D8.4 - SILVANUS platform release, 2nd version

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 101037247

Project Acronym	SILVANUS
Grant Agreement number	101037247 (H2020-LC-GD-2020-3)
Project Full Title	Integrated Technological and Information Platform for Wildfire Management
Funding Scheme	IA – Innovation action

DELIVERABLE INFORMATION

Deliverable Number:	D8.4
Deliverable Name:	Report on SILVANUS final reference architecture
Dissemination level:	PU
Type of Document:	Demo
Contractual date of delivery:	31/05/2024 (M32)
Date of submission:	03/06/2024
Deliverable Leader:	INTRA
Status:	Final
Version number:	V0.4
WPLeader/ TaskLeader:	INTRA/INTRA, UISAV, FINC
Keywords	Integration, SILVANUS system
Abstract	The current document describes the components that have been comprise the SILVANUS system and provides evidence of the integration activities. It accompanies the software code that has been produced and maintained in the SILVANUS Github repository which represents the 2nd version /release of SILVANUS developments.

Deliverable Leader:	INTRA
Lead Author(s)	Despina Anastasopoulos, Nelly Leligou, Fanis Orphanoudakis, Sofia Tsekeridou (INTRA)
Reviewers	RINI, CSIRO, UISAV

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 101037247

Disclaimer

All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose.

The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.

Document History			
Version	**Date**	**Contributor(s)**	**Description**
V0.1	04.3.2024	INTRA	ToC
V0.2	15/5/2024	INTRA, DELL, CTL, ATOS, AMIKOM, CERTH, MGS, HB, RINICOM, VTG, CMCC, WUT, SIMAVI, UISAV	Contributions to all chapters and integration by INTRA
V0.3	23/5/2024	INTRA, DELL, CTL, ATOS, AMIKOM, CERTH, MGS, HB, RINICOM, VTG, CMCC, WUT, SIMAVI, UISAV	Contributions to all chapters and integration by INTRA
V0.4	30/5/2024	INTRA	Revisions based on the comments from the internal and quality reviewers
V1.0	31.5.2024	PEGASO, VTG	Consolidated final deliverable release ready for the submission
V1.1	02/08/2024	INTRA	Revisions following the comments of the reviewers

List of beneficiaries

No	Partner Name	Short name	Country
1	UNIVERSITA TELEMATICA PEGASO	PEGASO	Italy
2	ZANASI ALESSANDRO SRL	Z&P	Italy
3	NETCOMPANY-INTRASOFT SA	INTRA	Luxembourg
4	THALES	TRT	France
5	FINCONS SPA	FINC	Italy
6	ATOS IT SOLUTIONS AND SERVICES IBERIA SL	ATOS IT	Spain
6.1	ATOS SPAIN SA	ATOS SA	Spain
7	EMC INFORMATION SYSTEMS INTERNATIONAL	DELL	Ireland
8	SOFTWARE IMAGINATION & VISION SRL	SIMAVI	Romania
9	CNET CENTRE FOR NEW ENERGY TECHNOLOGIES SA	EDP	Portugal
10	ADP VALOR SERVICOS AMBIENTAIS SA	ADP	Portugal
11	TERRAPRIMA - SERVICOS AMBIENTAIS SOCIEDADE UNIPESSOAL LDA	TP	Portugal
12	3MON, s. r. o.	3MON	Slovakia
13	CATALINK LIMITED	CTL	Cyprus
14	SYNTHESIS CENTER FOR RESEARCH AND EDUCATION LIMITED	SYNC	Cyprus
15	EXPERT SYSTEM SPA	EAI	Italy
16	ITTI SP ZOO	ITTI	Poland
17	Venaka Treleaf GbR	VTG	Germany
18	MASSIVE DYNAMIC SWEDEN AB	MDS	Sweden
19	FONDAZIONE CENTRO EURO-MEDITERRANEOSUI CAMBIAMENTI CLIMATICI	CMCC F	Italy
20	EXUS SOFTWARE MONOPROSOPI ETAIRIA PERIORISMENIS EVTHINIS	EXUS	Greece
21	RINIGARD DOO ZA USLUGE	RINI	Croatia
22	Micro Digital d.o.o.	MD	Croatia
23	POLITECHNIKA WARSZAWSKA	WUT	Poland
24	HOEGSKOLAN I BORAS	HB	Sweden
25	GEOPONIKO PANEPISTIMION ATHINON	AUA	Greece
26	ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS	CERTH	Greece
27	PANEPISTIMIO THESSALIAS	UTH	Greece

No	Partner Name	Short name	Country
28	ASSOCIACAO DO INSTITUTO SUPERIOR TECNICO PARA A INVESTIGACAO E DESENVOLVIMENTO	IST	Portugal
29	VELEUCILISTE VELIKA GORICA	UASVG	Croatia
30	USTAV INFORMATIKY, SLOVENSKA AKADEMIA VIED	UISAV	Slovakia
31	POMPIERS DE L'URGENCE INTERNATIONALE	PUI	France
32	THE MAIN SCHOOL OF FIRE SERVICE	SGPS	Poland
33	ASSET - Agenzia regionale Strategica per lo Sviluppo Ecosostenibile del Territorio	ASSET	Italy
34	LETS ITALIA srls	LETS	Italy
35	Parco Naturale Regionale di Tepilora	PNRT	Italy
36	FUNDATIA PENTRU SMURD	SMURD	Romania
37	Romanian Forestry Association - ASFOR	ASFOR	Romania
38	KENTRO MELETON ASFALEIAS	KEMEA	Greece
39	ELLINIKI OMADA DIASOSIS SOMATEIO	HRT	Greece
40	ARISTOTELIO PANEPISTIMIO THESSALONIKIS	AHEPA	Greece
41	Ospedale Israelitico	OIR	Italy
42	PERIFEREIA STEREAS ELLADAS	PSTE	Greece
43	HASICSKY ZACHRANNY SBOR MORAVSKOSLEZSKEHO KRAJE	FRB MSR	Czechia
44	Hrvatska vatrogasna zajednica	HVZ	Croatia
45	TECHNICKA UNIVERZITA VO ZVOLENE	TUZVO	Slovakia
46	Obcianske zdruzenie Plamen Badin	PLAMEN	Slovakia
47	Yayasan AMIKOM Yogyakarta	AMIKOM	Indonesia
48	COMMONWEALTH SCIENTIFIC AND INDUSTRIAL RESEARCH ORGANISATION	CSIRO	Australia
50	FUNDACAO COORDENACAO DE PROJETOS PESQUISAS E ESTUDOS TECNOLOGICOS COPPETEC	COPPETEC	Brazil
51	Rinicom Ltd	RINICOM	UK

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ACRONYM	DESCRIPTION
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
A&A	Authentication and authorisation
BDF	Big Data Framework
BS	BACKEND SERVICES
CA	Certificate Authority
CASPAR	Structured Data Semantic Exploitation Framework
CCP	Claim Check Pattern
CI/CD	Continuous Integration/Continuous Development
CPU	Central Processing Unit
CRL	Certificate Revocation List
DAG	Directed Acyclic Graph
DBMS	Database Management System
DEM	Digital Elevation Model
DEVOPS	Development and Operations
DL	Deep Learning
DOA	Description of Action
DSL	Domain-Specific Language
DSS	Decision Support System
DTLS	Datagram Transport Layer Security
ECMWF	European Centre for Medium-Range Weather Forecasts
ECS	Elastic Cloud Storage
EMDC	Edge Micro Data Centre
ETL	Extract, Transform, Load
FaaS	Function as a Service
FCC	Forward Command Centre
FiFo	First in First out
FWI	Fire Weather Indices
GIS	Geographic Information System
GPS	Global Positioning System
GRIB	General Regularly distributed Information in Binary form
GW	Gateway
IoT	Internet of Things
JSON	JavaScript Simple Object Notation
KPA	Knative Pod Autoscaling
KPI	Key Performance Indicator
MIME	Multipurpose Internet Mail Extensions
ML	Machine Learning
MVP	Minimum Viable Product
NDVI	Normalised Difference Vegetation Index
NetCDF	Network Common Data Form
PKI	Public Key Infrastructure
PMAS	Poll Management and Aggregation Service
PS	Pilot Site
RAM	Random Access Memory

RART	Resource Allocation of Response Teams
RDF	Resource Description Framework
REST	Representational state transfer
RoI	Region of Interest
ROS	Robot Operating System
SAFE	Standard Archive Format for Europe
SAL	Storage Abstraction Layer
SAR	SYNTHETIC APERTURE RADAR
SCM	Source Code Management
SDR	Software Defined Radio
SLP	Sea Level Pressure
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
TLS	Transport Layer Security
UAV	Unmanned Aerial Vehicle
UC	Use Case
UGV	Unmanned Ground Vehicle
UxVs	UAV and UGV
UI	User Interface
UP	User Product
VM	Virtual Machine
VR	Virtual Reality
WMO	World Meteorological Organisation
WRF	Weather Research and Forecasting Model
XR	Extended Reality

EXECUTIVE SUMMARY

D8.4 is the 4th deliverable of WP8 which essentially represents the second iteration of the platform. This platform is ready for piloting and it consists of the components described in D8.3 which are required to deliver the functionalities of the user products defined in D8.3, with the exceptions of a) components that are implemented under the Open Forest Map and b) components that mainly intend to perform lightweight processing and visualise indices which will be reported in the next release under D8.5. **It should be stressed that while in the initial DoA the current deliverable was foreseen for M36, in the amendment the deadline became M32. This decision was based on the needs of the pilots to have the platform ready before summertime (in Europe).**

This deliverable is of type demonstrator and thus, for each platform component, it provides a short summary, and it points to the relevant location in the SILVANUS GitHub where the software code of the different components and additional information regarding, e.g., the testing and validation of the components also exist.

It is important to note that:

a) A significantly higher number of components has been integrated in the 2nd version to meet as many user requirements as possible and
b) For those components that were included in the 1st version of the platform, the differences in functionality are described in D8.3 (delivered March 2024).

1 Introduction

1.1 Component summary template

In this deliverable which is of type "demonstrator", a summary of information per component included in the SILVANUS platform – version 2 is provided, while further details are provided in the project's GitHub. The template of the presentation of each component is shown in the Table 1, below.

Table 1: Component Information

Title	*This field holds the name of the SILVANUS component*	**WP**	*This field holds the WP that the component belongs*
Description/ Functionality	*This field holds the component's operation description and additional information associating this with the relevant service in D8.3.*		
Repository URL	*The absolute URL of the component's location in the Silvanus GitHub, if this is made publicly available*		
Integration component list	*This field holds the components list that this component interoperates and will integrate with. The number of components in the list can be 0 (if standalone) or other positive value*		
Deployment location	*This field holds deployment location in the Silvanus Cloud, if applicable*		
Container size	*If the component is containerized, then it provides the size of the Container*		
Requirements	*This field holds computational requirements for this component, e.g. CPU, RAM, STORAGE requirements of the component.*		
Contact email	*This field holds the email of the developer of the component.*		

1.2 Components' summary

This section includes the summary of the components currently deployed.

It is worth stressing that in the following tables (Table 2 to Table 41) components that are relevant to services that will be delivered to the users in the 2nd version of the platform are also described. Some of them are already deployed in the SILVANUS cloud and some others are running on infrastructures owned by the consortium partners.

Table 2: Description of the Fire Danger Tool API

Title	*Fire Danger Tool API*	**WP**	*WP4, WP5*
Description/ Functionality	*It implements a REST API service that provides information about* - *Daily Fire Weather Index based on the Canadian FWI* - *Weather forecast for the next 72 hours* - *ML-based Fire Danger Index* ***Relevant to BS1 in D8.3***		

Repository URL	*https://github.com/CMCC-Foundation/fire_danger_tool*
Integration component list	*The product integrates with SAL for querying data and to the dashboard for displaying the fire danger index map for pilot region.*
Deployment location	*CMCC on-premises facilities, Silvanus DockerHub, Silvanus cluster*
Container size	*4GB*
Requirements	*The API is based on Python* *CPU: 4 cores* *MEM: 8GB* *DISK: 25GB*
Contact email	*gabriele.accarino@cmcc.it, shahbaz.alvi@cmcc.it*

Table 3: Description of Twitter Crawler component

Title	*Twitter Crawler*	**WP**	*WP4-T4.4*
Description/ Functionality	*Collects tweets related to wildfires in almost real time from Twitter API based on various search criteria (keywords, accounts)* *Relevant to BS2 (in D8.3)*		
Repository URL	*UP3*		
Integration component list	*Knowledge Base, Dashboards, Fire Events Detection*		
Deployment location	*CERTH server*		
Container size	*1GB*		
Requirements	*Python 3.9* *Python libraries:* *tweepy==4.10.1* *regex==2021.4.4* *python-dateutil==2.8.1* *pandas==1.2.5* *asyncio==3.4.3* *DateTime==4.3* *requests==2.28.1* *urllib3==1.26.6* *pymongo==4.2.0* *aiohttp==3.8.3*		
Contact email	*arbozas@iti.gr, kouloglou@iti.gr, heliasgj@iti.gr*		

Table 4: Description of Visual Concept Extraction Module

Title	*Visual Concept Extraction Module*	**WP**	*WP4-T4.4*

Description/ Functionality	*Accepts a URL of an image as input and returns the top 10 concepts that define the image the best from 186 predefined concepts.* *Relevant to BS2 (in D8.3)*
Repository URL	*https://github.com/silvanus-prj/Visual-Concept-Extraction-Module*
Integration component list	*This module is one of the services used in T4.4 for social media detection (UP3)*
Deployment location	*CERTH server*
Container size	*12GB*
Requirements	*Python 3.9* *Python libraries:* *regex==2021.4.4* *python-dateutil==2.8.1* *pandas==1.2.5* *flask==3.4.3* *DateTime==4.3* *requests==2.28.1* *urllib3==1.26.6* *pymongo==4.2.0* *aiohttp==3.8.3*
Contact email	*arbozas@iti.gr, kouloglou@iti.gr, heliasgj@iti.gr*

Table 5: Description of Location Extraction Module

Title	*Location Extraction Module*	**WP**	*WP4-T4.4*
Description/ Functionality	*Accepts a text of a social media post, detects with NER tagging the placename found in text. Pushes these placenames to OpenStreetMap and takes the precise coordinates of these place names. Finally, it returns the location with coordinates found in the text in JSON format.* *This module works for English, Italian, German, French, Greek, Dutch, Finnish, Spanish languages.* *Relevant to BS2 (in D8.1)*		
Repository URL	*https://github.com/silvanus-prj/Location-Extraction-Module*		
Integration component list	*This module is one of the services used in T4.4 for social media detection (UP3)*		
Deployment location	*CERTH server*		
Container size	*32GB*		
Requirements	*Python 3.9* *Python libraries:* *flair==0.11.3* *Flask==2.1.1* *requests==2.27.1* *transformers==4.18.0*		

	Unidecode==1.3.4 *protobuf==3.19.4* *gr-nlp-toolkit==0.0.3*
Contact email	*arbozas@iti.gr, kouloglou@iti.gr, heliasgj@iti.gr*

Table 6: Description of Relevance Estimation Module

Title	*Relevance Estimation Module*	**WP**	*WP4-T4.4*
Description/ Functionality	*Accepts a text of a social media post and returns if the post text refers to fires.* *Relevant to BS2 (in D8.3)*		
Repository URL	*https://github.com/silvanus-prj/Relevance-Estimation-Module*		
Integration component list	*This module is one of the services used in T4.4 for social media detection (UP3)*		
Deployment location	*CERTH server*		
Container size	*~32GB*		
Requirements	*Python 3.9* *Python libraries:* *scikit-learn==1.3.0* *python-dateutil==2.8.2* *pandas==2.1.4* *tensorflow==2.13.1* *keras==2.13.1* *simpletransformers==0.64.3*		
Contact email	*arbozas@iti.gr, kouloglou@iti.gr, heliasgj@iti.gr*		

Table 7: Description of Wildfire Events Detection Module

Title	*Fire Events detection*	**WP**	*WP4-T4.4*
Description/ Functionality	*Consumes social media posts from Twitter, Facebook and Web crawlers and detect fire event found in these posts.* *Relevant to BS2 (in D8.3)*		
Repository URL	*https://github.com/silvanus-prj/Wildfire-Events-Detection-Module*		
Integration component list	*This module is one of the services used in T4.4 for social media detection (UP3)*		
Deployment location	*Silvanus cloud*		
Container size	*~6GB*		

Requirements	*Python 3.9* *Python libraries:* *pandas==1.2.5* *Haversine == 1.1.0* *pymongo==4.2.0* *requests==2.27.1*
Contact email	*arbozas@iti.gr, kouloglou@iti.gr, heliasgj@iti.gr*

Table 8: Description of Social media sensing image filtering Module

Title	*Social media sensing image filtering*	**WP**	*WP4-T4.4*
Description/ Functionality	*REST API that accepts image URLs as input, processes them with ML algorithms to return the probability of each image depicting irrelevant/unwanted content (e. g., contain inappropriate content).* *Relevant BS2 in D8.3*		
Repository URL	*https://github.com/silvanus-prj/social-media-sensing-image-filtering*		
Integration component list	*This module is one of the services used in T4.4 for social media detection (UP3)*		
Deployment location	*Catalink Server*		
Container size	*~10GB*		
Requirements	*Python3 and Python3 libraries (e.g., tensorflow, opennsfw2, opencv).* *STORAGE: ~15-20MB (~ maximum 5 images per request, of 3MB each, totaling in 15MB with some additional space for the output files)*		
Contact email	*maria.maslioukova@catalink.eu, vangelis.mathioudis@catalink.eu*		

Table 9: Description of Fire and Smoke Detection and Localization in Images Module

Title	*Fire and Smoke Detection and localization in Images*	**WP**	*WP4/WP5*
Description/ Functionality	*REST API that checks image URLs or images with metadata (from SAL or edge devices) whether they contain fire and smoke. For the SAL endpoint it additionally marks the fire's location within the image, using superpixels. The ML detection algorithms are the same ones used in the IoT for fire detection (UP4a).* *Information about UP4a Fire detection from IoT devices can be found here: https://github.com/silvanus-prj/fire-and-smoke-detection-edge-ctl* *Relevant to BS2, BS3 and BS14 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/fire-and-smoke-detection-ctl*		

Integration component list	*NiFi/SAL, Social Media Sensing (UP3), Edge devices (e.g., EMDCs), SILVANUS Dashboard*
Deployment location	*Catalink servers*
Container size	*~15GB for each detection algorithm (so ~30GB in total)*
Requirements	*Python3 and Python3 libraries (e.g., tensorflow, opencv).* *CPU: full utilisation of the available cores (suggested is a minimum of 4 cores)* *RAM: ~3.5GB* *STORAGE: ~15-20MB (~ maximum 5 images per request, of 3MB each, totalling in 15MB with some additional space for the output files)*
Contact email	*maria.maslioukova@catalink.eu, georgiach@catalink.eu, nikolas.petrou@catalink.eu, vangelis.mathioudis@catalink.eu*

