HIGHER PRIVATE SCHOOL OF TECHNOLOGY AND ENGINEERING

Dept. of Telecommunication Engineering

Design and Deploying AIoT Workloads on Cloud Native Edge Infrastructure

---

Realized by Selim BENFRADJ

Academic supervisor Safa SAOUDI

Professional supervisor Walid HADJ TAIEB

June, 2022

CERTIFICATE

This is to certify that the project progress report end **" THESIS "** which is being submitted by **JOHN DOE** (**ROLL. NO:XXXXETMXXX**) in partial fulfilment of the requirement for the degree of Master of Engineering in Electronics and Telecommunication Engineering with specialization XXXXXXXXXX & XXXXXX, has been prepared by him, under my supervision and guidance.

.....
Supervisor Name
Dept. of Electronics and
Telecommunication
IIEST, Shibpur

Countersigned by:

.....	.....
HOD NAME	Dr. Sudip Kr. Roy
Head of Dept.	Dean of Academic Affairs
Dept. of E.T.C.,	Dept. of Civil Engineering
IIEST, Shibpur.	IIEST, Shibpur.

ABSTRACT

Recent studies have suggested several ways to optimize the stability of the modernized infrastructure to help in making measures to prevent critical incidents. The goal of this project is to simulate Machine learning model using combined monitored performance data to predict critical system states. Meanwhile, we adopt a native cloud native design approach in order to implement the entire system in the post-validation production environment using Cloud native computing foundation technologies .

Contents

List of Figures

List of Tables

Acronyms

AIoT Artificial Intelligence of Things

ML Machine Learning

CNCF Cloud Native Computing Foundation

AIops Artificial Intelligence for IT Operations

ELK Elasticsearch, Logstash et Kibana

CSV comma separated values

SVC Services

NS Namespaces

VM Virtual machine

YAML Yet Another Markup Language

YML Yet Another Markup Language

K8S Kubernetes cluster

K3S Lightweight Kubernetes

Telco-cloud Telecom Core cloud

NFV Network function virtualisation

DC Date center

DCN Date center Networking

API Application program interface

SDN Software defined network

5G 5th generation mobile network

CPU Central processing unit

KPI Key performance indicator

General Introduction

Understanding the state of your infrastructure and systems is necessary to guarantee that your services are stable and reliable. Information regarding the health and performance of your deployments not only assists your team in addressing issues, but it also provides them the confidence to make improvements. A strong monitoring system that gathers measurements, visualizes data, and informs engineers when things appear to be faulty is one of the greatest ways of getting this insight.

Metrics, monitoring, and alerting are all interrelated concepts that together form the basis of a monitoring system. They have the ability to provide visibility into the health of your systems, help you understand trends in usage or behavior, and to understand the impact of changes you make. If the metrics fall outside of your expected ranges, these systems can send notifications to prompt an operator to take a look, and can then assist in surfacing information to help identify the possible causes. While metrics represent the data in your system, monitoring is the process of collecting, aggregating, and analyzing those values to improve awareness of your Components' characteristics and behavior. The data from various parts of your environment are collected into a monitoring system that is responsible for storage, aggregation, visualization, and initiating automated responses when the values meet specific requirements.

Monitoring systems responsibility is to accept and store incoming and historical data, values representing the current point in time are useful, it is more helpful to view those numbers in relation to past values to provide context around changes and trends. This means that a monitoring system should be capable of managing data over periods of time, which may involve sampling or aggregating older data. While metrics can be displayed and understood as individual values or tables, humans are much better at recognizing trends

and Understanding how components fit together when information is organized in a visually meaningful way. Monitoring systems typically represent the component they measure through configurable graphics and dashboards. This makes an opportunity to understand the interplay of complex variables or changes. Within a system by glancing at a display. Another additional function that monitoring systems provide is organizing and correlating data from various [6] Inputs. For the metrics to be useful, administrators need to be able to recognize patterns between different resources and across groups of servers. However, in modern DCNs, failures of infrastructure devices are the norm rather than the exception, and much research efforts have focused on dealing with failures after they happen.

