Default	DHCP options set dopt-7c5a201a	Route table rtb-92f3c1eb / WorkingMainRoute	Network ACL act-2f5ad756 / workingact
Default VPC Yes	IPv4 CIDR 172.31.0.0/16	IPv6 pool -	IPv6 CIDR -
Owner ID 331447178759			

Task Definitions

Task definitions specify the container information for your application, such as how many containers are part of your task, what resources they will use, how to

Create new Task Definition
Create new revision
Actions ▾

Status: ACTIVE INACTIVE

	Task Definition	Latest revision status
<input type="checkbox"/>	blockchain	ACTIVE
<input type="checkbox"/>	encap_svc	ACTIVE
<input type="checkbox"/>	iconet-opt	ACTIVE
<input type="checkbox"/>	iconet-routing	ACTIVE
<input type="checkbox"/>	ll3-routing-tsk	ACTIVE
<input type="checkbox"/>	networking-tsk	ACTIVE
<input type="checkbox"/>	routing-pipelined	ACTIVE

Clusters

An Amazon ECS cluster is a regional grouping of one or more container instances on which you can run task requests. Each account receives a default cluster the first time you use the Amazon ECS service. Clusters may contain more than one Amazon EC2 Instance type.

For more information, see the [ECS documentation](#).

[Create Cluster](#) [Get Started](#)

View list card 6 loaded of 6 clusters

Last updated on October 23, 2020 5:02:01 PM (0m ago)

Cluster name	CloudWatch monitoring	Services	Running tasks	Pending tasks	Container instances
routing-ci	✔ Container Insights	4	4	0	0
routing-pipelined	✔ Container Insights	1	1	0	0
blockchain	✔ Container Insights	1	1	0	0
encap-ci	✔ Container Insights	1	1	0	0
optimisation-ci	✔ Container Insights	1	1	0	0
networking-ci	✔ Container Insights	1	2	0	0

aws Services

- Amazon Container Services
- Amazon ECS Clusters
- Task definitions
- Amazon EKS Clusters
- Amazon ECR **Repositories**

ECR > Repositories

Repositories (12)

View push commands Delete Edit **Create repository**

Find repositories

Repository name	URI	Created at	Tag immutability	Scan on push	Encryption type
iconet-blockchain	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet-blockchain	06/24/20, 12:06:25 PM	Disabled	Disabled	AES-256
iconet-clms-encapsulation	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet-clms-encapsulation	04/29/20, 08:17:33 PM	Disabled	Disabled	AES-256
iconet-clms-shipment	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet-clms-shipment	04/08/20, 02:20:11 PM	Disabled	Disabled	AES-256
iconet-networking-svc	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet-networking-svc	04/02/20, 02:08:15 PM	Disabled	Disabled	AES-256
iconet-opt	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet-opt	02/03/20, 11:31:48 AM	Disabled	Disabled	AES-256
iconet-router	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet-router	04/02/20, 10:27:06 AM	Disabled	Disabled	AES-256
iconet-router-ll3	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet-router-ll3	09/18/20, 03:22:55 PM	Disabled	Disabled	AES-256
iconet/ngs/api-manager/shipment-service-api	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet/ngs/api-manager/shipment-service-api	04/28/20, 07:57:03 PM	Disabled	Enabled	AES-256
iconet/ngs/iot-manager/iot-parser-api	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet/ngs/iot-manager/iot-parser-api	04/29/20, 06:21:45 PM	Disabled	Enabled	AES-256
iconet/ngs/python-poetry	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet/ngs/python-poetry	04/23/20, 09:59:39 AM	Disabled	Enabled	AES-256
iconet/ngs/storage-services/iot-service	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet/ngs/storage-services/iot-service	04/29/20, 12:20:49 PM	Disabled	Enabled	AES-256
iconet/ngs/storage-services/shipment-service	331447178759.dkr.ecr.eu-west-1.amazonaws.com/iconet/ngs/storage-services/shipment-service	04/28/20, 02:26:12 PM	Disabled	Enabled	AES-256

Save as...