Table 10: Description of Resource Allocation of Response Teams (RART)

Title	*UP9.a – Resource Allocation of Response Teams (RART)*	**WP**	*WP5*
Description/ Functionality	*This is the Decision Support System (DSS) algorithm for the resource allocation of firefighter units in the field.*		
Repository URL	*https://github.com/silvanus-prj/resource-allocation*		
Integration component list	*SAL, FSM, dashboard*		
Deployment location	*Silvanus cloud* *Simulation URL: https://resource-allocation.platform.silvanus-project.eu/*		
Container size	*2.5GB*		
Requirements	*RAM 32GB, CPU core i7* *GDAL Ubuntu Base Image* *Libraries:* *gdal* *pika* *requests* *nested_lookup* *rasterio* *shapely* *geopandas* *matplotlib* *ortools* *boto3* *wget* *geojson* *celery*		

Contact email	*Theofanis.Orphanoudakis@netcompany.com*, *Nelly.LELIGOU@netcompany.com*

Table 11: Description of Fire and Smoke Detection in Images on the Edge Module

Title	*Fire and Smoke Detection in Images on the Edge*	**WP**	*WP4*
Description/ Functionality	*This user product detects the presence of fire and smoke in images or videos. The images are taken by UxVs (UAV and UGV) and ingested into the system using the SAL. From the UxVs, the images are sent via "mesh-in-the-sky" or downloaded when the UxV arrive to the Ground Station. The user product reads the images using Rabbit MQ and analyses the images in "soft" real time using computer vision algorithms (ML algorithms) to detect fire and smoke. The result is sent via Rabbit MQ to be displayed in the user dashboard. The analysis is made in the "edge" devices, that is, high-end computers with usually GPU-like capabilities.* *Part of BS4 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/fire-and-smoke-detection-Atos*		
Integration component list	*This componehnt integrates with UP4b*		
Deployment location	*Edge devices*		
Container size	*No container use is foreseen.*		
Requirements	*Python3 and Python3 libraries (e.g., Pytorch, ultralytics, opencv). High demand of computer power on the device. Use of GPU if available.*		
Contact email	*jose.martinezs@eviden.com*		

Table 12: Description of Terrain segmentation from Satellite Module

Title	*Terrain segmentation from satellite*	**WP**	*WP4-WP5*
Description/ Functionality	*This module produces segmentation of the terrain using satellite images as source* *Part of BS4 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/terrain-segmentation-and-super-resolution (No code in the repo for internal policy reasons, only readme uploaded)*		
Integration component list	*This module is part of the tools created in WP4 for satellite using AI. The produced data can be (are) used from multiple other components of SILVANUS platform.*		

Deployment location	*https://github.com/silvanus-prj/terrain-segmentation-and-super-resolution*
Container size	*24,2 Gb virtual, 9.18Mb*
Requirements	*Python environment in a computer with GPU capability*
Contact email	*jose.martinezs@eviden.com*

Table 13: Description of Terrain super resolution for Satellite Images Module

Title	*Terrain super-resolution for satellite images*	**WP**	*WP4-WP5*
Description/ Functionality	*This module improves the quality of the images using satellite images as source* *Part of BS4 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/terrain-segmentation-and-super-resolution (No code in the repo for internal policy reasons (model trained by us); only readme uploaded)*		
Integration component list	*This module is part of the tools created in WP4 for satellite using AI and is integrated with the fire danger prediction component.*		
Deployment location	*https://github.com/silvanus-prj/terrain-segmentation-and-super-resolution*		
Container size	*44.1 Gb virtual, 341kb*		
Requirements	*Python environment in a computer with GPU capability*		
Contact email	*jose.martinezs@eviden.com*		

Table 14: Description of the module for Detection of fire and fire related info from social media using CLIP

Title	*Detection of fire and fire related info from social media using CLIP*	**WP**	*WP4 (T4.4)*
Description/ Functionality	*This module detects fire and related information using text and images combined* *Part of BS4 in D8.1.*		
Repository URL	*https://github.com/silvanus-prj/social-media-data-extractor-from-Atos*		
Integration component list	*This module is part of the tools of T4.4 for social media detection, and thus integrated with UP3*		
Deployment location	*https://github.com/silvanus-prj/social-media-data-extractor-from-Atos*		
Container size	*No dockerization required (access using REST API)*		

Requirements	*Python environment in a computer with GPU capability*
Contact email	*jose.martinezs@eviden.com*

Table 15: Description of the module for Questions and answers from social media

Title	*Questions and answers from social media*	**WP**	*WP4 (T4.4)*
Description/ Functionality	*This module generates information about fire from social media using open questions (e.g.”is there fire in the image?”)* *Relevant to BS4 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/social-media-data-extractor-from-Atos*		
Integration component list	*This module is part of the tools of T4.4 for social media detection UP3*		
Deployment location	*https://github.com/silvanus-prj/social-media-data-extractor-from-Atos*		
Container size	*No dockerization required (access using REST API)*		
Requirements	*Python environment in a computer with GPU capability*		
Contact email	*jose.martinezs@eviden.com*		

Table 16: Description of the module for GeoLocation based on images in social media

Title	*GeoLocation based on images in social media*	**WP**	*WP4 (4.4)*
Description/ Functionality	*This module provides information about the place where a photo has been taken (as an estimation), thus helping finding the source of a possible fire* *Part of BS4 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/social-media-data-extractor-from-Atos*		
Integration component list	*This module is part of the tools of T4.4 for social media detection, UP3*		
Deployment location	*https://github.com/silvanus-prj/social-media-data-extractor-from-Atos*		
Container size	*No dockerization required (access using REST API)*		
Requirements	*Python environment in a computer with GPU capability*		
Contact email	*jose.martinezs@eviden,com*		

Table 17: Description of the Fire Spread Model

Title	*UP6 – Fire Spread Model*	**WP**	*WP5*
Description/ Functionality	Predicts the spread of the fire in several time intervals. Corresponds to BS5 of D8.3		
Repository URL	*https://github.com/silvanus-prj/fire-spread-model*		
Integration component list	*SAL, dashboards, Decision Support System, Health Impact Assessment*		
Deployment location	*Silvanus cloud*		
Container size	5GB (virtual 9GB)		
Requirements	RAM 32GB, CPU core i7 1165g7 or better		
Contact email	*a.bonanos@exus.ai, g.diles@exus.ai*		

Table 18: Description of Geo-location component

Title	*Geo-location*	**WP**	*WP2 and WP5*
Description/ Functionality	*Extraction and processing of geo-location of user-generated content. This component plays an important part in localisation of biodiversity data within the Woode application.* *Relevant to BS6 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/Geo-location*		
Integration component list	*This module is integrated in the pipeline of the Woode mobile application for extraction of geo-location data related to the biodiversity of forests.*		
Deployment location	*VTG server*		
Container size	*0.5GB*		
Requirements	*Java 8+* *Mapbox lib* *MySQL 8.0 database* *Android minSdk 28* *Android compileSdk 33* *Gson lib* *Retrofit lib*		
Contact email	*t.piatrik@venaka.eu, m.cavojsky@venaka.eu, r.pucek@venaka.eu*		

Table 19: Description of Image analytics component

Title	*Image analytics*	**WP**	*WP2*
Description/ Functionality	*This component is responsible for a range of image analytics processes, including image segmentation, augmentation and upsampling. These processes are part of the computer vision layer that is enabling the processing and analysis of the images of tree leaves gathered through the Woode mobile application.* *Relevant to BS6 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/Image-analytics*		
Integration component list	*This module is integrated in the pipeline of the Woode mobile application for analysis and processing of images.*		
Deployment location	*VTG server*		
Container size	*1GB*		
Requirements	*OpenCV* *TensorFlow* *MySQL 8.0 database* *Gson lib* *Retrofit lib* *Java 8+*		
Contact email	*t.piatrik@venaka.eu, m.cavojsky@venaka.eu, r.pucek@venaka.eu*		

Table 20: Description of Machine learning component for recognition of trees based on leaves' images

Title	*Machine learning*	**WP**	*WP2 and WP5*
Description/ Functionality	*This component is responsible for machine learning processes enabling classification of images and recognition of trees based on trained models. This includes deep learning models and convolutional neural networks that are specially tailored and optimised for targeted use case of the Woode application.* *Relevant to BS6 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/Machine-learning*		
Integration component list	*This module is integrated in the pipeline of the Woode mobile application for image classification and leaf/tree recognition tasks.*		
Deployment location	*VTG server*		
Container size	*1GB*		
Requirements	*TensorFlow* *TFLearn* *OpenCV*		

	MySQL 8.0 database *Gson lib* *Retrofit lib*
Contact email	*t.piatrik@venaka.eu, m.cavojsky@venaka.eu, r.pucek@venaka.eu*

Table 21: Description of Data annotation component

Title	*Data annotation*	**WP**	*WP5*
Description/ Functionality	*Large set of manual and machine-generated annotations of images of tree leaves. This component plays an important part in training of the machine learning algorithms and represents a valuable asset for any further scientific works on analysis of the biodiversity data* *This component is integrated in the pipeline of the Woode mobile application for training of machine learning modules and analysis of biodiversity data.* *Relevant to BS6 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/Data-annotation*		
Integration component list			
Deployment location	*VTG server*		
Container size	*Not yet determined*		
Requirements	*MySQL 8.0 database* *Gson lib* *Retrofit lib*		
Contact email	*t.piatrik@venaka.eu, m.cavojsky@venaka.eu, r.pucek@venaka.eu*		

Table 22: Description of Data aggregation component

Title	*Data aggregation*	**WP**	*WP5*
Description/ Functionality	*This module is responsible for data storage and knowledge management. It includes the database system designed to store the data extracted through the Woode mobile application. The component also includes all communication services between database and user-side application, and knowledge-based models for extraction of semantic data.* *Relevant to BS6 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/Data-aggregation*		
Integration component list	*This component is integrated in the pipeline of the Woode mobile application for storing, modelling, and knowledge management of the data.*		
Deployment location	*VTG server*		

Container size	*Not yet determined*
Requirements	*MySQL 8.0 database* *Gson lib* *Retrofit lib*
Contact email	*t.piatrik@venaka.eu, m.cavojsky@venaka.eu, r.pucek@venaka.eu*

Table 23: Description of Woode user-side mobile application component

Title	*Woode user-side mobile application*	**WP**	*WP2 and WP8*
Description/ Functionality	*This component represents the user-side of the Woode mobile application, including UI and all features necessary for gathering, visualising and communicating the data with the server side components.* *This component will be available through the app store and will be installed on the user mobile phone.* *Relevant to BS6 in D8.3.*		
Repository URL	*https://github.com/silvanus-prj/Woode-user-side-mobile-application*		
Integration component list	*NA*		
Deployment location	*VTG server and Google play store*		
Container size	*0.5GB*		
Requirements	*Java 8+* *Mapbox lib* *MySQL 8.0 database* *Android minSdk 28* *Android compileSdk 33* *Gson lib* *Retrofit lib*		
Contact email	*t.piatrik@venaka.eu, m.cavojsky@venaka.eu, r.pucek@venaka.eu*		

Table 24. Description of SILVANUS Semantic Knowledge Base

Title	*SILVANUS Semantic Knowledge Base*	**WP**	*WP5*
Description/ Functionality	*This component functions as an RDF triplestore, which stores both the T3.1 ontology as well as data from CTL's IoT fire/smoke detection device (T4.4), UTH's health monitoring device (T5.3) and CERTH's social media sensing (T4.3).* *BS7 in D8.3*		
Repository URL	*https://github.com/silvanus-prj/semantic-knowledge-base*		
Integration component list	*SAL*		

Deployment location	*Catalink's server*
Container size	*Not applicable*
Requirements	*RDF triplestore (e.g. Ontotext's GraphDB), python*
Contact email	*marios.iacovou@catalink.eu, skontogiannis@catalink.eu, maria.maslioukova@catalink.eu*

Table 25: Description of the Data Fusion Application

Title	*Data Fusion*	**WP**	*WP5*
Description/ Functionality	*Web services that provide the analysis of resource allocation in certain areas based on both area-wide and fire probability. We also provide a blueprint of the front-end concept as a reference for ITTI to build the front-end app* *Relevant to BS8 in D8.3*		
Repository URL	*(Webservices) https://github.com/silvanus-prj/fire-probability-analytics-back-end* *(Fe Blueprint - Private) https://gitlab.com/silvanus1/fire-probability-analitics/fe.git. Please contact us to become a collaborator*		
Integration component list	*Data ingestion, Fuzzy logic, Front-end map layer visualizer*		
Deployment location	*Amikom Local VM*		
Docker container size	*Webservices - 2 GB*		
Requirements	*Hardware: Minimum 4 VCPU, 8GB RAM, 25GB Storage* *Libraries: Python3, Fabric, numpy, Flask-SQLAlchemy, Flask-WTF, WTForms, coverage ,shortuuid, sqlalchemy-utils, geojson, pymysql, mysql-connector-python, pandas, geopandas, Flask_Cors, python-dotenv*		
Contact email	*kusrini@amikom.ac.id , arief_s@amikom.ac.id*		

Table 26 : Evacuation Route Planning

Title	*Evacuation Route Planning*	**WP**	*WP5 – T5.4.4*
Description/ Functionality	*The primary functionality of this component is to enhance the process of evacuation planning. Through integration of various SILVANUS components, such as forecasting fire spread and utilizing data from a range of internal and external sources, as well as applying appropriate models, this particular component has the capability to generate a set of routes that guarantee the stakeholders' safe migration along with the corresponding time frame within which they are considered valid.*		

	Relevant to BS9 in D8.3
Repository URL	*https://github.com/silvanus-prj/evacuation-paths*
Integration component list	*Storage Abstraction Layer, Knowledge Base, Decision Support System - Dashboard, Fire Spread Model, Health Impact Component*
Deployment location	*Local VM (silvanus.uth.gr).*
Container size	*In case of containerization, approximately 1.5 GB would be required.*
Requirements	*Python* *Python libraries (e.g., flask, requests, pymongo, openrouteservice, geojson, json)*
Contact email	*kostasks@uth.gr, paikonom@uth.gr, gboulougar@uth.gr*

Table 27: Description of the Health Impact Component

Title	*Health Impact Component*	**WP**	*WP5 - T5.3.3*
Description/ Functionality	*The main objective of this component is to monitor the levels of pollutants released by wildfires through the utilization of both portable and stationary IoT devices. It aims to assess the air quality in the impacted region and subsequently provide health related recommendations to stakeholders. In addition, it formulates a list of relative risk indicators associated with short-term and long-term exposure to emissions from wildfires.* *Relevant to BS10 in D8.3*		
Repository URL	*https://github.com/silvanus-prj/health-impact* *- http://silvanus.uth.gr/get-latest-data?emissions={INT}. Returns the latest n-th elements from the MongoDB in JSON format. Authentication is supported.* *- http://silvanus.uth.gr/aqi. Returns the most recently calculated AQI. Authentication is supported.* *- http://silvanus.uth.gr/data-metadata. Posts a http request (data) to the SAL. SILVANUS credentials are adopted.*		
Integration component list	*Storage Abstraction Layer, Knowledge Base, Decision Support System - Dashboard, Fire Spread Model*		
Deployment location	*Local VM (silvanus.uth.gr).* *MQTT broker (mqtt://iot.eclipse.org)*		
Container size	*In case of containerization, approximately 1.5 GB would be required.*		
Requirements	*Python* *Python libraries (e.g., flask, requests, pymongo, geojson, json, scipy)*		
Contact email	*kostasks@uth.gr, paikonom@uth.gr, gboulougar@uth.gr*		

Table 28: Description of Citizen Engagement App

Title	*Citizen Engagement App (CEA)*	**WP**	WP3
Description/ Functionality	*The Mobile Application for Citizen Engagement is implemented using React Native, Expo & Tailwind CSS. Contains several modules such as:* • *Educational Module containing Guidelines, News and Best Practices* • *Fire Reporting and Notification Module* *Relevant to BS11 in D8.3*		
Repository URL	*https://github.com/silvanus-prj/citizen-engagement-app*		
Integration component list	*SILVANUS Security Server* *Information sharing protocols between first responders and public (T8.2)* *Backend Services for the Citizen Engagement Mobile App's (CEA) / Content Management System (CMS)*		
Deployment location	*Google Play Store (test version):* *https://play.google.com/store/apps/details?id=cea.silvanus* *App Store:* *https://apps.apple.com/us/app/silvanus/id6483808614*		
Container size	*No container. .apk size: 44mb*		
Requirements	*The mobile app by itself is standalone and will be deployed in the Play Store & the App Store. Related backend services that are under development will be defined in later stages. The developed Backend Services include:* • *Fire Reporting and Notification Services (using the interfaced and customized EmerPoll service),* • *Content Management System*		
Contact email	*mariana@massivedynamic.se, emil.gatial@savba.sk*		

Table 29: Description of Backend Services for the Citizen Engagement Mobile App's (CEA)

Title	*Backend Services for the Citizen Engagement Mobile App's (CEA) / Fire Reporting Services*	**WP**	WP3 & WP8
Description/ Functionality	*One of the main modules of the Citizen Engagement Mobile App (CEA) is the "Fire Reporting and Notification" module. The backend of the module uses the EmerPoll cloud system (developed and customized by UISAV). The individual components of this Backend are the following:* • ***EmerPoll** – is a distributed cloud service for collecting and aggregating responses from mobile devices. It uses Polls/Channel/Template concepts to set up, execute and manage information collection and sharing campaigns. EmerPoll provides a UI as well as a REST API.*		

	• ***Information Sharing Protocol*** *– specification of message flows in Avro IDL schema format. The messages are compatible with the EmerPoll API. Specific configuration of Polls, Templates, Channels and Namespaces.* • ***MQTT Collector Node*** *– provides message persistence in the communication between CEA and EmerPoll. Uses MQTT with customized topics and reliable message delivery. It is also intended to manage binary data (images, videos) and mobile device location matching with Channels geo areas.* Relevant to BS11 in D8.3
Repository URL	The repositories for individual components: • *EmerPoll and MQTT Collector Node: Private GitLab repository* • *Information Sharing Protocol: https://github.com/silvanus-prj/protocols*
Integration component list	*Citizen Engagement App (CEA)* *Edge Micro Data Centre (EMDC)* *Mesh in the Sky* *SILVANUS Dashboard* *Storage Abstraction Layer (SAL)*
Deployment location	Deployment of Backend services are deployed on the UISAV's infrastructure: • *EmerPoll GUI: https://silvanus.emerpoll.eu/* • *EmerPoll REST API: https://silvanus.emerpoll.eu/rest/* • *Information Sharing Protocol: https://github.com/silvanus-prj/protocols* • *Collector Node: Erlang-based scalable service deployed in UISAV's Private Cloud.*
Container size	*Not using application containerization (Docker) but using system containers (LXC).*
Requirements	*The services to be deployed on a private cloud.*
Contact email	balogh@savba.sk, emil.gatial@savba.sk

