ADEPTNESS – Design-Operation Continuum Methods for Testing and Deployment under Unforeseen Conditions for Cyber-Physical Systems of Systems

EUROPEAN COMMISSION
Horizon 2020
H2020-ICT-01-2019
GA No. 871319

Deliverable No.	ADEPTNESS D1.1 (ANNEX B)	
Deliverable Title	ANNEX B Continuous Integration, Software Deployment, and Infrastructure Monitoring tools	
Deliverable Date	2020-08-31	
Deliverable Type	Report	
Dissemination level	Public	
Written by	IKERLAN	2020-07-22
Checked by	IKERLAN	2020-07-22
Approved by	General Assembly	2020-08-31
Status	First version	2020-09-07

H2020-ICT-01-2019 – 871319 – ADEPTNESS: Design-Operation Continuum Methods for Testing and Deployment under Unforeseen Conditions for Cyber-Physical Systems of Systems

Acknowledgement

The author(s) would like to thank the partners involved with the project for their valuable comments on previous drafts and for performing the review.

Project partners

1 – MGEP – Mondragon Goi Eskola Politeknikoa – ES

2 – ORO – Orona S. Coop – ES

3 – UES – Ulma Embedded Solutions S. Coop – ES

4 – SRL – Simula Research Laboratory – NO

5 – BT – Bombardier Transportation Sweden – SE

6 – IKL – Ikerlan S. Coop – ES

7 – EGM – Easy Global Market SAS – FR

8 – MDH – Maelardalens Hoegskola – SE

9 – TUW – Technische Universitaet Wien – AT

Disclaimer:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871319.

Document Information

Additional author(s) and contributing partners

Name	Organisation
David Antón	IKL
Haris Isakovic	TUW
Franck Le Gall	EGM

Document Change Log

Name	Date	Comments
V0.1	2020-05-25	Initial draft
V1.0	2020-09-07	First version

Exploitable results

Exploitable results	Organisation(s) that can exploit the result
There are not exploitable results in this document.	

CONTENTS

LIST OF FIGURES

1 PURPOSE OF THE DOCUMENT

The purpose of this document is to analyse relevant tools for the ADEPTNESS project in the following areas: continuous integration, remote software update and deployment, platform monitoring, data visualization and databases.

2 CONTINUOUS INTEGRATION TOOLS

Continuous integration is a software engineering practice that consists of making automatic integrations of a project as often as possible to detect failures as soon as possible. In this case, integration is understood as the compilation and execution of tests of an entire project.

The process happens periodically or when code changes and usually consists on downloading the sources from version control (e.g. CVS, Git) compiling it, running tests, and generating reports.

2.1 Jenkins

Jenkins is a free and open source automation server. Jenkins helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery. It supports version control tools, including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, TD/OMS, ClearCase and RTC, and can execute Apache Ant, Apache Maven and sbt based projects as well as arbitrary shell scripts and Windows batch commands.

Plugins are available for integrating Jenkins with most version control systems and bug databases. Many build tools are supported via their respective plugins. Plugins can also change the way Jenkins looks or add new functionality. There is also a set of plugins dedicated for the purpose of unit testing that generate test reports in various formats (for example, JUnit bundled with Jenkins, MSTest, NUnit, etc.) and automated testing that supports automated tests. Builds can generate test reports in various formats supported by plugins (JUnit support is currently bundled) and Jenkins can display the reports and generate trends and render them in the GUI.

Jenkins Pros:

- Price
- Customization
- Plugins system
- Full control of the system

Jenkins Cons:

- Dedicated server (or several servers) are required. That results in additional expenses. For the server itself, DevOps, etc.
- Time needed for configuration / customization

2.2 GitLab CI

GitLab CI/CD is a tool built into GitLab for software development through the continuous methodologies, Continuous Integration (CI), Continuous Delivery (CD) and Continuous Deployment (CD).

This integration of Gitlab CI with Gitlab is straight forward. Stages and jobs are described in a configuration file (gitlab-ci.yml). The configuration file includes a sequence of stages that will be executed in the specified order. After that, every job is described and configured with different options. Every job is part of a stage and will run parallel with other jobs in the same stage.

GitLab CI Pros:

- Complete integration with GitLab
- No infrastructure required
- Build jobs defines just by a yml file in the repository
- Integration with existing tools like JIRA

GitLab CI Cons:

- Some functionalities only available on paid plans (e.g. advanced security)

2.3 CircleCI

CircleCI is a hosted continuous integration service and delivery platform helps software teams rapidly release code with confidence by automating the build, test, and deploy process. Its server is available on both, cloud-based and self-hosted versions and supports any language that builds on Linux or macOS (including C++, JavaScript, .NET, PHP, Python, and Ruby). In CircleCI, all Jobs are constructed within a single file called "circle.yaml".

CircleCI Pros:

- Fast start
- CircleCI has a free plan for enterprise projects
- Easy and fast to start
- Lightweight, easily readable YAML config
- No need of a dedicated server to run CircleCI

CircleCI Cons:

- CircleCI supports only 2 versions of Ubuntu for free (12.04 and 14.04) and MacOS as a paid part.
- CircleCI out of the box only supports the following programming languages: Go, Haskell, Java, PHP, Python, Ruby/Rails, Scala.

2.4 TravisCI

Travis CI is a hosted continuous integration service used to build and test software projects hosted at GitHub.

Travis CI provides various paid plans for private projects, and a free plan for open source. TravisPro provides custom deployments of a proprietary version on the customer's own hardware.

Travis CI is configured by adding a file with extension *travis.yml*, which is a YAML format text file, to the root directory of the repository. This file specifies the programming language used, the desired building and testing environment (including dependencies which must be installed before the software can be built and tested), and various other parameters.

When Travis CI has been activated for a given repository, GitHub will notify it whenever new commits are pushed to that repository or a pull request is submitted. It can also be configured to only run for specific branches, or branches whose names match a specific pattern. Travis CI will then check out the relevant branch and run the commands specified in *ravis.yml*, which will build the software and run any automated tests. When that process has completed, Travis notifies the developer in the way it has been configured for example, by sending an email containing the test results (showing success or failure), or by posting a message on an IRC channel. In the case of pull requests, the pull request will be annotated with the outcome and a link to the build log, using a GitHub integration.