In this project, we take a different approach by predicting failures, thus the operators can intervene and fix the potential failures before they happen. We aim to determine during run-time whether a device failure will happen in the near future. The prediction is based on the measurements of the current devices system status and historical hardware failure cases that have been carefully labelled by network operators using the old data collection offered by monitoring system .

Chapter 1

General context

In this chapter, we present the hosting company and its hierarchy. Then we move to digital transformation trend in telecommunication industry focusing on the modernization of the Telco-Cloud infrastructure, hence the problem that this approach will solve.

1.1 Hosting company

Ooredoo is a leading international communications company delivering mobile, fixed, broadband internet and corporate managed services tailored to the needs of consumers and businesses across markets in the Middle East, North Africa and Southeast Asia. As a community-focused company, Ooredoo is guided by its vision of enriching people's lives and its belief that it can stimulate human growth by leveraging communications to help people achieve their full potential. Ooredoo has a presence in markets such as Tunisia, Qatar, Kuwait, Oman, Algeria, Iraq, Palestine, the Maldives, Myanmar and Indonesia. The company was named "Best Mobile Operator of the Year" at the World Communication Awards 2013. [4]

Figure 1.1: Ooredoo across the world

1.1.1 Ooredoo Tunisia

Ooredoo Tunisia is a leading global operator offering various services dedicated to both individuals (Mobile, Fixed, data ..) and companies through hosting solutions, IoT, cloud and very high speed. As a responsible company rooted in the community. Ooredoo Tunisia is guided by its vision to enrich the digital lives of its customers and its belief that it can stimulate human growth by leveraging new technologies to help the community realize its full potential.

Figure 1.2: Ooredoo logo

1.2 Modernization challenge

During transformation, the telco's infrastructure evolve from the expensive and difficult-to-manage set of discrete network elements, to a virtualized communications and cloud infrastructure, which can be managed in a highly autonomous fashion, at extremely low cost.

Network functions virtualization and software-defined networking (NFV/SDN) are making this first journey a reality, but there is a ways to go in making it manageable, and the cost, complexity, and disruption are huge. At the same time, 5G draws ever nearer, its next-generation network capabilities requiring NFV/SDN.

By the way Ooredoo Tunisia invest in Core Modernization since the beginning of the transformation waves .However with all those new Core features there are a new challenge for maintaining infrastructure health .with this huge responsibility our team need an innovate tool which support our private cloud with different approach .

1.3 Case study

Before introducing the proposed solution we need to explain briefly where the issues raise first time . Introducing private Cloud uses case in our Core infrastructure which is based on Openstack Triplo.

Figure 1.3: Openstack logo

1.3.1 Openstack based cloud

OpenStack is a cloud operating system that controls large pools of Compute, Storage,Controller and networking resources throughout a datacenter, all managed and provisioned through APIs with common authentication

mechanisms. Beyond standard infrastructure-as-a-service functionality, additional components provide orchestration, fault management and service management amongst other services to ensure high availability of user applications.

1.3.2 Openstack Triplo

TripleO is the friendly name for "OpenStack on OpenStack". It is an official OpenStack project with the goal of allowing you to deploy and manage a production cloud onto bare metal hardware using a subset of existing OpenStack components.[9]