Table 30: Description of the Backend Service of the Citizen Engagement Mobile App

Title	*Backend Services for the Citizen Engagement Mobile App*	**WP**	WP3
Description/ Functionality	*We have replaced the CMS solution with mobile native storage. The content is managed from the native app. This new solution was implemented since it helps to reduce the waiting time for the content to load, makes all the information available offline and it doesn't affect the download time or the performance of the app.*		
Repository URL	*https://github.com/silvanus-prj/citizen-engagement-app/ (same as app)*		

Integration component list	*Citizen Engagement Mobile App (CEA)* *SILVANUS Secure Server through the fire report module by EmerPoll*
Deployment location	*Within the native app.*
Container size	*NA*
Requirements	*The services are deployed in the react native app, no extra requirements.*
Contact email	mariana@massivedynamic.se, timo@massivedynamic.se

Table 31: Description of the Storage Abstraction Layer

Title	*Storage Abstraction Layer (SAL)*	**WP**	*WP5*
Description/ Functionality	*The SAL sits between the object store and the rest of the SILVANUS services. It hides the underlying store implementation from the services and provides additional functionality, such as the metadata index and emitting object events.* Relevant to BS12-21 in D8.3		
Repository URL	*https://github.com/silvanus-prj/sal*		
Integration component list	*1) Data ingestion services for obtaining data products from third-party systems.* *2) Data ingestion service for receiving data from UAVs, UGVs and IoT Gateways in the field.* *3) Knowledge Based System* *4) User products*		
Deployment location	*Silvanus Cloud*		
Container size	*1) Data & metadata ingestion microservice: 3.83GB* *2) Metadata index microservice: 258MB* *3) Schema microservice: 100MB* *4) Message queue microservice: 269MB* *5) Data retrieval microservice: 152MB* *6) Claim Check Pattern retrival microservice: 152*		
Requirements	*CPU: 8-12 vCPU* *RAM: 16GB* *STORAGE:* *1) Object storage MinIO +500GB.* *2)* Persistent Volume +50GB		
Contact email	*mustafa.albado@dell.com*		

Table 32: Description of the Data Ingestion Pipeline

Title	*Data Ingestion Pipeline*	**WP**	*WP4*

Description/ Functionality	*Data collection, aggregation and pre-processing engine from third-party and internal data sources.* *Implements BSs – 13, 14, 16, 17, 18 in D8.3*
Repository URL	*https://github.com/silvanus-prj/dip*
Integration component list	*1) SAL* *2) Internal Data Providers (UAV,UGV,IoT)* *3) UPs/Dynamic Data Consumers*
Deployment location	*SILVANUS Cloud / SILVANUS FCC*
Docker container size	*1) Pipeline Engine: 2GB* *2) Pipeline Initiator Microservice: 150MB* *3) RabbitMQ + UI Service (shared): 250MB*
Requirements	• *CPU: 4 Core+* • *RAM: 32GB+* • *STORAGE: 512GB+*
Contact email	*matthew_keating@dell.com*

Table 33: Description of OpenStreetMap Conversion module

Title	*OpenStreetMap Features Conversion*	**WP**	*WP4*
Description/ Functionality	*The program extracts roads and railways features from Open Street Map (OSM) shapefile and converts to NetCDF format.* *Relevant to BS13 in D8.3*		
Repository URL	*https://github.com/silvanus-prj/osm_to_netcdf*		
Integration component list	*The program is integrated in the Ingestion Data flow from source to SILVANUS Storage Abstraction Layer (Post-processing).*		
Deployment location	*Silvanus cloud*		
Container size	*No container*		

Requirements	*Python3 main libraries used:* • *Shapely* • *Numpy* • *Geopandas* • *dask_geopandas* • *netCDF4*
Contact email	*ivo.gama@terraprima.pt, jorge.palma@terraprima.pt*

Table 34: Description of Sentinel Derived Indices

Title	*Sentinel Derived Indices*	**WP**	*WP4*
Description/ Functionality	*The program downloads Sentinel2 images and create vegetation indexes to netcdf and/or gtiff format* *Relevant to BS13 in D8.3*		
Repository URL	*https://github.com/silvanus-prj/sentinel2_to_ndvi*		
Integration component list	*The program is integrated in the Ingestion Data flow from source to SILVANUS Storage Abstraction Layer (Post-processing).*		
Deployment location	*Silvanus cloud*		
Container size	*No container*		
Requirements	*Python3 main libraries used:* • *cdsetool* • *Numpy* • *Rasterio* • *Shapely* • *netCDF4*		
Contact email	*ivo.gama@terraprima.pt, jorge.palma@terraprima.pt*		

Table 35: Description of SILVANUS MetaData Extractor

Title	*SILVANUS Metadata Extractor*	**WP**	*WP4*

Description/ Functionality	*The developed system aims to extract metadata from the object data injected into the Silvanus platform and stores them, by json files, into the disposed repository.* *Relevant to BS15 in D8.3*
Repository URL	*https://github.com/silvanus-prj/metadata-extractor*
Integration component list	*The Silvanus Metadata Extractor is integrated with the Apache Nifi processor (ExecuteStreamCommand).*
Deployment location	*Silvanus Cloud*
Container size	*No container.*
Requirements	*CPU 1.80 GHz, RAM 16 GB, STORAGE (depends on the size and quantity of the processed data).*
Contact email	*mcefarelli@expert.ai, ccaterino@expert.ai*

Table 36: Description of SILVANUS Security Server

Title	*Silvanus Security Server*	***WP***	*WP5*
Description/ Functionality	*Silvanus Security Server container consists of a Keycloak authorization server and PostgreSQL database management system. Moreover, the Keycloak server is configured with the custom configuration allowing authentication and authorization based on Silvanus user roles as well as used pilot sites. The provided code contains a simple proof-of-concept Python web app that could be used during connectivity tests.* *Relevant to BS22 in D8.3*		
Repository URL	*https://github.com/silvanus-prj/silvanus-security-server*		
Integration component list	*This component is integrated with SAL (DELL) and the Dashboard (ITTI).*		
Deployment location	*Silvanus Cloud*		
Container size	*We do not use raw container - see more details concerning Kubernetes pod requirements below.*		
Requirements	*At least 1 virtual processor, at least 4GB RAM, 10GB of storage*		
Contact email	*krzysztof.cabaj@pw.edu.pl*		

Table 37: Description of UI framework

Title	*UI framework (common dashboard)*	**WP**	*WP5*

Description/ Functionality	*A web-based interface which will felicitate the crisis management during fires. Display of an interactive map for monitored area, with layers corresponding to different sources of data about fire probability and fire events.*
Repository URL	*https://github.com/silvanus-prj/UI-framework*
Integration component list	*SAL, RMQ,* *Fire Danger Index* *Fire Spread Forecast* *Notifications From IoT Devices* *notifications from Citizen Engagement App* *notifications from Social Media* *Fire Detection At The Edge* *UGV* *Evacuation Route Planning* *Health Impact Assessment* *Firefight Resource Allocation* *Multilingual Forest Fire Alert* *Priority Resource Allocation*
Deployment location	*Silvanus Cloud*
Container size	
Requirements	
Contact email	*mprzybysz@itti.com.pl*

Table 38: Description of robot navigation and mapping module

Title	*Robot navigation and mapping module*	**WP**	*WP4*
Description/ Functionality	*This is an on-robot software system for the proprietary sensor payload (lidar, IMU, cameras, GPS), which allows the robot to autonomously/semi-autonomously explore and navigate within wildfire environments, while mapping the environment in three dimensions as point clouds and associated images.* *The system includes a base station software component that allows the robot to be controlled and the sensor readings to be processed by a user in a safe location. The base station software also sends a number of pieces of information up to the Silvanus platform over REST, namely images, locations and orientations of the robot.*		
Repository URL	*https://github.com/silvanus-prj/ground-robotics-CSIRO*		
Integration component list	*This module integrates with the tools for T4.3, for navigation to/from wildfire fronts.*		
Deployment location	*None*		

Container size	*Not applicable*
Requirements	*This is a proprietary, embedded module that is specific to the sensor payload, and cannot be installed on different CPUs. It therefore requires the specific NUC processor, with a proprietary integration of Velodyne VLP16 lidar, IMU, cameras and GPS units in order to function. We do not support its transfer to a different sensor payload.*
Contact email	*thomas.lowe@csiro.au*

Table 39: Mesh in the Sky

Title	*Mesh in the Sky*	**WP**	*WP 5*
Description/ Functionality	*A self-configurable rapidly deployable mesh network utilising both ground and UAV-based nodes.*		
Repository URL	Not Applicable		
Integration component list	*IoT Sensors for detection and monitoring of forest fires.*		
Deployment location	*Self-contained and could be deployed at any pilot site.*		
Container size	*Not applicable*		
Requirements	*Requires UAV pilot license for operation of drones*		
Contact email	*garik@rinicom.com; lee.sessions@rinicom.com; projects@rinicom.com;*		

Table 40: Description of the Social Media Application

Title	*Social Media Sensing*	**WP**	*WP5*
Description/ Functionality	*API Classification: Web API that provides fire prediction based on text input in Indonesian. API NER: Web API that detects the location in a tweet. API Fire Tweet: Web API that provides a time-ranged count of tweets categorized in the label that correlated with fire forest.* *Relevant to BS8*		
Repository URL	*API Classification: https://github.com/silvanus-prj/social-media-sensing-api-ner* *API NER: https://github.com/silvanus-prj/social-media-sensing-api-ner* *API Fire Tweet: https://github.com/silvanus-prj/social-media-sensing-back-end*		
Integration component list	*With DSS in version 2 of the platform*		
Deployment location	*Amikom Local VM* *API Classification:*		

	API NER: *API Fire Tweet:*
Docker container size	*API Classification:* *API NER:* *API Fire Tweet:*
Requirements	*Hardware: Minimum 4 VCPU, 8GB RAM, 25GB Storage* *Classification & NER Libraries:* *scikit-learn, pandas, matplotlib, seqeval, Flask, PySastrawi, deep_translator, shortuuid, tensorflow==1.15, h5py==2.10, keras==2.3.1, keras-applications==1.0.8 , keras-preprocessing==1.0.5, protobuf==3.19, keras-team: git+https://www.github.com/keras-team/keras-contrib.git* *API Fire Tweet Library: Node JS 14*
Contact email	*kusrini@amikom.ac.id, arief_s@amikom.ac.id*

Table 41: Description of Kubeflow Pipeline Component Factory

Title	*KFP Component Factory*	**WP**	*WP5*
Description/ Functionality	*The developed system aims to create single Kubeflow Pipeline stages from single PY functions, in form of YAML component descriptor file.* *The components/files can be loaded and composed for creating KF pipelines for running machine learning experiments/trainings/evaluations.*		
Repository URL	*https://github.com/silvanus-prj/kfp-component-factory*		
Integration component list	*Stand-alone: the component can be exploited by each ML model developer.*		
Deployment location	*Silvanus Cloud*		
Container size	*No container.*		
Requirements	*Kubeflow engine on a reachable node.*		
Contact email	*ccaterino@expert.ai*		

2 Integration environment

In this section we give an overview of the technologies around which the SILVANUS platform has been developed and the main features of the infrastructure on which currently the SILVANUS backend services have been deployed. Specifically, we present details about the continuous integration/deployment environment and its configuration, the core of the SILVANUS middleware built around the Storage Abstraction Layer (SAL) that provides a shared service for data ingestion, storage and retrieval and the infrastructure that has been developed to host these services.

2.1 Software repository and development flow

In D8.1 [1] GitLab was presented as a platform to support the SILVANUS DevOps repository and related toolchain. At the beginning of the 2nd project year Gitlab announced a limitation of the number of users for private projects, which was considered a severe limitation for SILVANUS. Thus, the decision to move to the GitHub platform was taken, since GitHub was evaluated as the most appropriate solution. Its features and the SILVANUS project environment that has been setup to host the SILVANUS cloud platform are presented below in this deliverable. GitHub is a state-of-the-art framework used in many large-scale projects to manage source code and for version control [2]. GitHub is an open-source code management (SCM) system based on Git [3] but adding its own features covering for instance the DevOps pipeline. Hence, it keeps many of the GitLab features already considered since D8.1 and we briefly present the adapted development and integration flow below.

GitHub offers a rich set of solutions and features such as a git repository, issue tracking, projects and, most important, a set of CI/CD tools with GitHub Actions.

Development **Op**erations (DevOps) unifies the software development and operations with that software in an automated way. Focus on software testing, quality control, best practices, integrations tests and, at last, deployment on production environments. This automated workflow makes delivery faster with better quality and tested releases.

To achieve a more robust workflow, the following stages emerge:

1. Branches definition: main, development, staging
2. Tags definition for versions and releases
3. Definition of a build script to build the app
4. Definition of testing script of the build
5. QA evaluation of the build
6. Deploy to staging
7. Deploy to production

Continuous integration is a practice that prevents integration problem also known as integration hell since each pull request on targeted branch is tested and, if passed, merged. In GitHub the setup of continuous integration and deployment is achieved by workflows. The above is depicted in Figure 1.

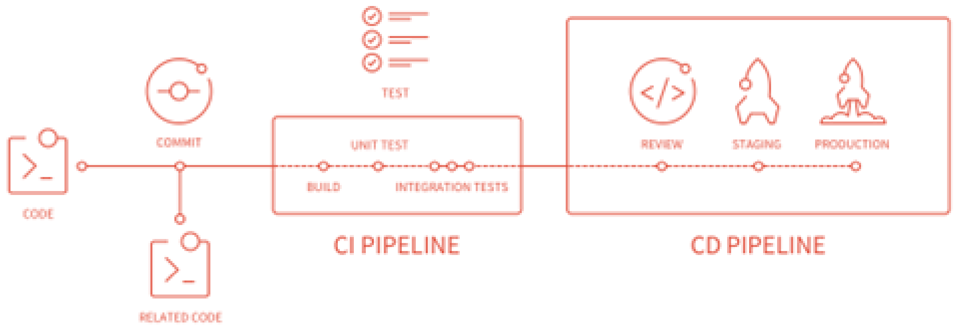


Figure 1: GitHub continuous integration schema

2.2 GitHub based CI/CD

2.2.1 Docker

Containers operate like Virtual Machines (VMs) providing abstraction and isolation of resources holding an application's dependencies all in one place; hence, making it portable and easy to transfer from one environment to another. However, containers unlike VMs are lightweight only virtualizing the operating system; thus, a standalone container image is enough to run an application on a system without having to install any additional packages. They can be easily transferred from one environment to another and work uniformly throughout. Docker is a technology that provides an abstraction layer over container management technology. Instead of virtualizing an operating system for each service deployment of an application, the deployment on docker is rather simple: the guest operating system provides all the network, storage and resource management, while inside a docker container fits a deployed application and its dependencies libs (nodejs, java, mysql, etc), providing encapsulation enhancing overall security. The abstraction technologies and their differences are shown in Figure 2

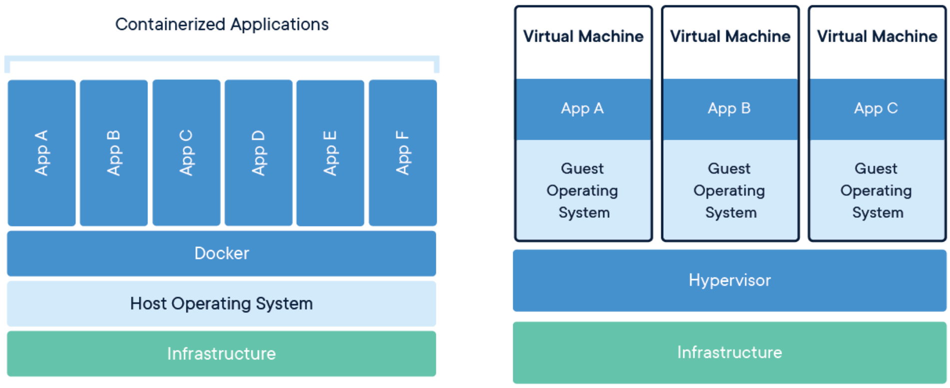


Figure 2: Abstraction technologies - Containers vs. VMs [13]

2.2.1.1 Dockerfile

The creation of a docker image is achieved with a Dockerfile script containing the setup of the app dependencies, app installation, database configuration and all the required steps.

2.2.1.2 Docker Registry

Like git repository, an image build can be committed to a repository called registry that holds the image changes.

The registry used for Silvanus project is an account in Docker Hub with Pro Plan in order to have unlimited private images.

- `Username: silvanusproject`
- `Token: *******`

2.2.2 GitHub Actions

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows to automate a build, test, and deployment pipeline. GitHub Actions goes beyond just DevOps and run workflows when other events happen in the repository.

2.2.2.1 Workflows – Events

Workflow is basically an automated procedure that's made up of one or more jobs. It can be triggered by 3 different ways:

1) By an event that happens on the Github repository
2) By setting a repetitive schedule
3) Or manually clicking on the run workflow button on the repository UI.

To create a workflow, we need to add a *.yml* file in the github*/workflows* directory of the repository (e.g. docker-ci.yml) containing the workflow jobs, the separate steps, the functions, and variables.

We can define how a workflow will be triggered using the **on** keyword.

```
on:
  push:
    branches: [ main ]
    tags: [ 'v*.*.*' ]
  schedule:
  - cron:  '*/15 * * * *'
  release:
    types: [published]
```

For a complete list of events that can be used to trigger workflows, see Events that trigger workflows[1].

2.2.2.2 Runners

A runner is a server that runs the workflows when they're triggered, so there's a need to attach a runner to run the job. Self-hosted GitHub runners have been deployed and added into the GitHub organization. They are shared, so they can be used with GitHub Actions from all organization repositories.

We use the **run-on** keyword with *self-hosted* label to specify a self-hosted runner we want to use. The job will be attached and run to an available self-hosted runner.

```
jobs:
  docker:
    runs-on: [self-hosted]
```

[1] https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows

2.2.2.3 Jobs – Steps – Actions

A workflow consists of one or multiple **jobs** as shown in Figure 3. All jobs inside a workflow normally run in parallel, unless they depend on each other, then in that case, they run serially. Each job will be run separately by a specific runner and is composed of multiple **steps**. Steps are individual tasks that run serially, one after another. Each step can have one or more **actions** (basically a standalone command). The good thing about action is that it can be reused. If someone has already written a GitHub action that we need, we can actually use it in our workflow.