Travis CI can be configured to run the tests on a range of different machines, with different software installed, and supports building software in numerous languages, including C, C++, C#, Clojure, D, Erlang, F#, Go, Apache Groovy, Haskell, Java, JavaScript, Julia, Perl, PHP, Python, R, Ruby, Rust, Scala, Swift, and Visual Basic. Several high-profile open source projects are using it to run builds and tests on every commit, such as Plone, Ruby on Rails, and Ruby.

Travis CI supports integration with external tools such as coverage analysers or static analysers.

Travis CI Pros:

- Build matrix out of the box
- Fast start
- Lightweight YAML config
- Free plan for open-sourced projects
- No dedicated server required

Travis CI Cons:

- Price is higher compared to CircleCI, no free enterprise plan
- Customization (Some features includes and 3rd party libraries/plugins)

2.5 Comparison

Service	Platform	Hosting		Notes
		Free	Paid	
Jenkins	Runs in Docker containers	On-Premise	---	Customization and 1000+ plugins allow integration with almost any other tool
GitLab CI	Platform Independent	Cloud/On-Premise	Cloud/On-Premise	Fully integrated with GitLab
TravisCI	Platform Independent	Cloud (for opensource projects)	On-Premise	
CircleCI	Runs in Docker containers, Linux and macOS VMs	Cloud	On-Premise	

3 SOFTWARE UPDATE AND DEPLOYMENT

The ability to effectively distribute software updates is a key part of deploying IoT technology. Remote software deployment intends to manage a secure, scalable, and efficient distribution of software updates over big groups of devices. These software updates must be executed in a secure, trackable manner that reduces costs and time over manual deployment efforts.

3.1 Hawkbit

Eclipse Hawkbit is a framework for managing software deployment to edge devices, as well as more powerful controllers and networked gateways.

Hawkbit aims to provide a domain-independent software deployment solution that meets the following objectives:

- Works for most IoT projects
- Focuses on software upgrades in IoT space and handles more complex deployment strategies needed for large-scale IoT projects.
- Capable of operating on its own for simple scenarios, while having the ability to integrate with existing systems and device management protocols.

In addition, it is designed to operate in the cloud and offer the following capabilities:

- Technical scalability: connecting millions of devices and sending terabytes of software worldwide.
- Functional scalability: deployments with hundreds of thousands of individual devices
- Managed device complexity: device topologies within each individual provisioning target.
- Integration flexibility: connect and integrate through various (non-)standardized device management protocols directly or through federated device management.

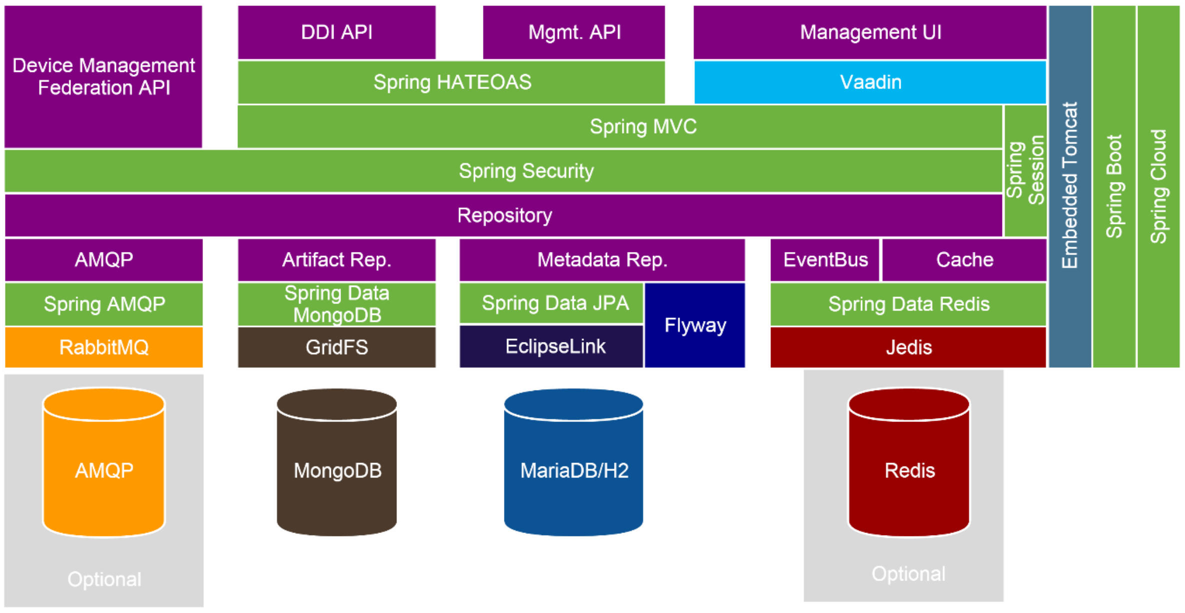

Fig. 1 HawkBit architecture

3.1.1 *Features*

Device and software repository

- Repository containing assignable software procurement targets and distributions.
- Includes a complete history of software updates for each device
- Plug and play support for devices (device is created if it is first authenticated)

Update management

- Direct deployment of a defined software distribution to a device (via the management user interface or API).
- Update management independent of device type, integration approach or connectivity.

Management user interface

- Create/Read/Update/Delete procurement transactions for targets (devices) and repository (software) content
- Manage and monitor software update operations.
- Ease of use of the drag and drop paradigm
- Flexible data grouping.
- Flexible filters for data navigation.

Content delivery

- Partial discharges are allowed.
- Resume download (RFC7233).
- Content management via RESTful API and user interface
- Authorization based on software assignment, i.e. a device can only download what it has been assigned to in the first place.
- Hosting of device signatures is supported.

Deployment Management

- Secure management of large volumes of devices at the time of deployment creation
- Flexible definition of deployment groups.
- Monitoring of deployment progress.
- Emergencies stop of deployment in case of upgrade failure.