Recent reports

New report

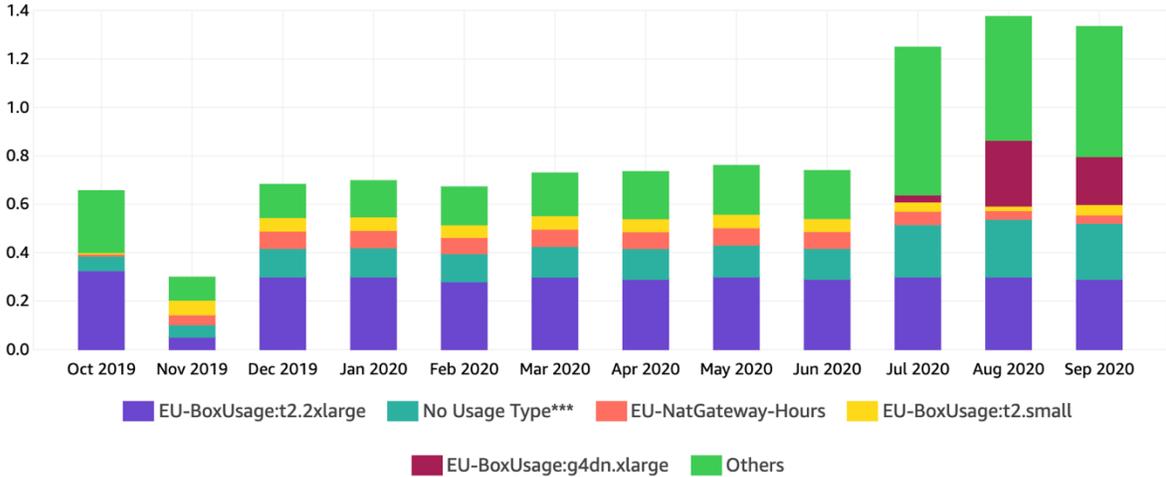
Last 12 Months

Monthly

Stack

Group by: Usage Type x Service Linked Account Region Instance Type Resource Cost Category Tag More

Costs (\$ in thousands)



Identity and Access Management (IAM)

- Dashboard
- Access management
 - Groups
 - Users**
 - Roles
 - Policies
 - Identity providers
 - Account settings
- Access reports
 - Access analyzer
 - Archive rules
 - Analyzers

Add user Delete user

Find users by username or access key

<input type="checkbox"/>	User name	Groups
<input type="checkbox"/>	alex.papageorgiou@inlecomsystems.com	Inlecom
<input type="checkbox"/>	antonis	CLMS
<input type="checkbox"/>	claudio.salvadori@ngs-sensors.it	NGS
<input type="checkbox"/>	dcipres@itainnova.es	ITAINNOVA
<input type="checkbox"/>	faisalgh@ie.ibm.com	Irkin
<input type="checkbox"/>	garanco	IBM
<input type="checkbox"/>	kgmf	IBM
<input type="checkbox"/>	kostas	VLTN
<input type="checkbox"/>	o.panagou@clmsuk.com	CLMS

OPENVPN Access Server v2.7.5

STATUS

CONFIGURATION

- Cluster
- License
- TLS Settings
- Network Settings
- VPN Settings
- Advanced VPN
- Web Server
- CWS Settings
- Failover

USER MANAGEMENT

AUTHENTICATION

TOOLS

Logout

Status Overview

The server is currently ON

[Stop the Server](#)

Active Configuration

Access Server version:	2.7.5
Server Name:	18.202.162.211
License Status:	2 devices
Current Active Users:	1
Authenticate users with:	local
Accepting VPN client connections on IP address:	eth0: 172.31.23.146
Port for VPN client connections:	tcp/443, udp/1194
OSI Layer:	3 (routing/NAT)
Clients access private subnets using:	NAT
Node:	openvpnas2

172.31.6.36/auth/login

anylogic cloud

Log in Sign up Guest

Password

Remember me

[Log in](#)

[Forgot password?](#)

By logging into account, you agree to AnyLogic Cloud's [Terms of use](#) and [Privacy policy](#)

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] "Internet 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>.