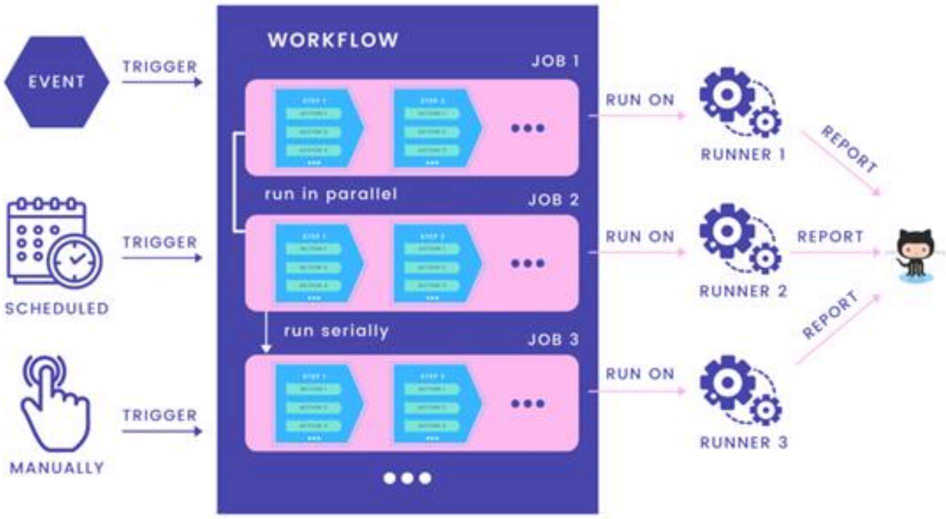


Figure 3: GitHub Actions [11]

2.2.2.4 Activities – Logs

After a workflow run has started (as shown in Figure 4), we can see a visualization graph of the run's progress and view each step's activity, logs, results on GitHub UI (shown in Figure 5).

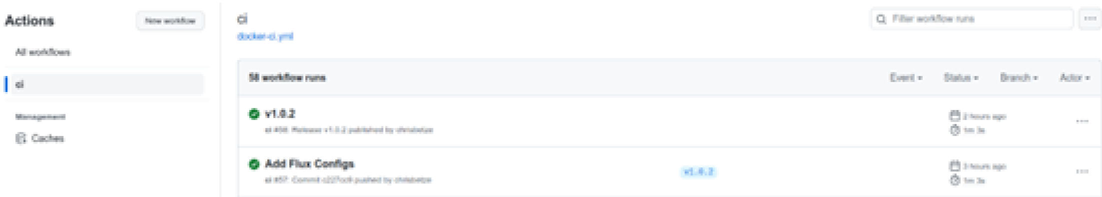


Figure 4 Workflow Runs

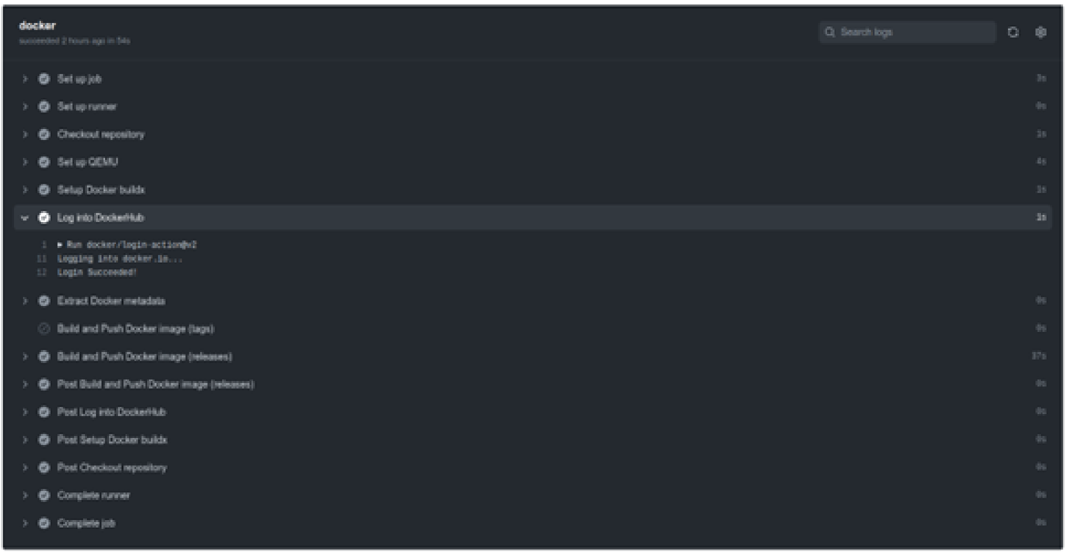


Figure 5: Job Run Details

For more information see [13].

2.2.3 Flux CD

Flux is a set of continuous and progressive delivery solutions for Kubernetes that are open and extensible. We use Flux to deploy our applications in a **GitOps** manner. The basic core concepts in Flux are shown in Figure 6.

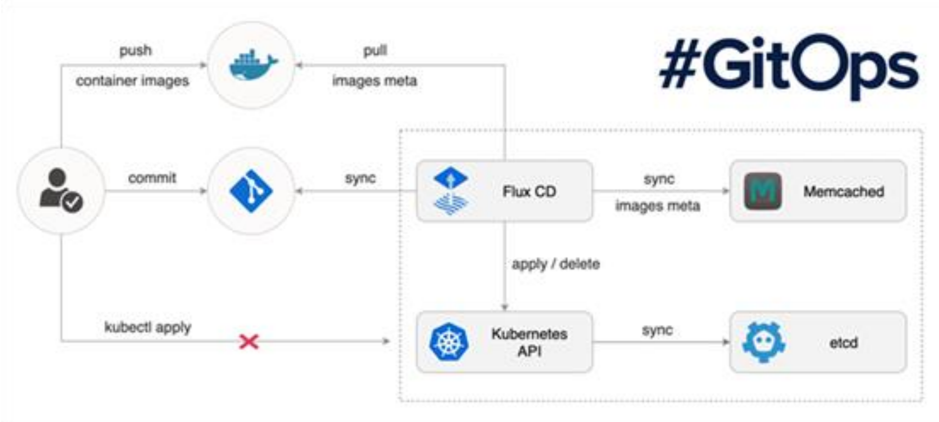


Figure 6: Flux Basic Flow [14]

2.2.3.1 GitOps

GitOps is a way of implementing Continuous Deployment for cloud native applications. It focuses on a developer-centric experience when operating infrastructure, by using tools developers are already familiar with, including Git and Continuous Deployment tools.

The core idea of GitOps is having a Git repository that always contains declarative descriptions of the infrastructure currently desired in the production environment and an automated process to make the production environment match the described state in the repository.

2.2.3.2 Sources

A Source defines the origin of a repository containing the desired state of the system and the requirements to obtain it. For example, the latest 1.x tag available from a Git repository over SSH.

The origin of the source is checked for changes on a defined interval, if there is a newer version available that matches the criteria, a new artifact is produced.

All sources are specified as *Custom Resources* in a Kubernetes cluster, examples of sources are **GitRepository**, **ImageRepository**, **HelmRepository** and **Bucket** resources.

2.2.3.3 Reconciliation

Reconciliation refers to ensuring that a given state (e.g. application running in the cluster, infrastructure) matches a desired state declaratively defined somewhere (e.g. a Git repository).

- **HelmRelease** reconciliation ensures the state of the Helm release matches what is defined in the resource, performs a release if this is not the case (including revision changes of a HelmChart resource).
- **Kustomization** reconciliation ensures the state of the application deployed on a cluster matches the resources defined in a Git or OCI repository or S3 bucket.
- **Bucket** reconciliation downloads and archives the contents of the declared bucket on a given interval and stores this as an artifact, records the observed revision of the artifact and the artifact itself in the status of resource.

2.2.3.4 Kustomization

The Kustomization custom resource represents a local set of Kubernetes resources (e.g. kustomize overlay) that Flux is supposed to reconcile in the cluster. The reconciliation runs every five minutes by default, but this can be changed with **.spec.interval**. If you make any changes to the cluster using kubectl edit/patch/delete, they will be promptly reverted. You either suspend the reconciliation or push your changes to a Git repository.

For more information about the basics of Flux CD see Core Concepts[2].

2.3 SILVANUS Data Ingestion, Storage and Retrieval

The Data Ingestion Pipeline (DIP) is a common ingestion framework for the collection, pre-processing and annotation of Data Objects. Data Object providers vary in a wide range of domains but can be summarized in two categories outlined below. The DIP does not provide the persistent storage of incoming Data Objects, rather this component is tightly coupled with the Storage Abstraction Layer (SAL) providing the input processed Data Objects (Objects + Metadata) as output.

In this section we provide an outline of both DIP, SAL, and other SILVANUS services (e.g., KB) components as well as examples of operation and references covering:

1. DIP – Data Ingestion mechanism for Internal Data Providers
2. DIP – Dynamic Data Request mechanism for User Products

[2] https://fluxcd.io/flux/concepts/

3. SAL – Data Retrieval mechanism for User Products

2.3.1 Component Details

2.3.1.1 Data Ingestion Pipeline

Internal Data Providers include:

- Drone / UAV (Image, Video)
- Ground Robot / UGV (Image, Video)
- IoT Devices (Temperature, Humidity, Gas/Smoke particles in air, Image)

External (3rd Party) Data Providers include:

- **Static Product Ingestion:** This category defines datasets which have statically defined parameters for ingestion. Parameters for ingestion refer to attributes such as the geospatial area, ingestion frequency, data format or temporal range. For some data sources we need to only consider a statically defined set of parameters (and therefore ingestion messages) which can be automated and ingested based on these requirements.
- **Custom Product ingestion:** Custom dataset parameters are a requirement of some datasets, specifically the user products that leverage these datasets within AI/ML training / inference and visualization dashboards. Storage Abstraction Layer

The Storage Abstraction Layer (SAL) serves as an intermediary between data sources, user products, and the object store within the SILVANUS system. Its primary function is to abstract the object store, offering two key advantages. Firstly, it enables flexibility in managing data at rest, allowing for efficient data management practices. Secondly, it decouples data from user products in a multi-source, multi-client environment, providing support for security, policy, privacy, and business constraints. By utilizing the SAL, the SILVANUS system achieves enhanced control and adaptability in handling data across various components.

2.3.1.1.1 Object store

The object store serves as the central repository within SILVANUS for storing both raw and processed data. Given that a significant portion of the ingested raw data in SILVANUS is unstructured, it is more efficient to store it in a unified object store rather than employing multiple databases for implementing the object store in SILVANUS, the MinIO object store is utilized and managed through the standard S3 storage API. This combination ensures seamless compatibility and efficient data management within the SILVANUS ecosystem. In the SILVANUS platform, the Apache NiFi PutS3Object processor is used to store the data in the MinIO object store.

MinIO is an open-source object storage system that is designed to be simple, scalable, and cloud-native. It allows you to store and retrieve large amounts of unstructured data, such as documents, images, videos, and other types of files. MinIO is built on the concept of object storage, where data is stored as objects rather than in a hierarchical file structure. Each object is assigned a unique identifier and is stored with its associated metadata. This approach allows for efficient and flexible storage of data, as objects can be accessed and manipulated independently. One of the key features of MinIO is its high scalability. It is designed to scale horizontally by distributing data across multiple servers, allowing you to expand storage capacity as your data grows.

2.3.1.1.2 Data and metadata ingestion

The interactions among the SAL for the three ingestion methods - external, internal, and user products - are depicted in Figure 7. A distinct SAL interface is required for each ingestion method to facilitate communication. During this process, the data object and its metadata are coupled and sent to the SAL by the Data Ingestion Pipeline (DIP) over endpoint. The SAL implementation then processes the input based on its origin, validates the metadata, and confirms that there are no data duplicates. The data objects are stored and/or forwarded to the user products via the message bus, and a metadata entry is added to the metadata index.

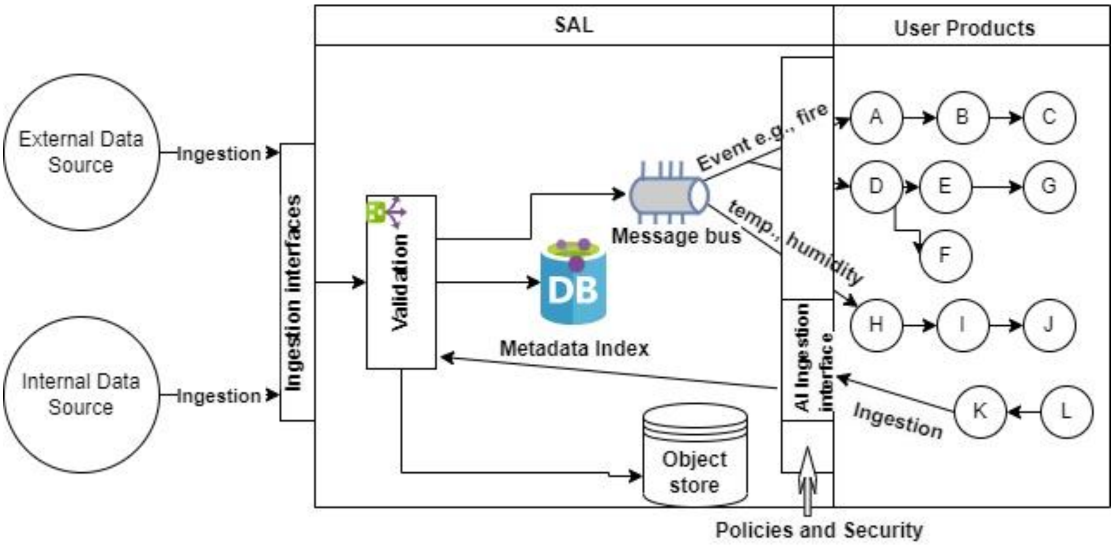


Figure 7: The interaction between SAL and other components in SILVANUS platform

Figure 8 illustrates the implementation of the validation steps using Apache NiFi. The validation process focuses on the mandatory fields specified in Table 7 of Deliverable 8.1 [1].

Figure 8: Metadata validation

The SILVANUS SAL metadata index relies on the Knowledge Graph technology, which is recommended to be implemented without blank nodes and duplicates to optimize search efficiency. To achieve this, three duplication check steps are employed. The first step involves a data duplication check, where the unique ID of the object data provided by the data source is utilized. The second step utilizes the metadata Format field, which comprises subfields such as type, resolution, and event. Lastly, the Spatial field describes the spatial characteristics of the data object, including coordination and pilot. These duplication check steps are executed using NiFi processors and JenaDB. Figure 9 shows the workflow of data and metadata duplication checks. Figure 10 showcases the implementation of the Data and Metadata duplication check process using Apache NiFi.

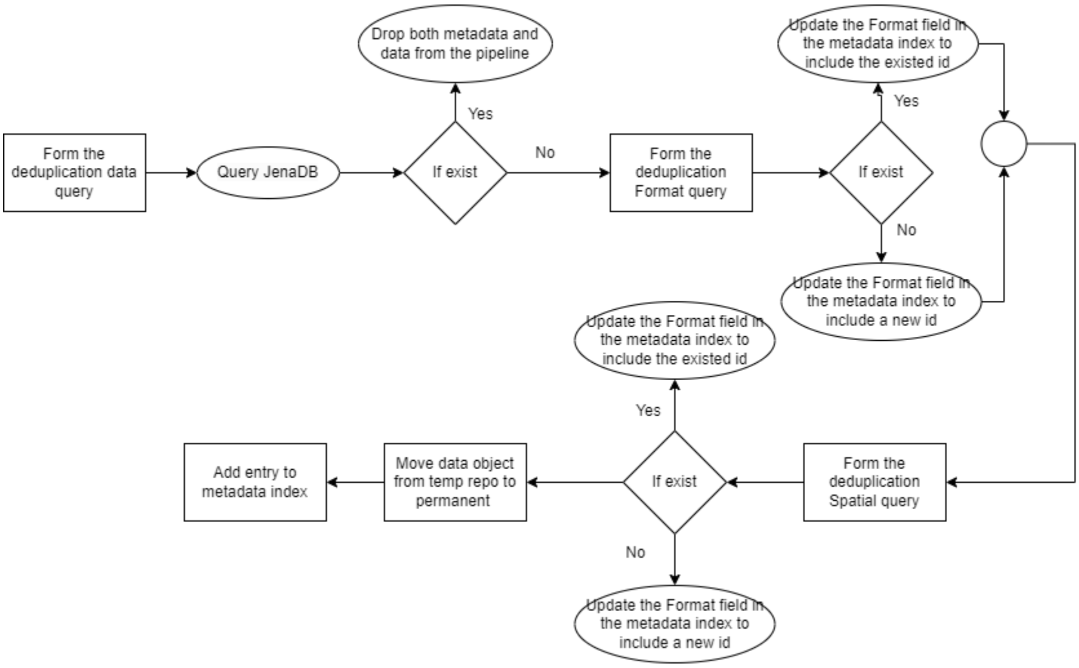


Figure 9: The flowchart of data and metadata deduplication

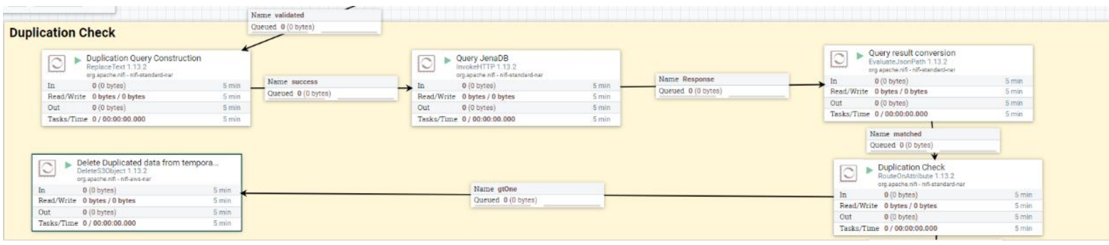


Figure 10: Data and Metadata duplication check

Data objects in SILVANUS can sometimes be quite large, up to 5GB. To optimize performance and minimize memory usage, these objects are primarily stored on disk in the temporary directory and are only loaded into memory when needed. Conversely, since metadata messages are lightweight, they are kept in memory. Data objects are only stored when they pass metadata and data duplication checks. Once stored, the data objects are saved in the Object storage using the Apache NiFi PutS3Object processor. If user products do not directly request the data objects, they are deleted from the temporary directory. However, if the data objects are passed to the message bus, they remain in the temporary directory until their expiration date.

Once the data has been stored in the object storage, the results of the "Data and metadata duplication check" process are transformed into Triples format before being included as an entry in the knowledge-graph-based metadata index.

2.3.2 User Product Endpoints

The currently defined UP endpoints are shown in Table 42 below.

Table 42: User Products Endpoints

Service	IPs/Endpoint	Format	Method
Data Ingestion Pipeline (DIP)	10.20.20.3		
UAV_Ingest	DIP:31903/	Multi-part form (data+metadata)	POST
UGV_Ingest	DIP:31904/	Multi-part form (data+metadata)	POST
IoT_Ingest	DIP:31905/	Multi-part form (data+metadata)	POST
AQI_Ingest	DIP:31906/	Multi-part form (data+metadata)	POST
Storage Abstraction Layer (SAL)	10.20.20.3: 30516		
Metadata Query API	SAL:30130/metadata/query	{key: value, key: value}	POST
Data Retrieval API	SAL:31222/api/getfiles	{'id': '*data-uuid*'}	POST
DIP Message Bus	TBD		

2.3.3 Demonstrations

2.3.3.1 DIP – Data Ingestion mechanism for Internal Data Providers

Figure 12 presents a sample Data Object and Metadata Descriptor generated for a single data point captured during a test flight (Figure 11).