Management API

- API RESTful
- Create/Read/Update/Delete procurement transactions for targets (i.e. devices) and tank contents (i.e. software)
- Manage and monitor software update operations
- Online API documentation.
- JSON payload with Hypermedia support.
- Supports filtering, sorting, and paging

Direct device integration API

- API-based RESTful HTTP for direct device integration
- The JSON load.
- Optimized traffic (content based, unmodified Etag generation)
- Device feedback channel.
- TLS encryption.

Device Management Federation API

- Indirect integration of devices through a device management service or an application on HawkBit.
- Optimized for high performance on a service-to-service basis with the AMQP messaging interface.
- A separate AMQP vHost for each tenant for maximum security

3.1.2 Hawkbit Server

Hawkbit Update Server is only responsible to initiate the software rollout (or deployment) of an artefact. Fetching of the software update is the responsibility of the device itself. The device must take any necessary steps and inform Hawkbit update server that it has successfully (or unsuccessfully) downloaded and installed the software update.

The Hawkbit server can be accessed in 4 ways:

- The administration interface: Users can access it at http://localhost:8080/UI/
- Management API: This is the API layer of the management user interface. Anything that can be done with the UI can be done using this API.
- API Direct Device Integration (DDI): This is for devices to register with the HawkBit server, check for updates, notify the HawkBit server of their status and many other things.
- Device Management Federation (DMF) API: Same functionality as DDI. The DMF is based on a publish-subscribe mechanism, using RabbitMQ as message broker.

3.1.3 Hawkbit Client

Hawkbit does not provide a client to work with Hawkbit Server. Instead, the APIs must be used to communicate with the server and implement a client adapted to each device needs. The APIs provide the framework for the client and the client itself must manage how update are performed by checking and reporting information to the Hawkbit Server.

However, there exist several Hawkbit Clients available from different sources, some of not only rely on Hawkbit Server to manage update but also make use of other programs to perform updates on the device once commanded from Hawkbit Server:

- Example Clients based on Feign Project
- RAUC HawkBit client
- Suricatta for SWUpdate
- Update Factory (Kinetycs)

3.2 RAUC

RAUC is a lightweight update client that runs on embedded Linux devices and reliably controls device's update procedure with a new firmware revision. RAUC is also a tool on the host system that allows to create, inspect, and modify upgrade artifacts for the device. It is both a target application that runs as an update client and a host/target tool that allows to create, inspect, and modify installation artifacts.

3.2.1 Features

- D-Bus interface
- SD/eMMC, UBI, raw NAND
- U-boot, grub, barefox, EFI
- Yocto (meta-rauc) and PTXdist support
- HawkBit client for OTA updates

3.3 SWUpdate

SWUpdate is a Linux Update agent with the goal to provide an efficient and safe way to update an embedded system. SWUpdate supports local and remote updates, multiple update strategies and it can be well integrated in the Yocto build system by adding the meta-swupdate layer.

It supports the common media on embedded devices such as NOR/NAND flashes, UBI volumes, SD/eMMC, and can be easily extended to introduce project specific update procedures. An update package is described by a sw-description file, using the libconfig syntax or JSON.

3.3.1 Features

- Install on embedded media (eMMC, SD, Raw NAND, NOR and SPI-NOR flashes)
- Allow delivery single image for multiple devices
- Multiple interfaces for getting software
- local storage
- integrated web server
- integrated REST client connector to HawkBit
- remote server download
- Software delivered as images, gzipped tarball, etc.
- Allow custom handlers for installing FPGA firmware, microcontroller firmware via custom protocols.
- Safe Power-Off
- Hardware / Software compatibility.

3.3.2 Suricatta

Suricatta is a daemon mode of SWUpdate, Suricatta regularly polls a remote server for updates, downloads, and installs them. Thereafter, it reboots the system and reports the update status to the server, based on an update state variable currently stored in bootloader's environment ensuring persistent storage across reboots.

Suricatta is designed to be extensible in terms of the servers supported. Currently, support for the HawkBit server is implemented via the HawkBit Direct Device Integration API.

3.4 Mender

Mender is an open source over-the-air (OTA) software updater for embedded Linux devices. Mender comprises a client running at the embedded device, as well as a server that manages deployments across many devices. This repository contains the Mender Deployments service, which is part of the Mender server.

The Mender server is designed as a microservices architecture and comprises several repositories. The Deployment Service manages artifacts, deployments, and reports of outcome of deployments.

3.4.1 Mender Server

The Mender server stores and controls the deployment of software updates over-the-air to devices. The Mender's web UI or REST APIs can be used to manage devices, upload, and manage software releases to the server and create deployments to roll out software to devices. APIs can also be used to integrate Mender with a CI environment: for example, to automatically upload build output from a CI system to the Mender server and create test deployments. The Mender management server is published on GitHub for on-premise installations. It is licensed under the Apache 2.0 license.

3.4.2 Mender Client

The Mender client runs on the device and reports to the server periodically to check for updates; if there is a software update available for that device, the client downloads and installs it.

3.4.3 Features

For system level updates, a dual A/B rootfs partition layout is required on the device. This ensures that the device can recover even if the deployment is incomplete or corrupted during installation for any reason, e.g. due to power loss during the update process. The Mender client daemon runs in user space in the currently booted rootfs partition.

During the update process, the Mender client writes the updated image to the rootfs partition that is not running and configures U-Boot to boot from the updated rootfs partition. The device is then rebooted. If booting the updated partition fails, the partition that was running is booted instead, ensuring that the device does not get bricked. If the boot succeeds, Mender sets the updated partition to boot permanently when Mender starts as part of the boot process. As Mender downloads and installs the image, other applications on the device continue to run as normal.

The only time the device has downtime is during the reboot into the updated partition. Persistent data can be stored in the data partition, which is left unchanged during the update process. For partial updates, like application updates, the A/B partition is not necessary, and the Mender client can be installed as a .deb package.

3.5 Comparison

Service	Server	Client	Notes
Hawkbit	Web-based	---	Offers no client, but a set of APIs to interact with the server
RAUC	---	Linux	It is a Hawkbit client
SWUpdate	---	Linux	Suricatta mode is compatible with Hawkbit
Mender	Web-based	Linux	Free for open source projects

4 MONITORING

IT infrastructure includes all the assets that are necessary to deliver and support IT services: data centres, servers, networks, computer hardware and software, storage, and other equipment. While the IT infrastructure includes both physical assets and virtual assets (software, virtual machines, virtual servers, etc.). Its purpose is to collect and analyse data from the IT infrastructure and to leverage that data to detect operational issues, identify possible security breaches or malicious attacks.