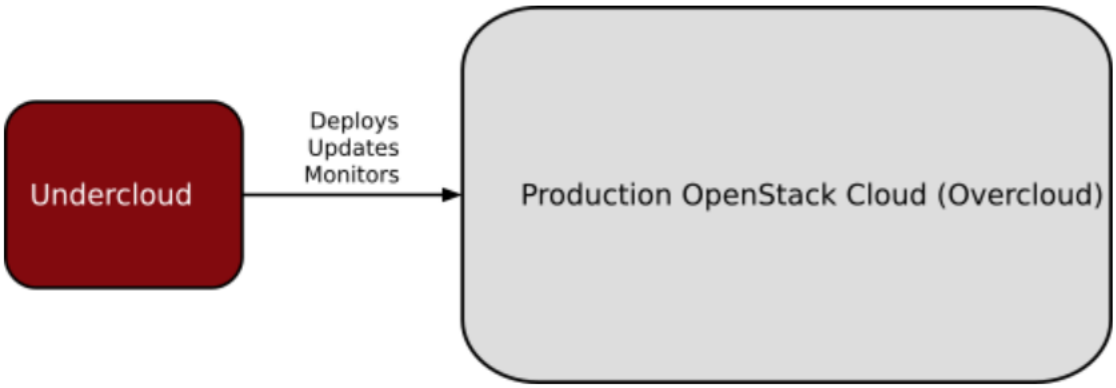


Figure 1.4: Undercloud vs Overcloud

TripleO leverages several existing core components of OpenStack including Nova, Ironic, Neutron, Heat, Glance and Ceilometer to deploy OpenStack on baremetal hardware. Nova and Ironic are used in the Undercloud to manage baremetal instances that comprise the infrastructure for the overcloud. Neutron is utilized to provide a networking environment in which to deploy the overcloud, machine images are stored in Glance, and Ceilometer collects metrics about overcloud.

The following diagram 1.5 illustrates a physical view of how the Undercloud may be hosted on a single physical server and the Overcloud distributed across many physical servers.