Figure 11: *Drone image capture during mission*

```
{
  "descriptor": {
       "uuid": "01879876-66bb-7cd4-9bac-3b3b40051722",
       "obj-class": "UAV",
       "format": {
            "type": "jpg"
       },
       "access": "slovak-pilot",
       "dataset-type": "image",
       "created": "1681890268.745728"
  },
  "lineage": {
       "source": [],
       "processing": "primitive"
  },
  "spatial": {
       "type": "Point",
       "coordinates": [
            {
                 "lon": "48.115451",
                 "lat": "17.138385"
            }
       ],
       "wkt": "POINT (48.115451 17.138385)",
       "pilot": "slovak",
       "properties": {}
  },
  "temporal": {
       "datetime": "1681890268.745728"
  },
  "tag": {
    "LeftTop": {
      "Latitude": 48.115451591784364,
      "Longitude": 17.138036680355089
    },
    "RightTop": {
      "Latitude": 48.115451591784364,
      "Longitude": 17.138734865055067
    },
    "LeftBottom": {
      "Latitude": 48.115062934582824,
      "Longitude": 17.13803668299569
    },
    "RightBottom": {
      "Latitude": 48.115062934582824,
      "Longitude": 17.138734862414466
    },
    "Center": {
      "Latitude": 48.115451591784364,
      "Longitude": 17.138385772705078
    },
    "Azimuth": 0.0,
    "Altitude": 50.0,
    "FocalLength": 0.0,
    "FieldOfView": 83.0,
    "Angle": 90.0,
    "UploadType": 5
  }
}
```


Figure 12: *Metadata Descriptor for drone capture – (SILVANUS Metadata JSON-format-v2.2)*

The Data Object is a simple .jpg image from one point of the mission, while the Metadata Descriptor describes the range of attributes about this data point. Description of each metadata field can be found in [4].

Critically:

1. The Metadata Descriptor 'uuid' field matches the uuid of the filename / data object
2. The Metadata Descriptor we consider the 'tags' field, encoding additional indexing data generated during the mission – this data can be later used in an SQL-like (SPARQL) interface as well as an abstraction API allowing for JSON based query match indexing

```
curl --request POST \
  --url http://10.20.20.3:31903/ \
  --header 'Content-Type: multipart/form-data' \
  --header 'User-Agent: Insomnia/2023.5.7' \
  --form 'data=@C\...\UAV_Ingestion\01879876-66bb-7cd4-9bac-3b3b40051722.jpg' \
  --form 'metadata=@C\...\UAV_Ingestion\01879876-66bb-7cd4-9bac-3b3b40051722.json'
```

Figure 13: *Ingestion Request containing Data Object & Metadata Descriptor from Data Provider*

In *Figure 13*, we ingest the desired data point (image + descriptor) to the relevant Data Ingestion Pipeline specifically, the UAV ingestion endpoint, with no faults we should see a request status 200. This process remains consistent across any type of data that may be ingested into the system, assuming the Metadata Descriptor and access is correctly generated, a wide range of data objects can be ingested into the SILVANUS Storage Layer.

2.3.3.2 DIP – Dynamic Data Request mechanism for User Products

Requests for custom datasets are consumed by the SILVANUS Message Bus and initiate the Data Ingestion Pipeline of the relevant data provider with user provided parameters. Figure 14 shows a sample of a message sent to the Message Bus using a RabbitMQ dashboard UI and a simple JSON message payload.

Other methods of interaction with the Message Bus include language specific APIs / packages and HTTP based interface implemented by the RMQ server. In the current implementation there is no direct request response or user notification of an ingestion error, other than messages that support the claim-check pattern. This is a priority feature to be implemented over the coming interactions of the Data Ingestion Pipeline and Message Bus. In Figure 14 an example of how to ingest data into a specific RabbitMQ queue is shown while the full set of the RabbitMQ queues are listed in Table 43.

-

Table 43: The RabbitMQ queues

Queue Name	Dataset	Parameters	Output
`ingest.dem`	Digital Elevation Model	`pilot: [*pilot_string]` `type: [dem, asp, slp]`	Tiff
`Ingest.cdem`	Digital Elevation Model	`bbox: [*GeoJSON_coords]` `type: [dem, asp, slp]`	Tiff

`ingest.osm`	OpenStreetMaps Road / Rail	`Pilot: [*pilot_string]`
		`type: [road, rail]`
		`resolution: [*Int]`
		`bbox: [*GeoJSON_bbox]`
			GeoJSON, NetCDF

`ingest.sentinel-ndvi`	Sentinel-2/3 + NDVI	`resolution: [*Int]`
		`footprint: [*GeoJSON_bbox]`
		`cloud: [*Int]`
			SAFE, Tiff

`ingest.lst`	Land Surface Temperature	`type: - [H, DC, TCI]`	NetCDF
`ingest.ba`	Burned Area	`version: - [V1, V3]`	NetCDF
`ingest.pop`	Population Density	`pop_year: - [*YYYY]` `country: - [*ISO 3166 code]`	CSV, Tiff

`ingest.stf`	Short-term Forecast	`prod_date: - [*YYYY_MM_DD]`
		`b_north: - [*float]`
		`b_south: - [*float]`
		`b_west: - [*float]`
		`b_east: -[*float]`
			NetCDF

`ingest.clc`	Corine Landcover	`year: - [*YYYY]`
		`b_north: - [*float]`
		`b_south: - [*float]`
		`b_west: - [*float]`
		`b_east: -[*float]`
			Tiff

```
import json
import requests

#Demo script that shows how data can be ingested into rabbitmq.
headers = {}

# Read metadata.json and use its contents as the header
# IMPORTANT uuid needs to be changed everytime data is ingested as system does not allow for duplicate uuids
#metadata.json is your metadata put into a json file
with open('metadata.json', 'r') as f:
    headers["metadata"] = json.load(f)

headers['metadata'] = json.dumps(headers['metadata'])

# Add the additional header
headers['amqp$vhost'] = 'DemovHost'

#headers['amqp$exchange'] = 'fanout'
#Demo.ex.fanout allows user to send 1 input to multiple queues in this case the input is sent to Demo.q.fanout1 and Demo.q.fanout2
headers['amqp$exchange'] = 'Demo.ex.direct'
headers['amqp$routingKey'] = 'direct' #should be direct if using Demo.ex.direct and fanout if using Demo.ex.fanout

# Please enter here the name of file you want to send
with open('#here#', 'r') as f:
    data = f.read()

# URL to post data
url = 'http://10.20.20.3:30516/metadata/ingest'

# Post the data with the metadata as the header
response = requests.post(url, headers=headers, data=data)

# Print the response
print(response.text)

# Optionally, handle the response. For example, check if the request was successful:
if response.status_code == 200:
    print("Data successfully posted!")
else:
    print(f"Failed to post data. Status code: {response.status_code}. Response text: {response.text}")
```

Figure 14: Data ingestion to SAL and RabbitMQ

2.3.3.3 SAL – Claim-check - Data Retrieval User Products

Due to the substantial size of some of the datasets handled by the SILVANUS system, which can reach up to approximately 5GB, it is not advisable to directly pass such large files as part of the event messages between services. To address this, the Claim Check Pattern (CCP) design is employed. The SILVANUS SAL makes decisions based on this pattern's ccp_threshold and the pub/sub queue policy. It saves the data objects in a temporary data repository while updating the metadata with relevant details for retrieving the stored data objects using the "Add retrieval Info to Metadata" processor depicted in Figure 15. As shown in the figure, the updated field format enables consumers to retrieve the data utilizing the CCP solution efficiently. In Figure 16, an example of code to consume CCP automatically is shown.

```python
import pika
import sys
import requests
import json

url = 'http://10.20.20.3:30666/api/getfile'
#url = 'http://192.168.168.3:5000/api/getfiles'
headers = {'Content-Type': 'application/json'}

# Set up connection parameters
credentials = pika.PlainCredentials('DemoUser', 'DemoUsrPaswrd1984')
parameters = pika.ConnectionParameters('10.20.20.3', 30672, 'DemovHost', credentials)

# Create a connection
connection = pika.BlockingConnection(parameters)

# Create a channel
channel = connection.channel()

#specify queue you want to read from
queue_name = 'Demo.q.direct'
file_type_param = 'geojson'
# Declare the queue
channel.queue_declare(queue=queue_name, durable = True)

# Define a callback function to handle received messages
def callback(ch, method, properties, body):
    body_str = body.decode('utf-8')
    try:
        # Deserialize the JSON string to a dictionary
        body_dict = json.loads(body_str)
    except json.JSONDecodeError:
        print("Failed to decode message body as JSON.")
        return
    print("Received message: %r" % body_dict['ccp_info']['id'])
    data = {"id": body_dict['ccp_info']['id']}
    #====================
    response = requests.post(url, headers=headers, data=json.dumps(data))

    if response.status_code == 200:
        try:
            json_data = response.json()
            with open(f"data-ccp-new.{file_type_param}", 'w') as file:
                json.dump(json_data, file, indent=4)
            print(f"Response saved successfully as a {file_type_param} file.")
            ch.basic_ack(delivery_tag=method.delivery_tag)
        except ValueError:
            print("Response is not in valid {file_type_param} format.")
    else:
        print("Error occurred. Status Code:", response.status_code)
    #====================

# Start consuming messages
channel.basic_consume(queue=queue_name, on_message_callback=callback, auto_ack=True)

print("Waiting for messages. To exit, press CTRL+C")
```

Figure 15: Retrieval of Data from Message Queues

```
[
  {
    "operation": "default",
    "spec":
    {
      "ccp_info":
      {
        "id": "${filename}",
        "endpoint": "http://10.20.20.3:30666/api/getfile
      }
    }
  }
]
```

Figure 16: The format of added filed to the metadata to enable CCP using NiFi JoltTransformJSON Processor

2.3.3.4 SAL – Metadata Query - Data Retrieval User Products

The SILVANUS SAL offers a metadata interface (i.e., the query interface in Figure 17) that enables SILVANUS services to search the Metadata Index and locate the specific data objects they need to fulfil their respective functions. This interface also provides the capability to obtain additional metadata related to the requested object(s). When querying the Metadata Index (Steps 1-4 in Figure 17 the response comprises a list of metadata entries that satisfy the specified query constraints. Among the available fields within the metadata, the 'id' field is particularly relevant, as it can be utilized to retrieve the corresponding data object through the data retrieval interface (Steps 5-8 in Figure 17).

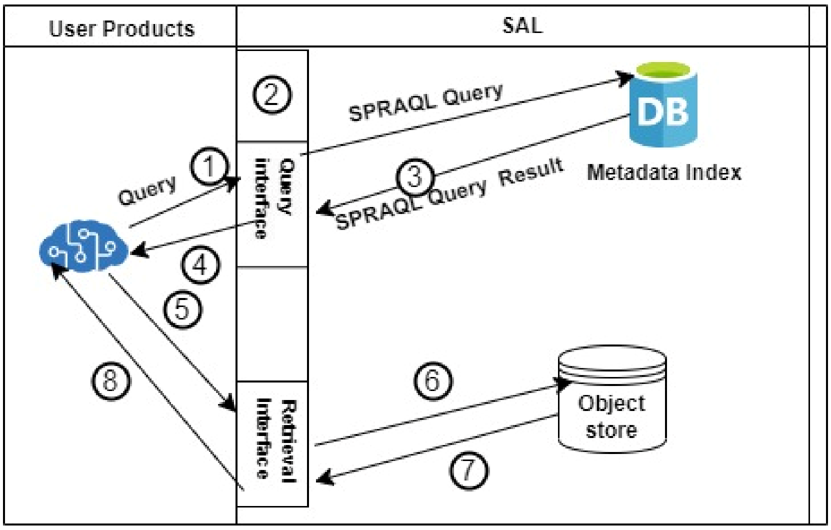


Figure 17: The workflow for the data retrieval solution

In Figure 18, an example of the query format for a user product is displayed, representing Step 1 as depicted in Figure 17. It is noteworthy that the query format shares similarities with the metadata input, both in terms of the format itself and the fields utilized within the query. This alignment in format and fields allows for consistency and ease of use between the user product query and the associated metadata input.

```
[redacted]@silvanuscr12-5:~$ curl -X POST -H "Content-Type: application/json" -d '{"descriptor":{"obj-class":"IoT","format":{"type":"json","output":"json"}},"spatial":{"pilot":"greek"}}' http://10.20.20.3:31555/api/getinfo
```

Figure 18: Query format

Figure 19 displays a portion of the response corresponding to the query depicted in Figure 18. It provides an excerpt of the response that was generated as a result of executing the query, presenting relevant information or data related to the query criteria.


```
  "results": [
    {
      "descriptor": {
        "created": "1674574406.7829435",
        "dataset-type": "air-quality",
        "format": {
          "event": null,
          "output": "json",
          "resolution": "100",
          "type": "json"
        },
        "id": "silvanus-ld:eo:d015ddd1-20c0-48f3-9be8-8ac8ba65cd6d",
        "obj-class": "IoT"
      },
      "spatial": {
        "bbox": "POLYGON ((16.0295831170816712 41.9183734508349062, 16.0295831170816712 41.
32522880823319, 16.0872243646491313 41.9183734508349062, 16.0295831170816712 41.91837345083
        "pilot": "greek"
      },
      "temporal": {
        "daterange": "from:to",
        "datetime": "latest"
      }
    },
```


Figure 19: An example of query results

Figure 20 demonstrates a sample Python code that downloads a file with the id 'silvanus-ld:eo:d015ddd1-20c0-48f3-9be8-8ac8ba65cd6d' from SAL.

```python
import requests
import json

url = 'http://10.20.20.3:31222/api/getfiles'
headers = {'Content-Type': 'application/json'}
data = [{"id": "silvanus-ld:eo:d015ddd1-20c0-48f3-9be8-8ac8ba65cd6d"}]

response = requests.post(url, headers=headers, data=json.dumps(data))

if response.status_code == 200:
    try:
        json_data = response.json()
        with open('response.json', 'w') as file:
            json.dump(json_data, file, indent=4)
        print("Response saved successfully as a JSON file.")
    except ValueError:
        print("Response is not in valid JSON format.")
else:
    print("Error occurred. Status Code:", response.status_code)
```

Figure 20: File download request example

The demo demonstrated in the 5th GA can be found in [5].

For more information about data sources please refer to the following deliverables D4.1 [6], D4.2 [7], D4.3 [8] [9], D4.4 [9].

2.3.3.5 Multi-queue consumption code – UTH integration

This section focuses on the consumption of messages from various RabbitMQ queues, processing the incoming data, and organizing it according to particular criteria or content types.

The Python code, configuration file, and instructions are available at [10].

2.3.3.6 RabbitMQ queue size

The pika python package used in Figure 14 and Figure 15 has the ability to check how many messages are present inside a queue. An example of this can be seen in Figure 21. Once the queue is declared then the command ".method.message_count" will retrieve the number of messages found inside a queue. This does not allow one to see the contents of messages.

```
import pika
# Set up connection parameters
credentials = pika.PlainCredentials('DemoUser', 'DemoUsrPaswrd1984')
parameters = pika.ConnectionParameters('10.20.20.3', 30672, 'DemovHost', credentials)

# Create a connection
connection = pika.BlockingConnection(parameters)

# Create a channel
channel = connection.channel()

#declare queue you want
q = channel.queue_declare(queue='Demo.q.direct', durable = True)
#Get number of messages inside queue
q_len = q.method.message_count
#print number of messages inside queue
print(q_len)
```

Figure 21: Number of messages inside a queue

2.3.4 Update Function

The SAL system allows users to update existing data in the database. This can be done by sending the new data under the old UUID to a specific url. An example of code for this would be Figure 14 with the following url http://10.20.20.3:30515/metadata/ingest

2.3.5 Delete on Demand

The Delete on Demand feature can be used to remove metadata and S3 data associated with a specific UUID. Please refer to Figure 22 for the metadata format. Figure 23 displays a list of UUIDs that need to be deleted separated by enter. These UUIDs should be saved in a TXT file. This should be sent to the address http://10.20.20.3:30515/metadata/ingest.

```
{
"uuid":"Delete"
}
```

Figure 22: Delete on Demand metadata

```
01879876-66ea-7cd4-9bac-3b3b400517a9
01879876-66ea-7cd4-9bac-3b3251d51846
45245526-2465-5sa4-46ga-564a78c4d461
7512a156-bc48-6av1-798b-cda918479aa8
```

Figure 23: Delete on Demand data.

Dummy data can be marked with an expiry flag seen highlighted in Figure 24. This data will be deleted regularly.

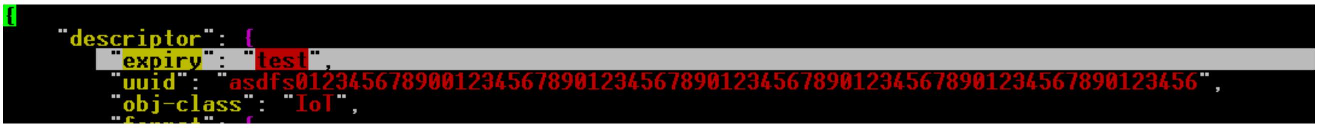


Figure 24: Data marked with uuid flag

2.4 SILVANUS platform cloud infrastructure

In order to host the development environment described in Sections 0 and 2.2 and deploy the SILVANUS components reported in Section 1.2 a Kubernetes cluster has been installed on nodes provided by Hetzner Cloud VPS hosting[3]. The overall process involved several steps, including setting up the Hetzner Cloud environment, provisioning the virtual machines (VMs), and then installing and configuring Kubernetes on these VMs. The resulting topology is shown in Figure 25.