Hardware monitoring tools capture data from the sensors that can be found in computers and other machines. These can include battery life data, power and load sensors, current and voltage sensors, fan speed sensors and user-defined artificial sensors that collect data on the operating system.

Network monitoring helps to verify that the network is functioning appropriately and delivering the expected levels of speed and performance. IT infrastructure monitoring tools can track the transfer rates and connectivity levels that users are experiencing on the network, as well as monitoring incoming and outgoing connections.

Application monitoring is a critical aspect of IT infrastructure monitoring. Software applications represent a potential attack vector for a malicious actor and a powerful source of operational and business intelligence.

4.1 Prometheus

Prometheus is a free software application used for event monitoring and alerting. It records real-time metrics in a time series database (allowing for high dimensionality) built using a HTTP pull model, with flexible queries and real-time alerting.

Prometheus works well for recording any purely numeric time series. It fits both machine-centric monitoring as well as monitoring of highly dynamic service-oriented architectures. Oriented to microservices, it supports multi-dimensional data collection and querying.

Prometheus is designed for reliability, to be a system that helps to quickly diagnose problems. Each Prometheus server is standalone, not depending on network storage or other remote services. It is reliable even when other parts of the infrastructure are broken.

Prometheus scrapes metrics from instrumented jobs, either directly or via an intermediary push gateway for short-lived jobs. It stores all scraped samples locally and runs rules over this data to either aggregate and record new time series from existing data or generate alerts. Grafana or other API consumers can be used to visualize the collected data.

4.1.1 Architecture

A typical monitoring platform with Prometheus has the following components:

- Prometheus server to scrap, centralize and store the metrics.
- Multiple exporters that typically run on the monitored host to export local metrics (HAProxy, StatsD, Graphite, etc.).
- A push gateway for supporting short-lived jobs

- Alertmanager to trigger alerts based on those metrics.
- Grafana to produce dashboards.
- PromQL is the query language used to create dashboards and alerts.

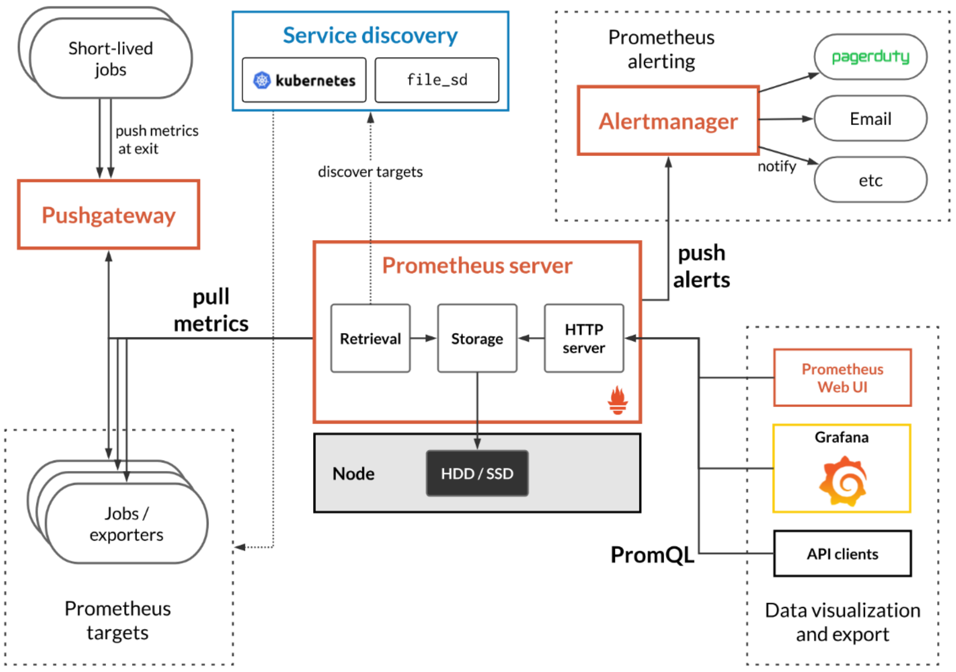

Fig. 2 Prometheus Architecture

4.1.2 Features

- A multi-dimensional data model with time series data identified by metric name and key/value pairs
- PromQL, a flexible query language to leverage this dimensionality
- No reliance on distributed storage; single server nodes are autonomous
- Time series collection happens via a pull model over HTTP
- Pushing time series is supported via an intermediary gateway
- Targets are discovered via service discovery or static configuration
- Multiple modes of graphing and dashboarding support

4.2 Cortex

Cortex is an open-source Prometheus-as-a-Service platform that seeks to fill those gaps and to thereby provide a complete, secure, multi-tenant Prometheus. As a Prometheus-as-a-Service platform, Cortex fills in all these crucial gaps with aplomb and thus provides a complete out-of-the-box solution for even the most demanding monitoring and observability use cases.

- It supports four long-term storage systems out of the box: AWS DynamoDB, AWS S3, Apache Cassandra, and Google Cloud Bigtable.
- It offers a global view of Prometheus time series data that includes data in long-term storage, greatly expanding the usefulness of PromQL for analytical purposes.
- It has multi-tenancy built into its very core. All Prometheus metrics that pass-through Cortex are associated with a specific tenant.

Cortex has a fundamentally service-based design, with its essential functions split up into single-purpose components that can be independently scaled:

- Distributor — Handles time series data written to Cortex by Prometheus instances using Prometheus' remote write API. Incoming data is automatically replicated and sharded and sent to multiple Cortex ingesters in parallel.
- Ingester — Receives time series data from distributor nodes and then writes that data to long-term storage backends, compressing data into Prometheus chunks for efficiency.
- Ruler — Executes rules and generates alerts, sending them to Alertmanager (Cortex installations include Alertmanager).
- Querier — Handles PromQL queries from clients (including Grafana dashboards), abstracting over both ephemeral time series data and samples in long-term storage.