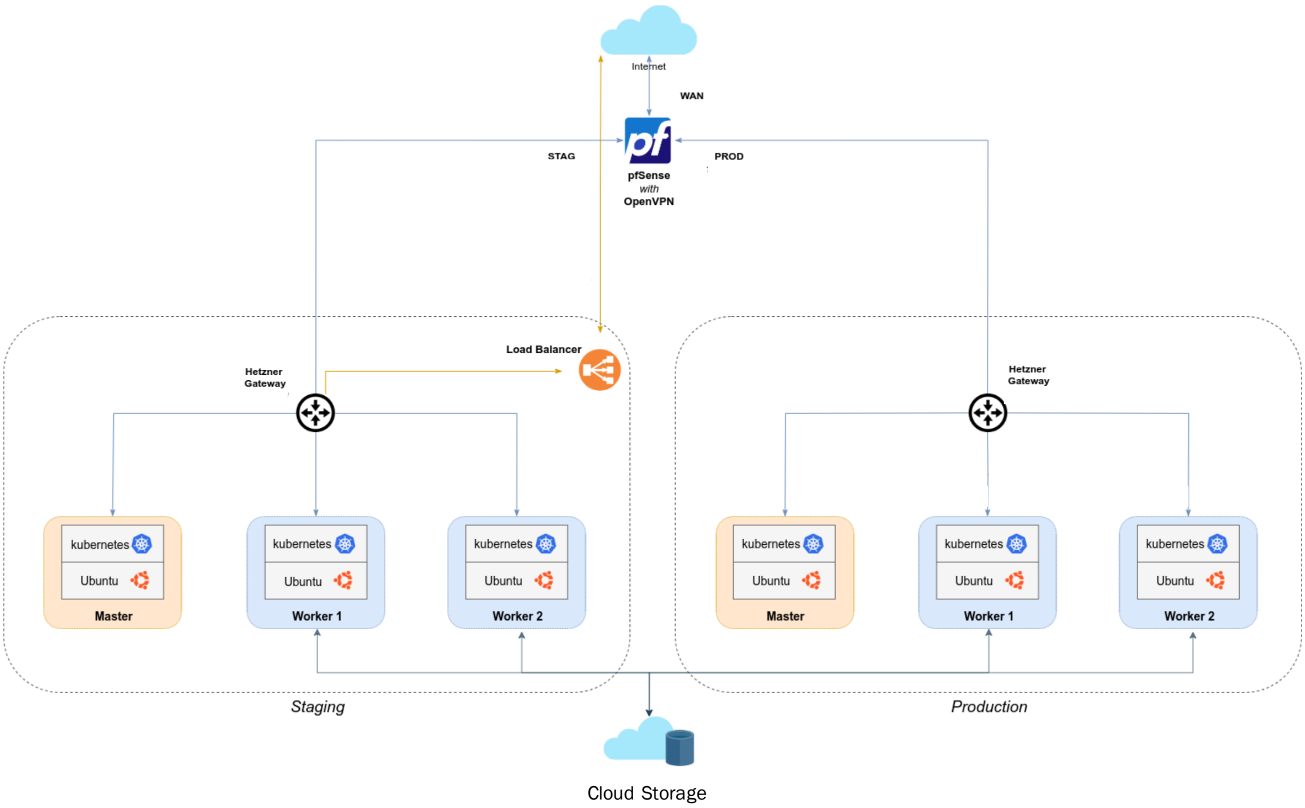


Figure 25: SILVANUS hosted cloud infrastructure (staging and production Kubernetes clusters)

To implement a robust security framework that includes network segmentation, traffic filtering, VPN access, intrusion detection, and comprehensive monitoring we integrated pfSense Open-Source Firewall and router software with the SILVANUS Kubernetes cluster as shown in Figure 25. This layered approach significantly enhances the security posture of the Kubernetes environment. The set-up of pfSense included a number of steps as follows:

- pfSense installation and basic configuration,
- Network Configuration (creating VLANs for Kubernetes Components and interface configuration),

[3] https://www.hetzner.com/cloud/

- Firewall rule creation to restrict access to the Kubernetes API server and isolate Kubernetes nodes,
- Setting up of NAT and port forwarding,
- Configuration of intrusion detection and prevention,
- Configuration of monitoring and alerts.

An instance of the SILVANUS pfSence dashboard is shown in Figure 26.

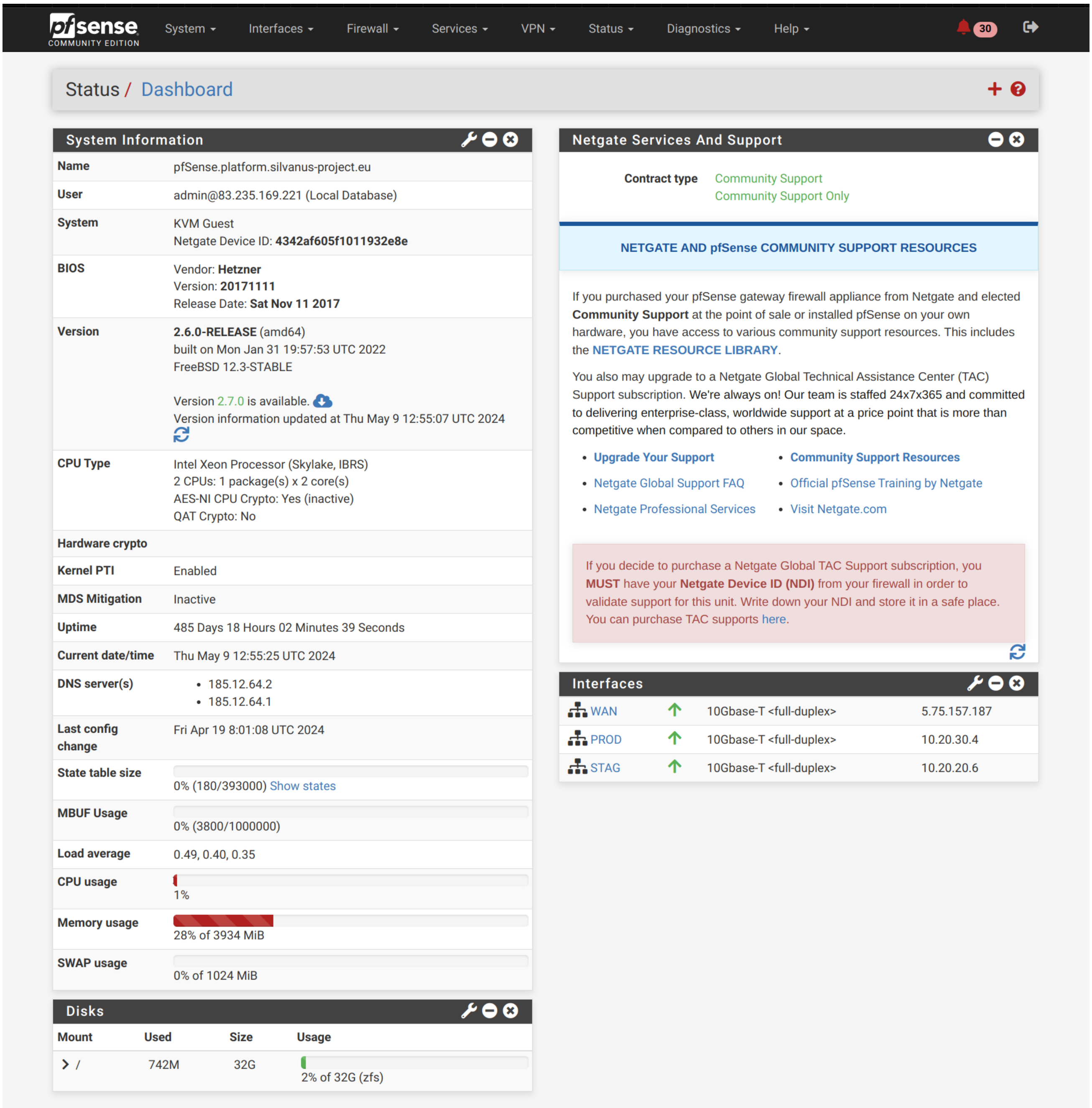
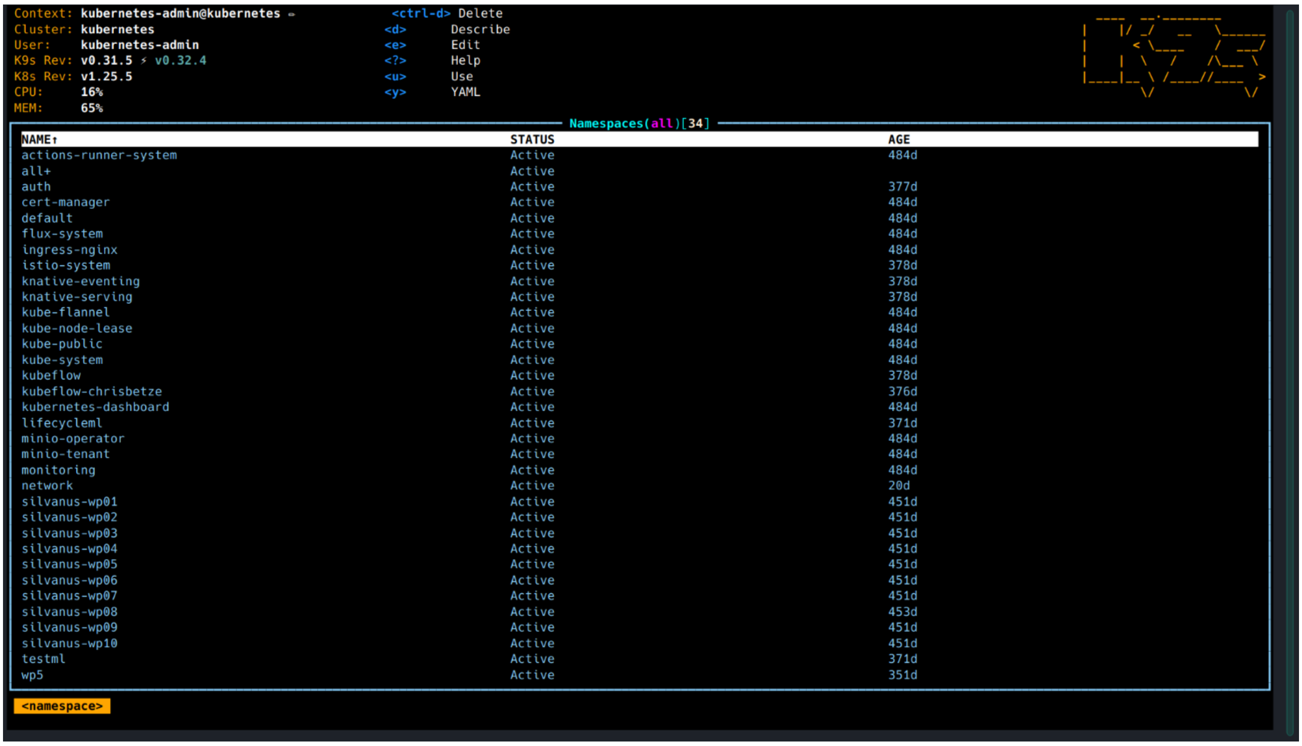


Figure 26: pfSense dashboard

In Figure 27 we show the Kubernetes nodes of the staging cluster and the namespaces on them.

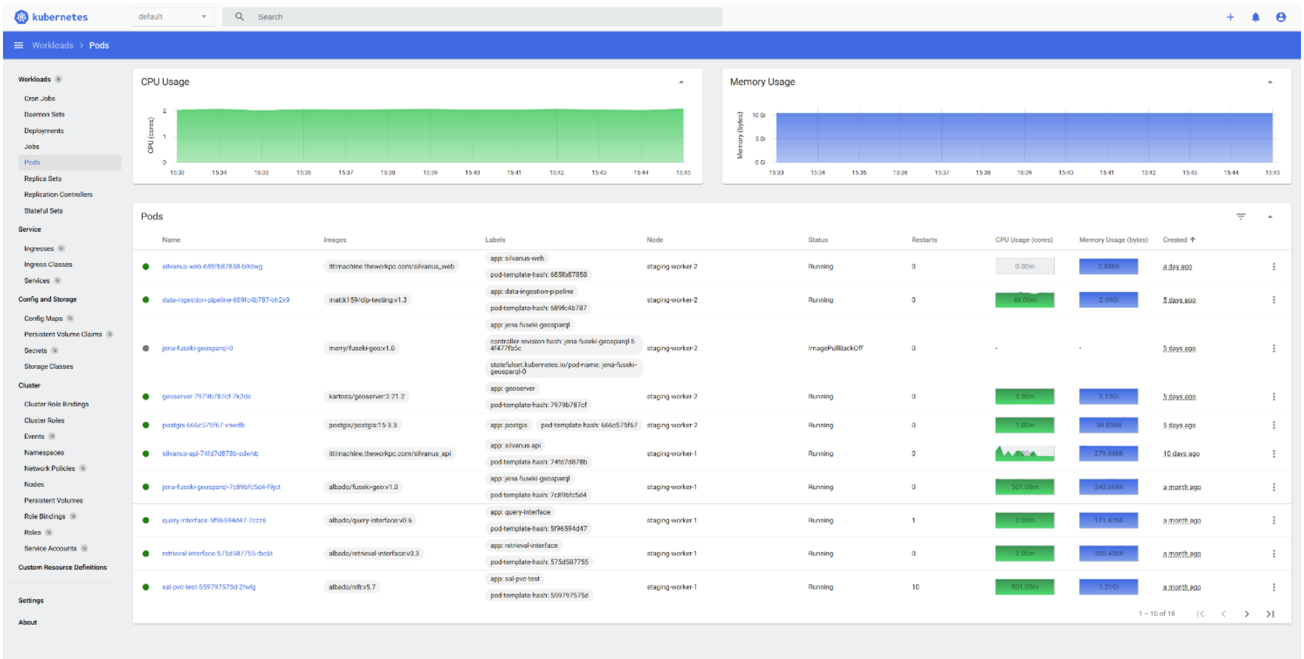
(a)



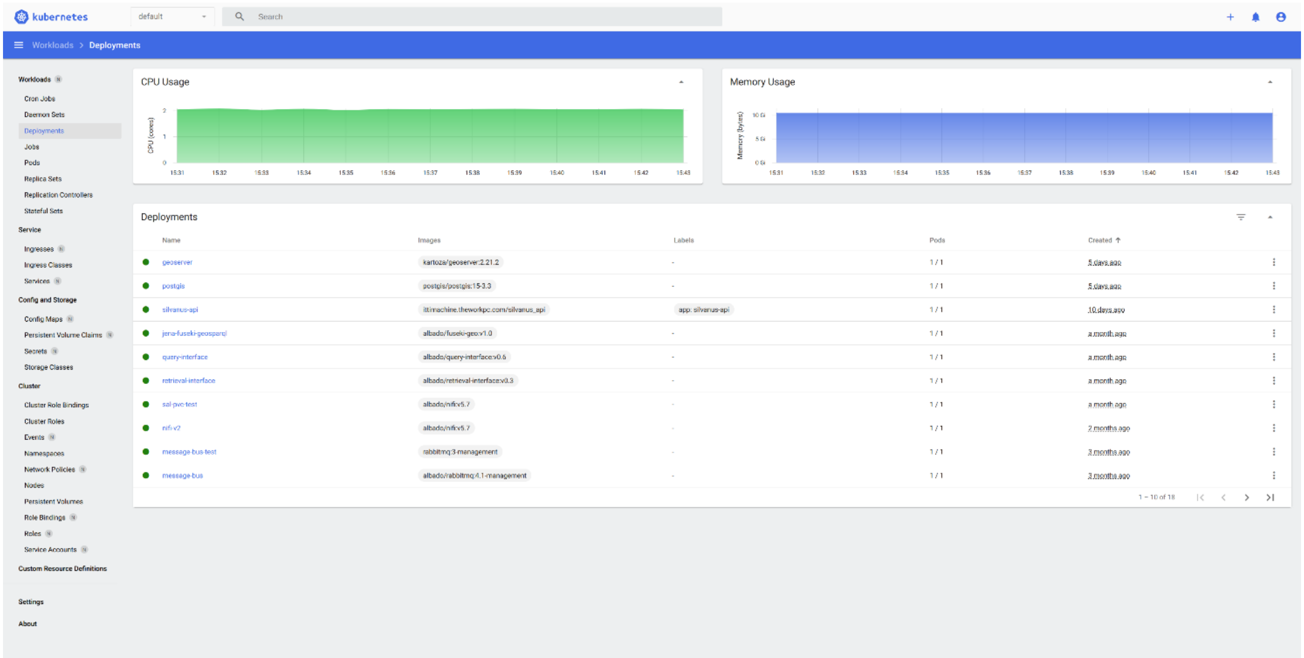
(b)

Figure 27: a) Kubernetes nodes of the staging cluster and b) the namespaces on them

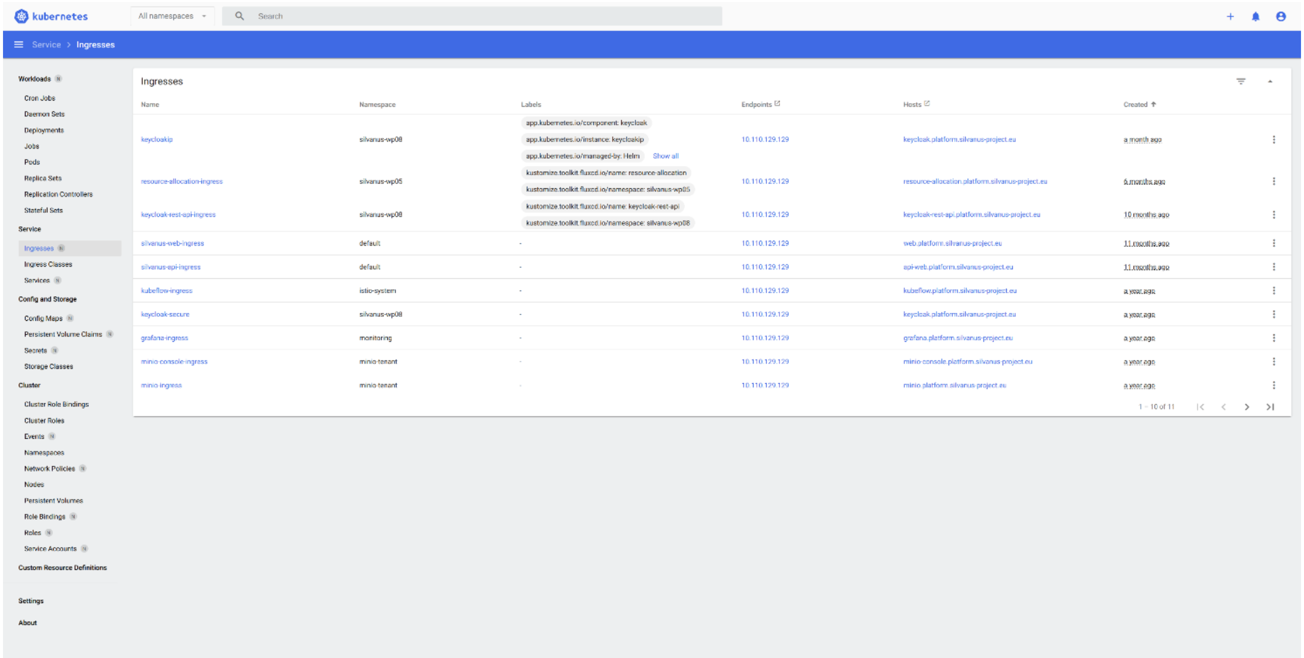
Having configured the SILVANUS cluster of Kubernetes nodes we monitor the status of deployments using the Kubernetes dashboard, which provides a visual and interactive way to manage and monitor various resources within the Kubernetes cluster, including Pods, Deployments, Ingress resources, and Workloads. An example snapshot of the SILVANUS platform monitoring the above resources is shown in Figure 28.



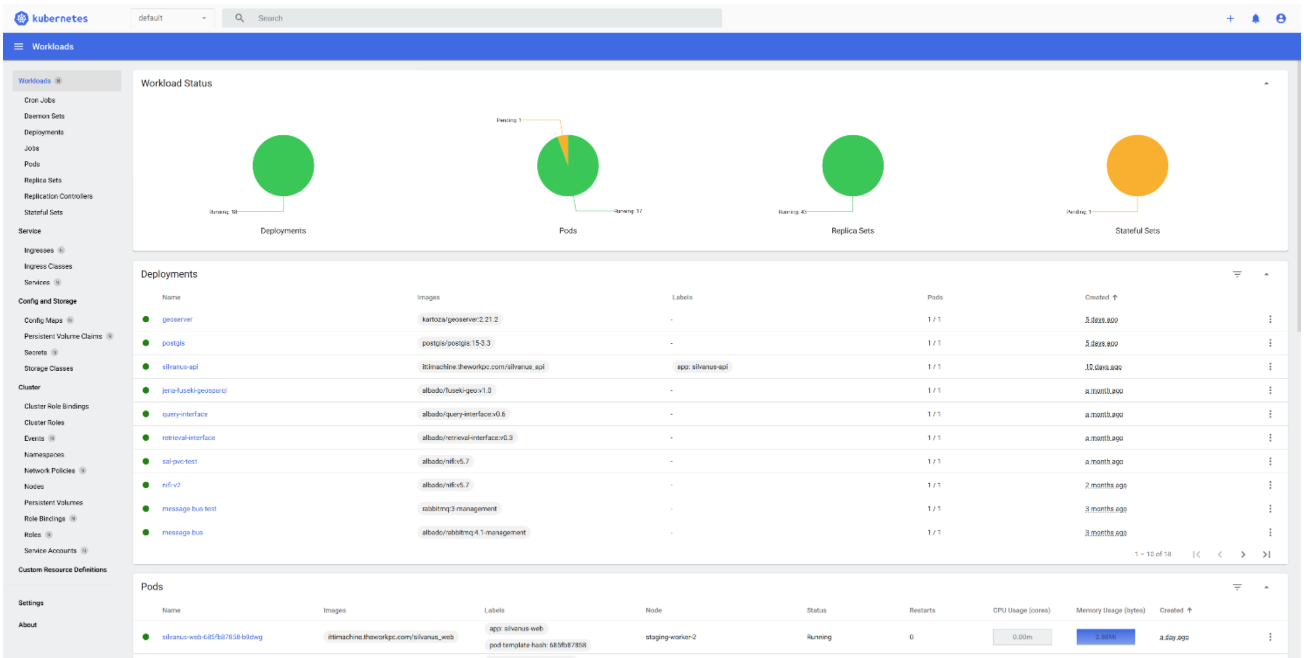
(a)



(b)



(c)



(d)

Figure 28: Snapshot of the SILVANUS platform monitoring a) Pods, b) Deployments, c) Ingress resources, d) and Workloads.

Managing storage is distinct from managing compute instances. The PersistentVolume subsystem in Kubernetes provides an API to abstract the details of storage provision and consumption through two key resources: PersistentVolume (PV) and PersistentVolumeClaim (PVC).

- PersistentVolume (PV): A PV is a storage unit in the cluster, provisioned by an administrator or dynamically via Storage Classes. It is independent of any specific Pod and can use various storage backends like NFS, iSCSI, or cloud-based storage.
- PersistentVolumeClaim (PVC): A PVC is a user's request for storage, specifying size and access modes (e.g., ReadWriteOnce, ReadOnlyMany). PVCs consume PV resources similar to how Pods consume node resources.

To accommodate different storage needs, such as varying performance requirements, the StorageClass resource allows cluster administrators to offer diverse types of PVs without exposing implementation details to users. In Figure 29.

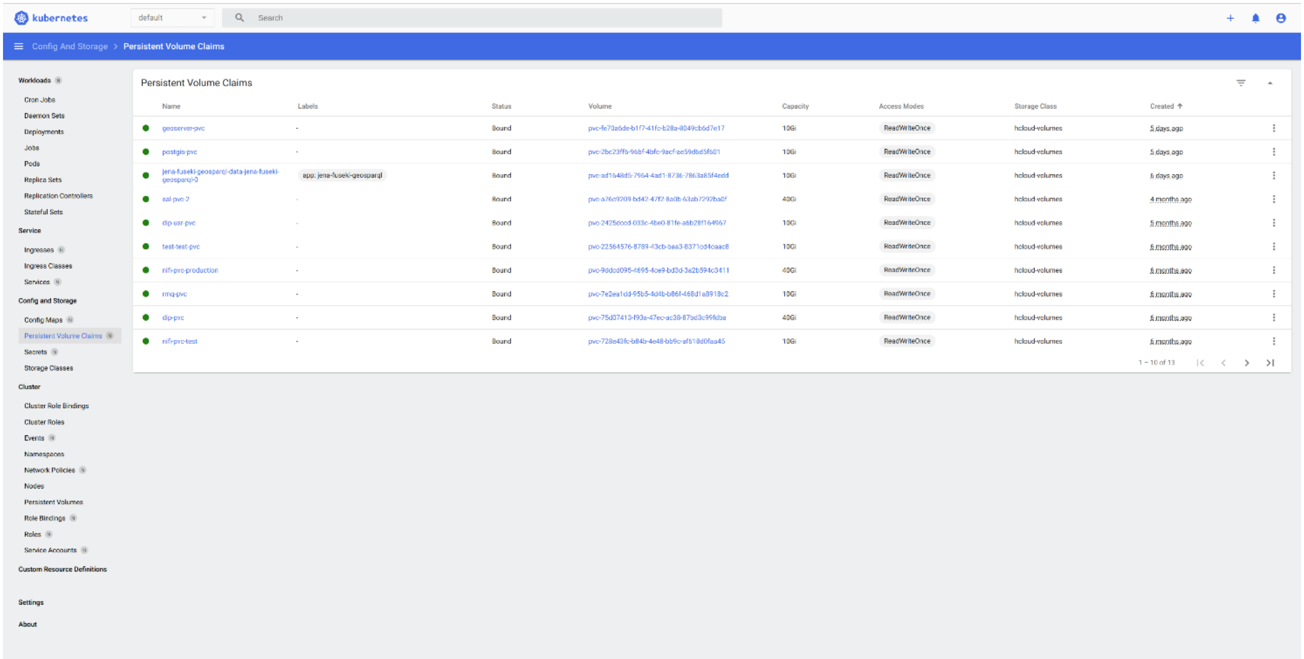


Figure 29: Snapshot of Persistent Volume Claims (PVCs) on the SILVANUS cluster.

Section 2.3.1.1.1 described the use of MinIO object store to serve as the central repository within SILVANUS for storing both raw and processed data managed through the standard S3 storage API. An instance of its usage demonstrating the number of buckets deployed, objects stored, servers and drives deployed is shown in Figure 30.

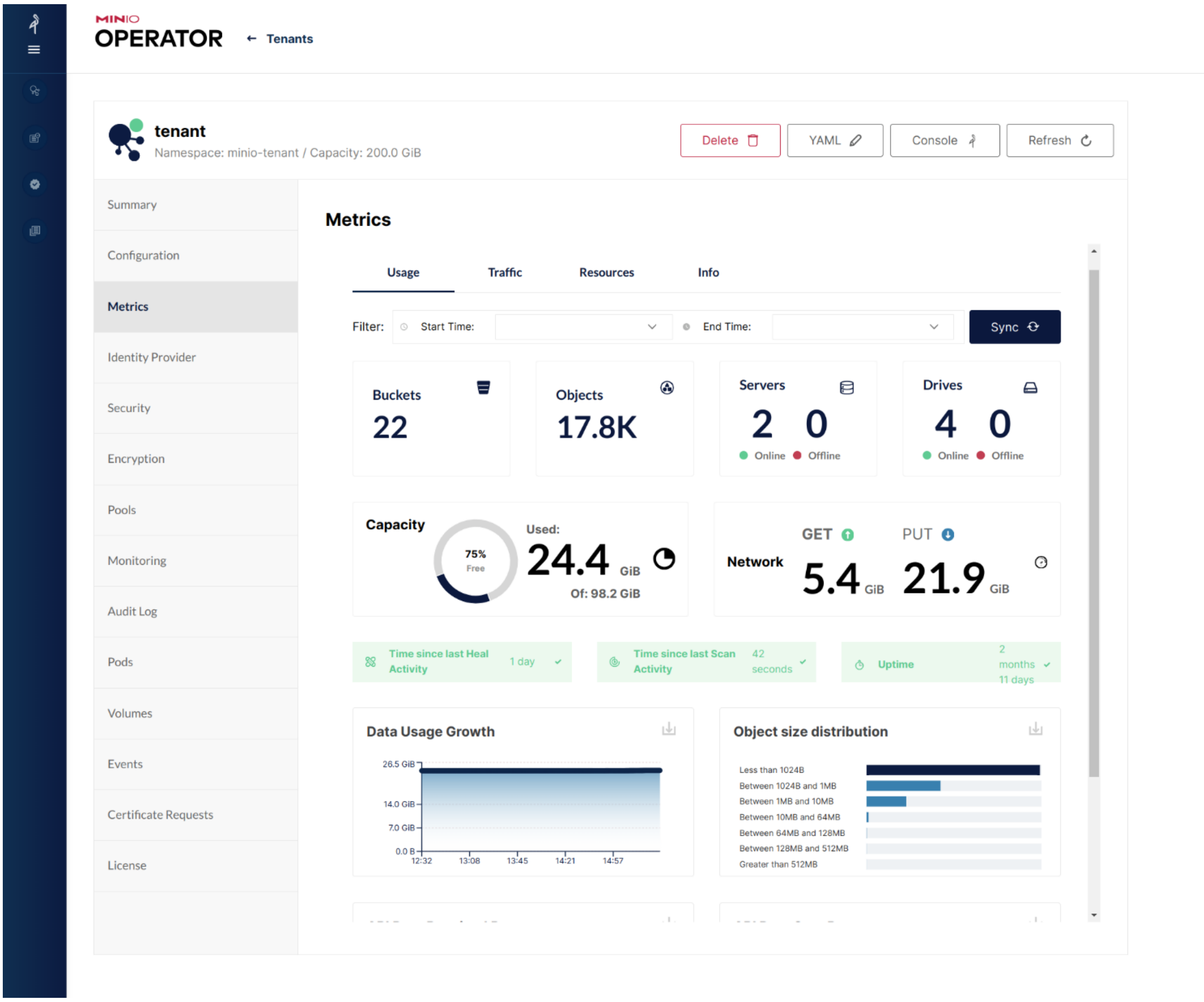
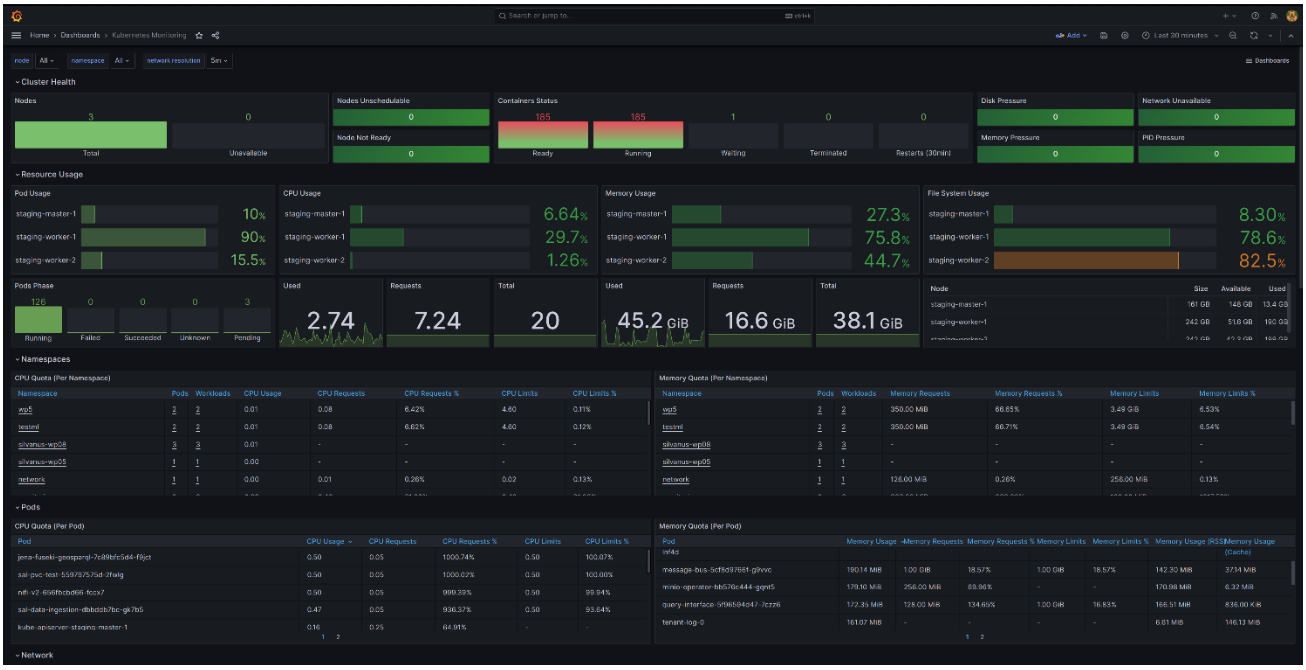


Figure 30: Monitoring instance of minIO object store on the SILVANUS cluster.

To visualize and manage Kubernetes cluster metrics we used Grafana, which is a popular open-source platform for monitoring and observability with a number of pre-built and custom dashboards that can help monitor various aspects of the cluster's health, performance, and resource utilization. A screenshot of the SILVANUS platform resource visualization on the Grafana dashboard is shown in Figure 31.



(a)

(b)

Figure 31: a) SILVANUS cluster performance metrics monitored and b) SILVANUS VPN network statics on Grafana

Since the SILVANUS platform evolves over time following the DevOps methodology described in section 2.2 the amounts of resources listed above and their utilization changes dynamically over time. Therefore, the SILVANUS cluster may scale dynamically in time exploiting the flexibility of the Kubernetes ecosystem. The latest results on the SILVANUS platform will be reported on the final version of this deliverable of T8.5 (D8.5) documenting the final platform release.

3 Conclusions

The current deliverable provides a summary of the software components that comprise SILVANUS platform version 2, which has been developed based on the final reference architecture described in D8.3. The details of each component exist in the relevant space in the GitHub.

4 References

[1] SILVANUS D8.3 Report on final SILVANUS reference architecture, 03/2024

[2] GitHub, "GitHub," GitHub, [Online]. Available: https://github.com/about [Accessed 17 05 2022]

[3] Git, "Git," [Online]. Available: https://git-scm.com/ [Accessed 24 04 2019].

[4] https://venakatreleaf.sharepoint.com/:x:/r/sites/silvanus-ga/Shared%20Documents/General/Work%20Packages/WP5/T5.1/metadata_field_meaning.xlsx?d=w217b70c16b8041eb8467c7690c87b2cf&csf=1&web=1&e=IXEjuZ

[5] https://venakatreleaf.sharepoint.com/sites/silvanus-ga/Shared%20Documents/Forms/AllItems.aspx?csf=1&web=1&e=Wk5Hiq&cid=d3cc6c3e%2D3e07%2D4209%2Dbf21%2Db085ed3bbcb1&RootFolder=%2Fsites%2Fsilvanus%2Dga%2FShared%20Documents%2FGeneral%2FWork%20Packages%2FWP5%2F5%2Dth%20GA&FolderCTID=0x01200021B059F7C183DD4BAA105110A018229B

[6] D4.1 available at: https://venakatreleaf.sharepoint.com/:w:/r/sites/silvanus-ga/_layouts/15/Doc.aspx?sourcedoc=%7BF5BF8E62-10F5-461B-AB7A-6A45F796F8A6%7D&file=SILVANUS_D4.1_v1.0.docx&action=default&mobileredirect=true

[7] D4.2 available at: https://venakatreleaf.sharepoint.com/:w:/r/sites/silvanus-ga/_layouts/15/Doc.aspx?sourcedoc=%7B9AA1B55D-068A-4D45-92ED-8F688B0D26B5%7D&file=SILVANUS_D4.2%20-%20Demonstration%20of%20social%20media%20analytics%20for%20localising%20the%20origin%20of%20wildfire%20ignition_v1.pdf.docx&action=default&mobileredirect=true

[8] D4.3 available at: https://venakatreleaf.sharepoint.com/:w:/r/sites/silvanus-ga/_layouts/15/Doc.aspx?sourcedoc=%7B9AA1B55D-068A-4D45-92ED-8F688B0D26B5%7D&file=SILVANUS_D4.2%20-%20Demonstration%20of%20social%20media%20analytics%20for%20localising%20the%20origin%20of%20wildfire%20ignition_v1.pdf.docx&action=default&mobileredirect=true

[9] D4.4 available at: https://venakatreleaf.sharepoint.com/:w:/r/sites/silvanus-ga/_layouts/15/Doc.aspx?sourcedoc=%7B6A3D2868-2E92-4F9B-944C-32BEBE17E5D9%7D&file=D4_4_V1_0.docx&action=default&mobileredirect=true

[10] https://venakatreleaf.sharepoint.com/:u:/r/sites/silvanus-ga/Shared%20Documents/General/Work%20Packages/WP5/5-th%20GA/UTH-integration.zip?csf=1&web=1&e=XboZ02

[11] https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

[12] source: https://dev.to/techschoolguru/how-to-setup-github-actions-for-go-postgres-to-run-automated-tests-81o

[13] source: https://www.docker.com/resources/what-container/

[14] [source: https://vnclagoon.com/gitops-why-your-company-should-embrace-it/]

Appendix 1. Example of integration workflow

In the example-app[4] github repository we have a hello world test application that uses:

- **GiHub Actions**: For the CI steps of our pipeline. (The CI steps of the staging pipeline are shown in Figure 32 while for the production pipeline in Figure 33.)
- **Flux**: For the CD steps of our pipeline in a GitOps manner.

Repository Structure

The Git repository contains the following directories:

- **clusters/** directory contains the Flux configuration per cluster
- **deploy/base/** directory contains common infra tools and configurations same for both clusters
- **deploy/staging/** directory contains the staging .yaml configurations
- **deploy/production/** directory contains the production configurations

Dockerfile

We create a *Dockerfile* file in the root of the repository.

```
FROM node:8
WORKDIR /app
ADD . /app
RUN npm install
EXPOSE 3000
CMD npm start
```

Kubernetes Manifest

We create the k8s *.yaml* configuration files in the *deploy* directories depending on the cluster we want to deploy.