Each of these components can be managed independently, which is key to Cortex's scalability and operations story.

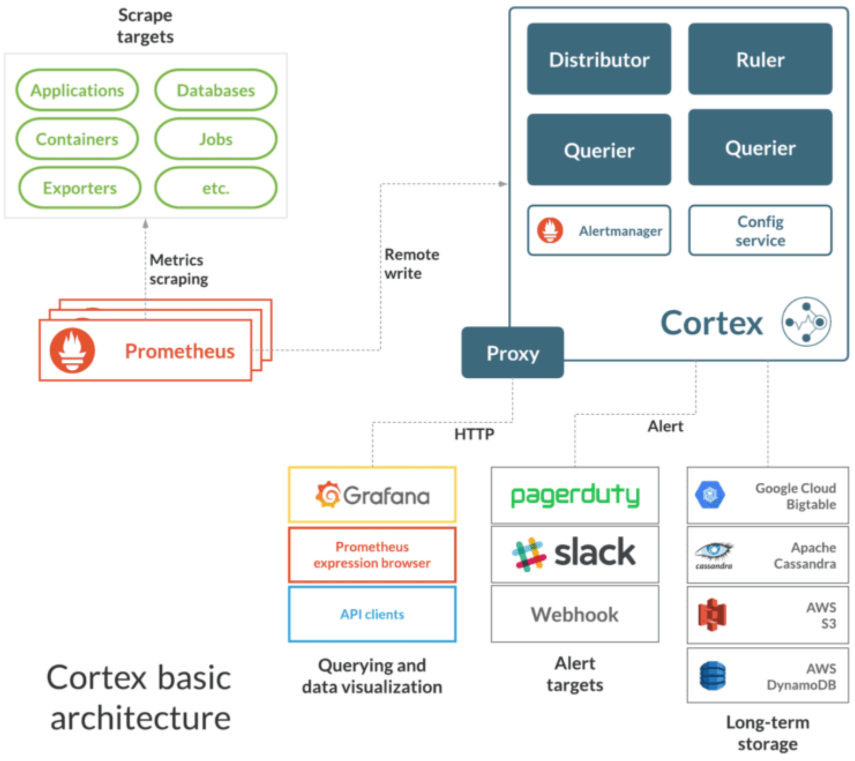

Fig. 3 Cortex Architecture

4.3 Nagios

Nagios is a free and open-source computer-software application that monitors systems, networks, and infrastructure. Nagios offers monitoring and alerting services for servers, switches, applications, and services. It alerts users when things go wrong and alerts them a second time when the problem has been resolved. Additionally, a commercial version, Nagios XI, includes extended features.

4.3.1 Features

- Monitoring of network services (SMTP, POP3, HTTP, NNTP, ICMP, SNMP, FTP, SSH)
- Monitoring of host resources (processor load, disk usage, system logs) on a majority of network operating systems, including Microsoft Windows, using monitoring agents.
- Monitoring of any hardware (like probes for temperature, alarms, etc.) which can send collected data via a network to specifically written plugins
- Monitoring via remotely run scripts via Nagios Remote Plugin Executor
- Remote monitoring supported through SSH or SSL encrypted tunnels.
- A simple plugin design that allows users to easily develop their own service checks depending on needs, by using their tools of choice (shell scripts, C++, Perl, Ruby, Python, PHP, C#, etc.)
- Available data graphing plugins
- Parallelized service checks
- Flat-text formatted configuration files (integrates with many config editors)
- The ability to define network host using 'parent' hosts, allowing the detection of and distinction between hosts that are down or unreachable
- Contact notifications when service or host problems occur and get resolved (via e-mail, pager, SMS, or any user-defined method through plugin system)
- The ability to define event handlers to be run during service or host events for proactive problem resolution
- Automatic log file rotation
- Support for implementing redundant monitoring hosts
- Support for implementing performance data graphing
- Support for database backend (such as NDOUtils)
- Push notifications
- A web-interface for viewing current network status, notifications, problem history, log files, etc.

4.4 Zabbix

Zabbix is an open-source monitoring software tool for diverse IT components, including networks, servers, virtual machines (VMs) and cloud services. Zabbix provides monitoring metrics, among others network utilization, CPU load and disk space consumption.

4.4.1 Features

- ***Metric collection*** **-** Collect metrics from any devices, systems, applications Metric collection methods:
  - Multi-platform Zabbix agent
  - SNMP and IPMI agents
  - Agentless monitoring of user services
  - Custom methods
  - Calculation and aggregation
  - End user web monitoring

- *Problem detection* - Detect problem states within the incoming metric flow automatically. No need to peer at incoming metrics continuously.
  - Highly flexible definition options
  - Separate problem conditions and resolution conditions
  - Multiple severity levels
  - Root cause analysis
  - Anomaly detection
  - Trend prediction
- *Visualization* - The native web interface provides multiple ways of presenting a visual overview of your IT environment:
  - Widget-based dashboards
  - Graphs
  - Network maps
  - Slideshows
  - Drill-down reports

4.4.2 Zabbix server

Zabbix server is the central process of Zabbix software. The server performs the polling and trapping of data, it calculates triggers, sends notifications to users. It is the central component to which Zabbix agents and proxies report data on availability and integrity of systems. The server can itself remotely check networked services (such as web servers and mail servers) using simple service checks.

It can be used for:

- high level monitoring of IT services.
- centralized monitoring of servers and applications.
- monitoring of SNMP-enabled devices.
- performance monitoring (process load, network activity, disk activity, memory usage, OS parameters etc.).
- data visualization.

4.4.3 Zabbix client

The following list of checks is supported by Zabbix agent out of the box.

- Network: Packets/bytes transferred, Errors/dropped packets, Collisions
- CPU: Load average, CPU idle/usage, CPU utilization data per individual process
- Memory: Free/used memory, Swap/pagefile utilization,
- Disk: Space free/used, Read and write I/O,
- Service : Process status, Process memory usage, Service status (ssh, ntp, ldap, smtp, ftp, http, pop, nntp, imap), Windows service status, DNS resolution, TCP connectivity, TCP response time
- File: File size/time, File exists, Checksum, MD5 hash, RegExp search
- Log: Text log, Windows eventlog
- Other: System uptime, System time, Users connected, Performance counter (Windows)

4.5 FogFlow

FogFlow is an IoT edge computing framework to automatically orchestrate dynamic data processing flows over cloud and edges driven by context, including system context on the available system resources from all layers, data context on the registered metadata of all available data entities, and also usage context on the expected QoS defined by users.