Deployment

- In the deploy/staging directory:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-app
  labels:
    app: example-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: example-app
  template:
    metadata:
      labels:
        app: example-app
    spec:
```

[4] https://github.com/silvanus-prj/example-app

```
    containers:
    - name: example-app
      image: silvanusproject/example-app:v1.0.1
      imagePullPolicy: IfNotPresent
      ports:
      - name: nodejs-port
        containerPort: 3000
    imagePullSecrets:
    - name: regcred
```

- In the deploy/production directory we have the same deployment, but with 3 replicas instead of 1. The image tag is e.g. *example-app:RELEASE-v1.0.1.*

Service

In the deploy/base directory we have the ClusterIP service:

```
apiVersion: v1
kind: Service
metadata:
  name: example-app-service
spec:
  ports:
  - port: 31001
    targetPort: nodejs-port
    protocol: TCP
  selector:
    app: example-app
```

Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-app-ingress
spec:
  ingressClassName: nginx
  tls:
  - hosts:
      - example-app.platform.silvanus-project.eu
  rules:
  - host: example-app.platform.silvanus-project.eu
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: example-app-service
            port:
              number: 31001
```

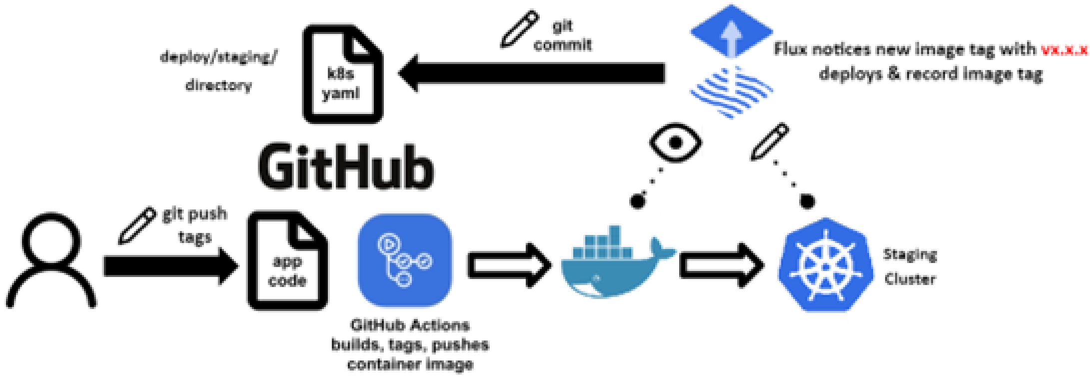


Figure 32: Staging CI/CD

CI/CD Pipelines

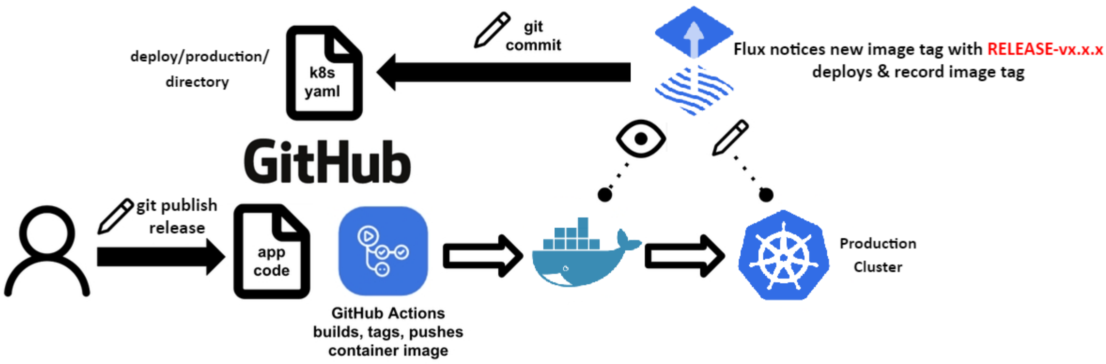


Figure 33: Production CI/CD

Repository Secrets

We should add some actions secrets for log-in to DockerHub. These are shown in Figure 34. In the GitHub Repository, go to Settings -> Secrets -> Actions

- `DOCKERHUB_USERNAME: silvanusproject`
- `DOCKERHUB_TOKEN: **********`

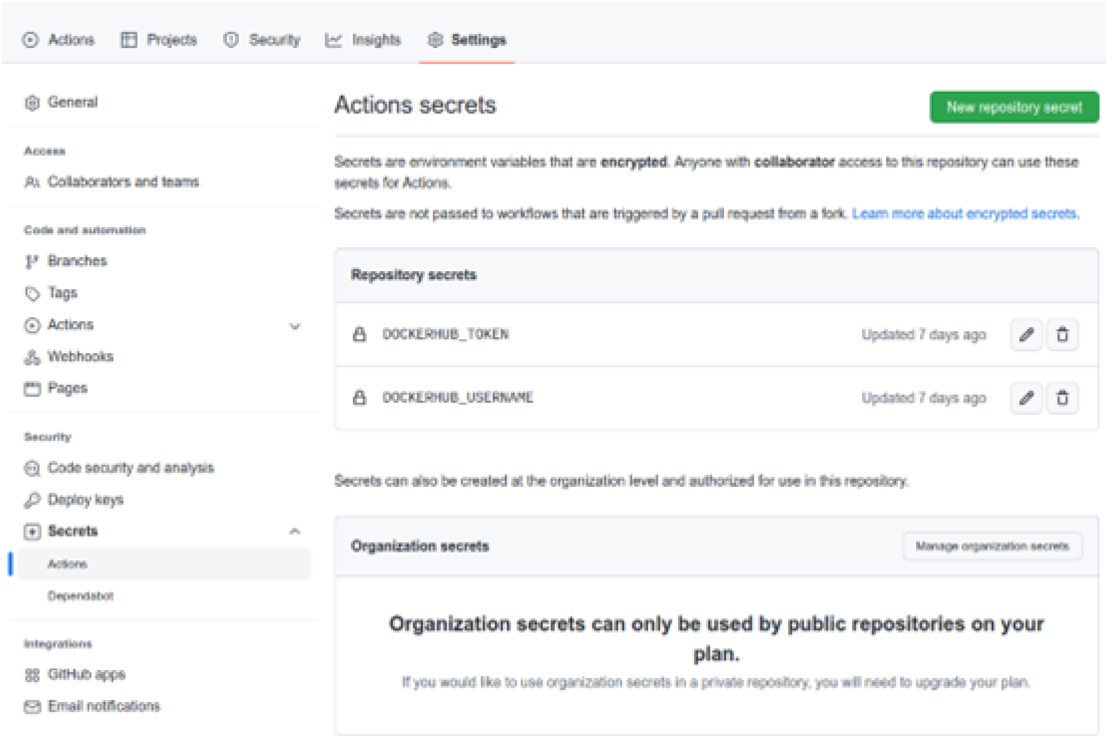


Figure 34 Repository Secrets

CI with GitHub Actions

We create a *ci.yml* file in the *.github/workflows* directory of the repository.

Events that trigger the workflow:

```
name: ci
on:
  release:
    types: [published]
  push:
    branches: [ "main" ]
    tags: [ 'v*.*.*' ]
```

For example, this workflow will run when someone pushes to main, pushes tags or publishes a release.

Environment variables used from job:

```
env:
  REGISTRY: docker.io
  IMAGE_NAME: ${{secrets.DOCKERHUB_USERNAME}}/
${{github.event.repository.name }}
```

Job & Steps:

```
jobs:
  docker:
    if: github.event.head_commit.author.name != 'fluxcdbot'
    runs-on: [self-hosted]
```

```
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Set up QEMU
        uses: docker/setup-qemu-action@v2

      - name: Setup Docker buildx
        uses: docker/setup-buildx-action@v2
        with:
          driver: docker

      - name: Test
        run: echo "::debug::Here goes your test actions"

      - name: Lint
        run: echo "::debug::Here goes your lint actions"

      - name: Log into DockerHub
        if: github.ref_name != 'main'
        uses: docker/login-action@v2
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}

      - name: Extract Docker metadata
        id: meta
        uses: docker/metadata-action@v4
        with:
          images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}

      - name: Build and Push Docker image (main and tags)
        if: ${{ github.event_name != 'release' }}
        id: build-and-push-tags
        uses: docker/build-push-action@v3
        with:
          context: .
          push: ${{ github.ref_name != 'main' }}
          tags: ${{ steps.meta.outputs.tags }}

      - name: Build and Push Docker image (releases)
        if: ${{ github.event_name == 'release' }}
        id: build-and-push-releases
        uses: docker/build-push-action@v3
        with:
          context: .
          push: ${{ github.ref_name != 'main' }}
          tags: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:RELEASE-${{
github.ref_name }}, ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:latest
```

For example, this job will run the following steps on a self-hosted runner according to event trigger.

- Pushes to main
  - Checkout
  - Setup Requirements
  - Test
  - Lint
  - Docker Build
- Pushes tag
  - Checkout

  - Setup Requirements
  - Test
  - Lint
  - Docker Build
  - Docker Push with tags: *latest* and *tag_name*
    (silvanusproject/example-app:latest, silvanusproject/example-app:v1.0.1)

- Publishes a release from a tag
  - Checkout
  - Setup Requirements
  - Test
  - Lint
  - Docker Build
  - Docker Push with tags: *latest* and *RELEASE-tag_name*
    (silvanusproject/example-app:latest, silvanusproject/example-app:RELEASE-v1.0.1)

The container image tags will used for the deployment of our application.

CD with Flux

We follow the [Automate image updates to Git] guide from official Flux docs in order to automate the deployment stage of our application to staging and to production cluster.

We configure Flux to:

1) Checks the Git repository and produce an artifact for a revision **(GitRepository)**
2) Scan the container registry and fetch the image tags **(ImageRepository)**
3) Select the latest tag based on the *semver* policy **(ImagePolicy)**
4) Replace the tag in Kubernetes manifests, checkout a branch, commit and push the changes to the remote Git repository **(ImageUpdateAutomation)**
5) Apply the changes and rollout the container image (Reconcile **Kustomization**)

Git Repository

Before we deploy the GitRepository, we should create a secret in the same namespace, with our **username** and a GitHub personal access token (**PAT**) with *repo permissions*. See the GitHub documentation on [creating a personal access token].

```
apiVersion: v1
kind: Secret
metadata:
  name: example-app-auth
  namespace: silvanus-wp08
type: Opaque
data:
  username: < Base64_encoded_username >
  password: < Base64_encoded_pat >


apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: GitRepository
metadata:
  name: example-app
  namespace: silvanus-wp08
spec:
```

```
  interval: 5m
  url: https://github.com/silvanus-prj/example-app
  ref:
    branch: main
  secretRef:
    name: example-app-auth
```

- A GitRepository named example-app is created, indicated by the *.metadata.name* field.
- The source-controller checks the Git repository every five minutes, indicated by the *.spec.interval* field.
- It clones the main branch of the https://github.com/silvanus-prj/example-app repository, indicated by the *.spec.ref.branch* and *.spec.url* fields.
- The *.spec.secretRef.name* field specifies the name reference of the above Secret containing the authentication credentials for the Git repository.

For more information see Git Repositories.

Image Repository

We create an Image Repository to tell Flux which container registry to scan for new tags.

```
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImageRepository
metadata:
  name: example-app
  namespace: silvanus-wp08
spec:
  image: silvanusproject/example-app
  interval: 1m0s
  secretRef:
    name: regcred
```

This example fetches metadata for the private image *silvanusproject/example-app* every minute.

For the silvanusproject private images, we have created a Kubernetes secret in the same namespace. So, we configure Flux to use the credentials by referencing the Kubernetes secret in the *.spec.secretRef.name* field.

For more information see Image Repositories.

Image Policy

We create an ImagePolicy to tell Flux which semver range to use when filtering tags

- Staging Cluster: *vx.x.x*

```
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImagePolicy
metadata:
  name: example-app
  namespace: silvanus-wp08
spec:
  imageRepositoryRef:
    name: example-app
  policy:
```

```
    semver:
      range: vx.x.x
```

- Production Cluster: *RELEASE-vx.x.x*

```
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImagePolicy
metadata:
  name: example-app
  namespace: silvanus-wp08
spec:
  imageRepositoryRef:
    name: example-app
  filterTags:
    extract: $version
    pattern: ^RELEASE-(?P<version>v?\d+\.\d+.\d+[a-zA-Z]*)$
  policy:
    semver:
      range: '*'
```

For other policies that make use of CalVer, build IDs or alphabetical sorting, have a look at the examples[5].

Then, we should edit the deployment.yaml and add a marker to tell Flux which policy to use when updating the container image:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-app
  labels:
    app: example-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: example-app
  template:
    metadata:
      labels:
        app: example-app
    spec:
      containers:
      - name: example-app
        image: silvanusproject/example-app:v1.0.1 # {"$imagepolicy":
"silvanus-wp08:example-app"}
        imagePullPolicy: IfNotPresent
        ports:
        - name: nodejs-port
          containerPort: 3000
      imagePullSecrets:
      - name: regcred
```

For more information see Image Policies.

Image Update Automation

[5] https://fluxcd.io/flux/components/image/imagepolicies/#examples

We create an Image Update Automation to tell Flux which Git repository to write image updates to. The ImageUpdateAutomation type defines an automation process that will update a git repository, based on image policy objects in the same namespace. The updates are governed by marking fields to be updated in each YAML file. For each field marked, the automation process checks the image policy named, and updates the field value if there is a new image selected by the policy.

```
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImageUpdateAutomation
metadata:
  name: example-app
  namespace: silvanus-wp08
spec:
  interval: 1m0s
  sourceRef:
    kind: GitRepository
    name: example-app
  git:
    checkout:
      ref:
        branch: main
    commit:
      author:
        email: fluxcdbot@users.noreply.github.com
        name: fluxcdbot
      messageTemplate: '{{range .Updated.Images}}{{println .}}{{end}}'
    push:
      branch: main
  update:
    path: ./deploy/staging
    strategy: Setters
```

The *sourceRef* field refers to the GitRepository object that has details on how to access the Git repository to be updated. The required field *interval* gives a period for automation runs.

Strategy "Setters" uses field markers referring to image policies, as described before. The *.spec.update.path* field specifies the path to the directory containing the manifests to be updated.

- For staging: *./deploy/staging*
- For production: *./deploy/production*

For more information see Image Update Automations[6].

Kustomization

The Kustomization is the most important Custom Resource Definition, because it **reconciles** on the cluster the Kubernetes manifests stored in a Git repository.

```
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
  name: example-app
  namespace: silvanus-wp08
spec:
  interval: 5m
  targetNamespace: silvanus-wp08
  sourceRef:
    kind: GitRepository
```

[6] https://fluxcd.io/flux/components/image/imageupdateautomations/

```
      name: example-app
    path: "./deploy/staging"
    prune: true
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
  name: example-app-base
  namespace: silvanus-wp08
spec:
  interval: 5m
  targetNamespace: silvanus-wp08
  sourceRef:
    kind: GitRepository
    name: example-app
  path: "./deploy/base"
  prune: true
```

- A Flux Kustomization named example-app is created that watches the axample-app GitRepository for artifact changes in the path ./deploy/staging. (For production we have ./deploy/production).
- A Flux Kustomization named example-app is created that watches the example-app GitRepository for artifact changes in the path ./deploy/base. (Same for staging and production).
- The Kustomization builds the YAML manifests located at the specified *spec.path*, sets the namespace of all objects to the *spec.targetNamespace*, validates the objects against the Kubernetes API, and finally applies them on the cluster.
- Every 5 minutes, the Kustomization runs a server-side apply dry-run to detect and correct drift inside the cluster.
- When the Git revision changes, the manifests are reconciled automatically. If previously applied objects are missing from the current revision, these objects are deleted from the cluster when *spec.prune* is enabled.

For more information see Kustomization[7].

Summary

The overall Flux configuration described above can be found in the *kustomization.yaml* file in the *clusters/* directory per cluster.

We can deploy the application with the Flux configuration from the command line:

- **Staging**:
  ```
  kubectl apply -f clusters/staging/kustomization.yaml \
  --kubeconfig=silvanus_staging_config
  ```

- **Production**:
  ```
  kubectl apply -f clusters/production/kustomization.yaml \
  --kubeconfig=silvanus_production_config
  ```

After a few seconds/minutes of applying the previous *.yaml* file in the cluster, we can visit the https://example-app.platform.silvanus-project.eu/ and we should see our test web page (Hello World) with a valid TLS Certificate.

[7] https://fluxcd.io/flux/components/kustomize/kustomization/

Appendix 2. Example Kubernetes Config Files

Template File

```
apiVersion: v1
apiVersion: apps/v1
kind: Deployment
metadata:
  name: __BUILD_PIPELINE__
  labels:
    app: __BUILD_PIPELINE__
spec:
  replicas: 3
  selector:
    matchLabels:
      app: __BUILD_PIPELINE__
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 33%
  template:
    metadata:
      labels:
        app: __BUILD_PIPELINE__
    spec:
      containers:
        - name: __BUILD_PIPELINE__
          image: silvanusproject/__BUILD_PIPELINE__
          imagePullPolicy: Always
          env:
            - name: NODE_ENV
              value: __BUILD_PIPELINE__
          ports:
            - containerPort: __SERVICE_PORT__ #internal
          command: [ "__START_COMMAND__" ]
          args: [ "__START_COMMAND_ARGUMENTS__" ]
      imagePullSecrets:
      - name: regcred
---
apiVersion: v1
kind: Service
metadata:
  name: __BUILD_PIPELINE__-service
  annotations:
    load-balancer.hetzner.cloud/name: "staging-lb"
    load-balancer.hetzner.cloud/use-private-ip: "true"
spec:
  selector:
    app: __BUILD_PIPELINE__
  ports:
  - protocol: TCP
    port: __SERVICE_PORT__ #external
    targetPort: __SERVICE_PORT__ #internal
  type: LoadBalancer

---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
```

```
  name: __BUILD_PIPELINE__-ingress
spec:
  ingressClassName: nginx
  tls:
  - hosts:
      - __BUILD_PIPELINE__.platform.silvanus-project.eu
  rules:
  - host: __BUILD_PIPELINE__.platform.silvanus-project.eu
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: __BUILD_PIPELINE__-service
          serviceName: __SERVICE_PORT__
```

Variables Details

`__namespace__` : Is the network namespace the component should be deployed to. Only components in the same namespace can see each other, without any extra network configuration (silvanus-wpxx)

__BUILD_PIPELINE__ : The name of the application to be deployed.

__START_COMMAND__: The command the container will run on boot.
__START_COMMAND_ARGUMENTS__: The arguments to pass to the command that will run when the container first starts up.

__SERVICE_PORT__ `#external` : The port to expose on the "internet"

__SERVICE_PORT__ `#internal` : The port to expose the container on the local network. This should match the port exposed on the Dockerfile (example: EXPOSE 3000 should mean __SERVICE_PORT__ = 3000)