FogFlow provides a standard-based and data-centric edge programming model for IoT service providers to realize services easily and fast for various business demands. With its data-driven and optimized service orchestration mechanism, FogFlow helps supervising infrastructure by automatically and efficiently managing thousands of cloud and edge nodes for city-scale IoT services to achieve optimized performance.

In FogFlow, an IoT service is defined as a data processing flow represented by a graph of linked operators. An operator takes input from the IoT devices or from earlier parts of the processing flow. It performs the business logic of the IoT service and passes the intermediate result to the next operator. An operator is realized as a dockerized application and instantiated by FogFlow to run as a task within a dedicated docker container. Tasks are linked with each other during runtime based on the data dependency of their inputs and outputs. IoT services are orchestrated as dynamic data processing flows between producers (e.g., sensors) and consumers (e.g., actuators or applications) to perform necessary data processing.

The unique feature of FogFlow is context-driven, meaning that FogFlow can orchestrate dynamic data processing flows over cloud & edges based on three types of contexts:

- System context: available resources which are changing over time The resources in a cloud-edge environment are geo-distributed in nature and they are dynamically changing over time; As compared to cloud computing, resources in such a cloud-edge environment are more heterogenous and dynamical.
- Data context: the structure and registered metadata of available data, including both raw sensor data and intermediate data based on the standardized and unified data model and communication interface, namely NGSI, FogFlow is able to see the content of all data generated by sensors and data processing tasks in the system, such as data type, attributes, registered metadata, relations, and geo-locations.
- Usage context: high level intents defined by all different types of users (developers, service consumers, data providers) to specify what they want to achieve. In FogFlow, orchestration decisions are made to meet those user-definable objectives during the runtime.

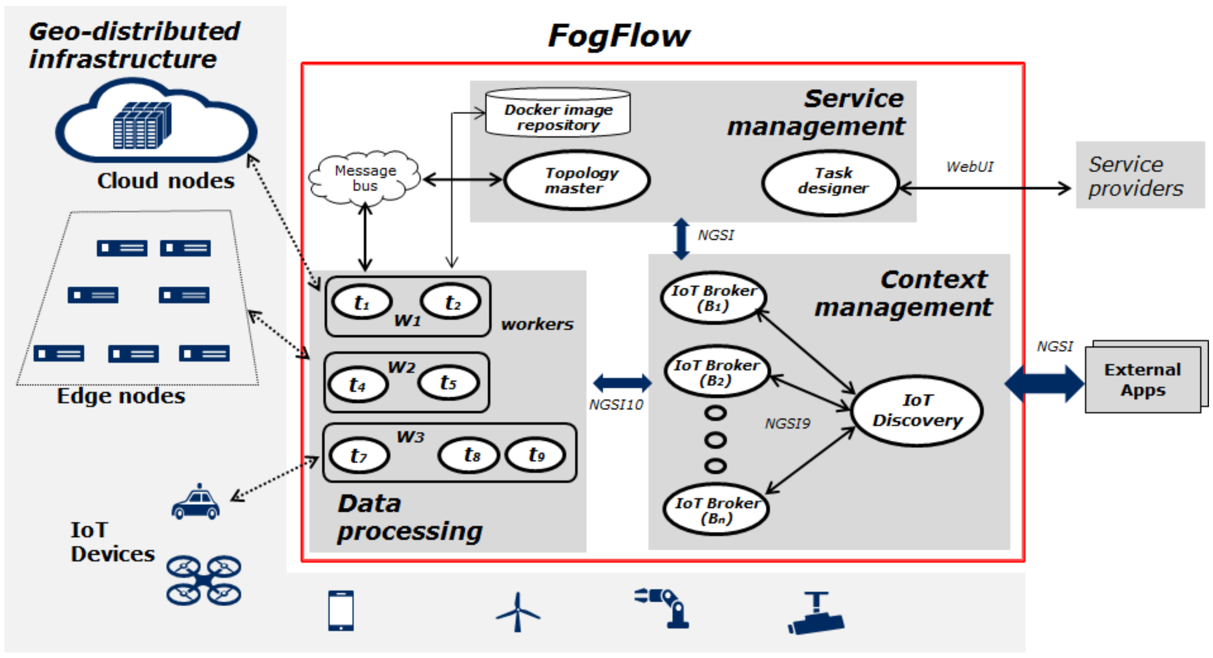

Fig. 4 FogFlow Architecture

Logically, FogFlow consists of the following three layers:

- Service management: convert service requirements into concrete execution plan and then deploy the generated execution plan over cloud and edges
- Context management: manage all context information and make them discoverable and accessible via flexible query and subscribe interfaces
- Data processing: launch data processing tasks and establish data flows between tasks via the pub/sub interfaces provided by the context management layer

There are three centralized service components to be deployed in the cloud:

- Task Designer: provide the web-based interface for service providers to specify, register, and manage their tasks and service topologies.
- Topology Master: figure out when and which task should be instantiated, dynamically configure them, and decide where to deploy them over cloud and edges.
- IoT Discovery: manage all registered context availability information, including its ID, entity type, attribute list, and metadata; allow other components to query and subscribe their interested context availability information via NGSI9

And several distributed components to be deployed both in the cloud and at edges:

- Worker: according to the assignment from the topology master, each worker will launch its scheduled task instances in docker containers on its local host; configure their inputs and outputs and manage all task instances locally based on task priority
- IoT Broker: each broker manages a part of context entities published by nearby IoT devices and also provides a single view of all context entities for IoT devices to query and subscribe the entities they need.

Besides, the following external service components are needed to use FogFlow:

- Dock Registry: managed all docker images provided by developers.
- RabbitMQ: The internal communication between topology master and the workers
- PostgreSQL: the backend database to save the registered context availability information

4.6 Comparison

Service	Monitoring	Server	Client
Prometheus	Physical components, virtualized components, and applications (microservices)	X	Exporters (HAProxy, StatsD, Graphite, etc.).
Cortex	Multitenant solution for Prometheus. Provides scalability and long-term storage.	X	Same as Prometheus
Nagios		X	Nagios Remote Plugin Executor (NRPE)
Zabbix		X	X
FogFlow	Monitor metrics of FogFlow cloud as well as edge nodes	X	Provides APIs for edge devices. Workers run on edge devices.

5 VISUALIZATION

In the era of big data, visual dashboards have become an important tool for business decisions. A series of means of visually presenting complex and abstract data in a more understandable form is called data visualization. Data visualization is designed to enable people to quickly understand the meaning behind the data. The dashboard for large screen can be used for information display, data analysis, monitoring and early warning.

The ability to remotely manage and monitor automation systems can help to handle a variety of tasks and services including incident management, system backup and recovery, data analysis and retrieval, software updates and patches, real-time monitoring of software and systems, and online automation and system programming.

5.1 Grafana

Grafana is an open source tool to visualize time series data. It is written in Go Language and Node.js and includes a strong Application Programming Interface (API).

5.1.1 Features

- Visualize: Fast and flexible client-side graphs with a multitude of options to visualize metrics and logs.
- Dynamic Dashboards: Create dynamic & reusable dashboards with template variables that appear as dropdowns at the top of the dashboard.
- Explore Metrics: Explore data through ad-hoc queries and dynamic drilldown. Split view and compare different time ranges, queries, and data sources side by side.
- Explore Logs: Experience the magic of switching from metrics to logs with preserved label filters. Quickly search through all logs or stream them live.
- Alerting: Visually define alert rules for most important metrics. Grafana will continuously evaluate and send notifications to systems like Slack, PagerDuty, VictorOps, OpsGenie.
- Supported Data Sources: Built-in support or data source plugins for: AWS CloudWatch, Azure Monitor, Elasticsearch, Google Stackdriver, Graphite, InfluxDB, Loki, Microsoft SQL Server (MSSQL), Mixed, MySQL, OpenTSDB, PostgreSQL, Prometheus, Testdata.
- Mixed Data Sources: Mix different data sources in the same graph. It is possible to specify a data source on a per-query basis.

5.2 Kibana

Kibana is an open source browser-based analytics and search dashboard application for Elasticsearch. Kibana is a visual interface tool that allows to explore, visualize, and build a dashboard over the log data massed in Elasticsearch Clusters. The core feature of Kibana is data querying & analysis. In addition, Kibana's visualization features allow to visualize data in alternate ways using heat maps, line graphs, histograms, pie charts, and geospatial support.

5.2.1 Features

- Data Discovery - Interactive data exploration by enabling access of each field in conjunction with default time. You can set the time filter, submit search queries, view document data, and filter the search results.
- Visualization - With the broad variety of visualization styles, Kibana allows to create a visualization of data in the Elasticsearch indices. You can combine pie charts, data tables, line charts, single metric visualization, time series, Geo maps, and markdown visualization into dashboards.
- Custom Dashboards - The Kibana dashboard is extremely dynamic and adaptable. For example, it is possible to filter the data on the fly and open the dashboard in full-page format. The customizable dashboard feature allows to resize, arrange, and modify the dashboard content and save it so it can be shared.
- Timeseries visualization - Analyse and visualize the time series data using simple expression language and it is designed to bring together the completely independent data sources within a single interface. It offers a way to define queries, visualization, and transformation in one place.
- Machine Learning - This feature automatically models the behaviour of the Elasticsearch data, including periodicity & trends and spot issues faster thereby reduce false positives.
- Management - The management page of Kibana is used for performing Kibana's runtime configuration, which includes three vital actions:

  Index pattern – supports for initial setup & ongoing configuration of index names

  Saved objects – hosts the saved visualization, dashboards, and searches

  Advanced settings – includes setting to tweak the Kibana's behaviour

  Therefore, whether it is setting up security controls, adding data sources or handling pipelines, Kibana offers a single interface solution.

5.3 Chronograf

Chronograf is an open source web application for visualizing and monitoring InfluxDB timeseries data and to create alerting and automation rules. Chronograf allows to quickly see the data stored in InfluxDB and it includes templates and libraries to allow to rapidly build dashboards with real-time data visualizations.

5.3.1 Features

- Dashboards - Chronograf offers a complete dashboarding solution for visualizing data. Over 20 pre-canned dashboards are available to clone and create customized dashboards or dashboards can be built from scratch.
- Administration - Chronograf is the administrative tool for InfluxData deployments — the open source instances of InfluxData as well as InfluxDB Enterprise and InfluxDB Cloud instances. Chronograf also provides a number of security options like user authentication services (GitHub, Google, Auth0, Heroku, and Okta) and Role-Based Access Controls (RBAC) to help administrators provision the correct resources (dashboards, InfluxDB and Kapacitor connections) to ensure security and compliance postures.

- **Alerting** - Chronograf is also the user interface for Kapacitor — a native data processing engine that can process both stream and batch data from InfluxDB. It allows to create alerts with a simple step-by-step UI and see the alert history in Chronograf. Chronograf also provides a TICKscript editor the ability to manage the alert history and TICKscripts. Chronograf offers a log viewer tools to view, search, filter, and analyse log data.

5.4 Comparison

Service	Data Sources	Queries	Visualization	Custom Dashboards	Alerts	Machine Learning
Grafana	AWS CloudWatch, Azure Monitor, Elasticsearch, Google Stackdriver, Graphite, InfluxDB, Loki, Microsoft SQL Server (MSSQL), Mixed, MySQL, OpenTSDB, PostgreSQL, Prometheus, Testdata	Query language of the data source.	Tables and Charts	X	X	
Kibana	Elasticsearch	Kibana Query Language (KQL)	Tables and Charts	X	X	X
Chronograf	InfluxDB	SQL like syntax	Tables and Charts	X	X	

6 DATABASES

6.1 ObjectBox

ObjectBox is an object-oriented database, designed to manage various types of data quickly and efficiently, enabling complex processing on the edge. As an ACID (Atomicity, Consistency, Isolation, Durability) compliant data product, ObjectBox ensures data reliability. ObjectBox is an embedded high-performance database backend for implementing ObjectBox EdgeX v1.1 since it offers data reliability and high-speed database operations and it can be used on 32-bit ARM devices (EdgeX Foundry only supports 64-bit).

Objectbox EdgeX combines ObjectBox' speed and ease of use with EdgeX Foundry IoT Edge Platform. ObjectBox' small footprint of less than 1MB makes the database uniquely suited for small devices. There are several native language APIs available for ObjectBox, including Go, C/C++, Java, Python, Kotlin, and Swift, and runs on multiple platforms: Linux, Windows, Mac/iOS, and Android.

6.1.1 Features

- **Objects:** It is highly optimized for persisting objects. Due to the EdgeX architecture and Go sources, costly transformations can be avoided as it puts Go's objects (structs) in the centre of its interface.
- **Embedded database:** Redis and MongoDB are client/server databases running in separate processes. ObjectBox, however, is running in the same process as EdgeX itself, I.e. each EdgeX microservice. This has definite efficiency advantages, but it also comes with some restrictions: Whereas you can put Redis/MongoDB in separate Dockers or machines, this option is not available for ObjectBox yet.
- **Transaction merging:** ObjectBox can execute individual write operations in a common database transaction. Thus, the costly transactions for several write operations can be reduced.

Combining the speed and size advantages of ObjectBox on the EdgeX platform, it is possible to analyse more data locally on the machine, enabling new use cases. EdgeX Foundry comes with a choice of two database engines: MongoDB and Redis. ObjectBox EdgeX brings an alternative to Redis and MongoDB due to its properties: (1) it is an embedded database, (2) it is optimized for high speed and (3) it is ease of use while also delivering data reliability.

6.2 Redis

Redis is an in-memory, persistent data structure store used as a database, cache, and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperlog logs, geospatial indexes with radius queries and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions, and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

Other features include:

- Transactions
- Pub/Sub
- Lua scripting
- Keys with a limited time-to-live

- LRU eviction of keys
- Automatic failover

Redis is written in ANSI C and works in most POSIX systems like Linux, *BSD, OS X without external dependencies. Linux and OS X are the two operating systems where Redis is developed and tested the most. Redis may work in Solaris-derived systems like SmartOS, but the support is best effort. There is no official support for Windows builds. Redis is being supported in most of the languages, like JavaScript, Java, Go, C, C++, C#, Python, Objective-C, or PHP.

6.2.1 Persistence

Depending on the use case, the persistence can be performed either by dumping the dataset to disk occasionally, or by appending each command to a log. Redis provides a different range of persistence options:

- RDB persistence: It performs point-in-time snapshots of your dataset at specified intervals.
- AOF persistence: It logs every write operation received by the server, that will be played again at server start-up, reconstructing the original dataset. Commands are logged using the same format as the Redis protocol itself, in an append-only fashion. Redis is able to rewrite the log in the background when it gets too big.
- RDB + AOF: It is possible to combine both AOF and RDB in the same instance. Notice that, in this case, when Redis restarts the AOF file will be used to reconstruct the original dataset since it is guaranteed to be the most complete.

RDB is NOT good if you need to minimize the chance of data loss in case Redis stops working (for example after a power outage). You can configure different save points where an RDB is produced (for instance after at least five minutes and 100 writes against the data set, but you can have multiple save points). However, you will usually create an RDB snapshot every five minutes or more, so in case of Redis stopping working without a correct shutdown for any reason you should be prepared to lose the latest minutes of data. Redis snapshotting does NOT provide good durability guarantees if up to a few minutes of data loss is not acceptable in case of incidents, so it is usage is limited to applications and environments where losing recent data is not very important.

AOF files are usually bigger than the equivalent RDB files for the same dataset. It can be slower than RDB depending on the exact fsync policy. In general, with fsync set to every second performance is still very high, and with fsync disabled it should be exactly as fast as RDB even under high load. Still RDB is able to provide more guarantees about the maximum latency even in the case of a huge write load.

6.3 SQLite

SQLite is a C-language library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is an embedded relational database engine which source code is in the public domain. Unlike most other SQL databases, SQLite is serverless, i.e. it does not have a separate server process. Moreover, SQLite reads and writes directly to ordinary disk files and a complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform. Thus, a database can be freely copied between 32-bit and 64-bit systems or between big-endian and little-endian architectures.

SQLite engine is not a standalone process like other databases, you can link it statically or dynamically as per your requirement with your application. SQLite is a compact library. With all features enabled, the library size can be less than 600KiB, depending on the target platform and compiler optimization settings. There is a trade-off between memory usage and speed, running faster if more memory is given. Nevertheless, performance is usually quite good even in low-memory environments.

SQLite uses dynamic types for tables. It means you can store any value in any column, regardless of the data type. SQLite allows a single database connection to access multiple database files simultaneously. This brings many nice features like joining tables in different databases or copying data between databases in a single command. SQLite can create in-memory databases that are very fast to work with.

6.3.1 Main features

- Transactions are ACID.
- Zero-configuration - no setup or administration needed.
- Full-featured SQL implementation with advanced capabilities: partial indexes, indexes on expressions, JSON, common table expressions, and window functions.
- A complete database is stored in a single cross-platform disk file.
- Supports terabyte-sized databases and gigabyte-sized strings and blobs.
- Small code footprint: less than 600KiB fully configured or much less with optional features omitted.
- Simple, easy to use API.
- In some cases, SQLite can be faster than direct filesystem I/O
- Written in ANSI-C.
- Available as a single ANSI-C source-code file that is easy to compile and add into a larger project.
- Self-contained: no external dependencies.
- Cross-platform: Android, *BSD, iOS, Linux, Mac, Solaris, VxWorks, and Windows (Win32, WinCE, WinRT) are supported out of the box. Easy to port to other systems.
- Comes with a standalone command-line interface (CLI) client that can be used to administer SQLite databases.

6.3.2 Typical application use-cases

- Embedded devices and IoT.
- Application file format.
- Websites.
- Data analysis.
- Cache for enterprise data.
- Server-side database.
- Data transfer format.
- File archive and/or data container.