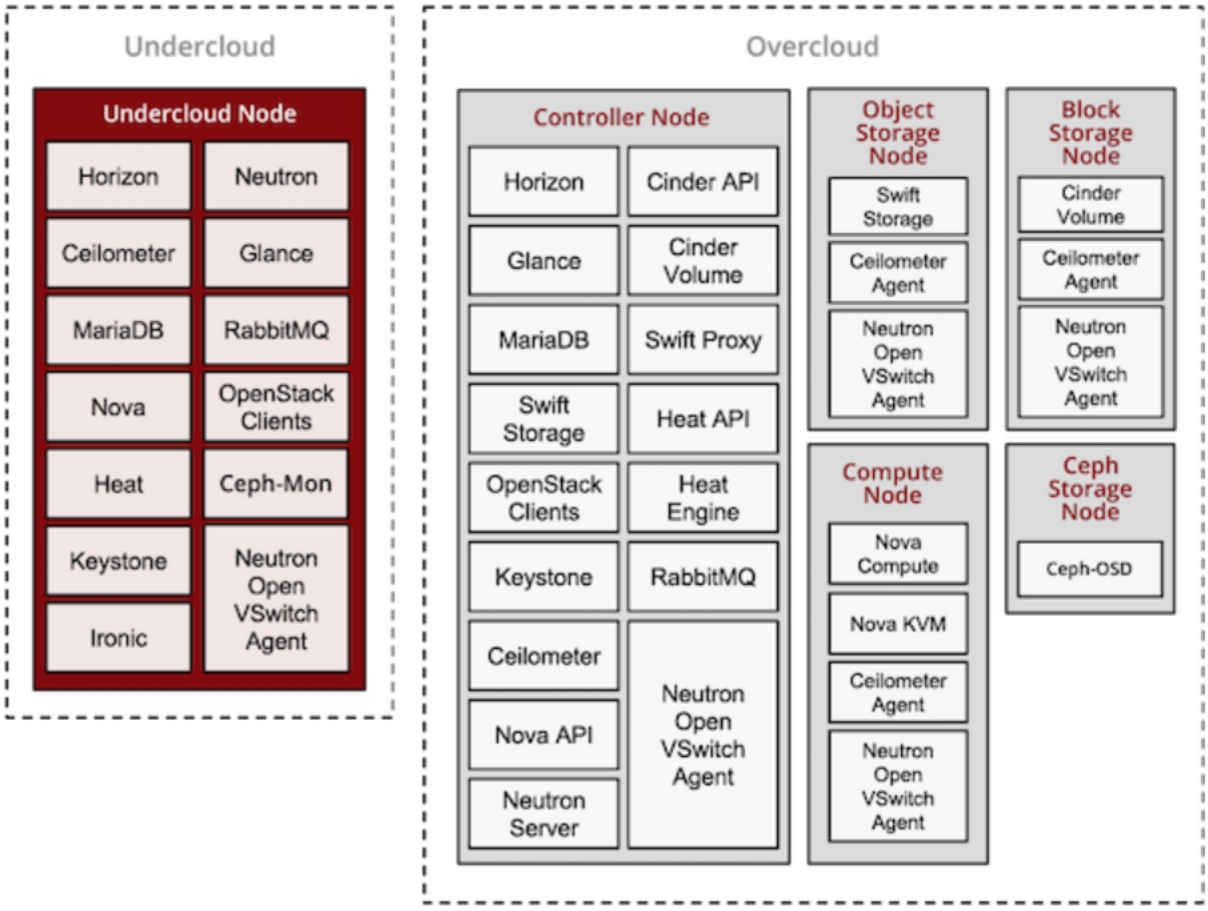


Figure 1.5: Undercloud single server vs Overcloud multi servers

1.4 Challenge

The Overcloud computes host all Virtual machines stack which compose the Network function virtualized services. Regarding this environment criteria ,its clear that supporting all cloud components is an extremely sensitive mission by maintaining system health to minimize production downtime in case

of failure,one of the critical daily routine workload for our team is to perform virtual machines hot and cold migration between computes in order to avoid application impact as much as possible.

Hence the necessity for a new tool to identify system issues even before they occur. using machine learning features and cloud Native approach the proposed solution will combine a collection of metrics and logs to identify failure in ordered and human readable way for both real and forcasted data.

1.5 Project's methodology

Basically every project must follow a clear roadmap using a work methodology to optimize the dedicated development period . In this section, we're going to expose the process of choosing a work methodology to adopt all through our project's life cycle

1.5.1 Agile methods

Agile methods are procedures of software design practice seen more practical than traditional methods. These methods ensure a high interaction with the client by involving him in the development of the product process. Hence, it increases the customer's satisfaction. In fact, agile methods are a set of "best practice" which purpose is reducing the software development cycle by developing a minimal version, then evolving the product through an iterative process based on the client interaction and interests as well as tests taking place all along the development cycle.
The goal of the software development is to produce software of quality. Several criteria try to define the quality of software. So we are obliged to follow a methodology of development to ensure a better quality for our software. Indeed, the processes of development govern the production's activities of the software according to two aspects:

- The static aspect which represents the process in terms of tasks to be carried out.

- The dynamic aspect which represents the temporal dimension of the process.

We will start by giving a comparison between the various processes of development to choose the best adapted to our project. Considering the big number of methodologies, we will limit ourselves to a comparison between the XP, RUP, Scrum and FDD which are widely used in the companies.

1.5.2 The chosen methodology: Scrum

In order to ensure the full potentials of the project, the choice is landed on the Scrum process of agile methods.
The term Scrum (which means "mixed" in rugby) is closer to a management human resources rather than an actual method development. It is a simple framework used to organize teams and get work done more productively with higher quality.

1.6 conclusion

In this chapter we introduce the project global scoop by diving through real world use case from a telco-cloud infrastructure view and what is the added value behind this project and last but not least we put a road-map methodology to be followed next.

Chapter 2

Proof of concept and Theoretical study

In this chapter we discuss the solution expected requirements to proceed after with methodology and machine learning theoretical study .

2.1 Functional requirements

in this section we describe main desired system functionality then turn it into rules and constraints to fit the global goal

1. Data scraping:
   One of our system main functionality is to scrap metrics in a comprehensible manner whether is the type of metrics itself

   - A gauge metric : in which the value measures a specific instant in time. For example, metrics measuring CPU utilization are gauge metrics; each point records the CPU utilization at the time of measurement.
   - A delta metric, in which the value measures the change since it was last recorded. For example, metrics measuring request counts are delta metrics; each value records how many requests were received since the last data point was recorded.
   - A cumulative metric, in which the value constantly increases over time. For example, a metric for "sent bytes" might be cumulative;

each value records the total number of bytes sent by a service at that time.

furthermore scraping mechanism must support textual data in case of scrapping log event from multiple file without indexing the whole columns in log to optimize resources consumption which are limited

2. Data collection:
   A data collection process must offer persistence mechanism even in case of disaster where the collector tool is unreachable or down hence the need for a persistence storage

3. Data visualisation:
   Visualizing data is the purpose behind all monitoring previous process that's why dealing with it very important in term of data-source supporting and the way they are shown in comprehensible manner .

4. Data Prediction:
   Whether is the system distribution across infrastructure it must contain a functionality where the data are predicted using machine learning diverse features.this function should be given a high priority in term of resources and development time.

5. Cloud Native Approach:
   According to the Cloud Native Computing Foundation (CNCF): "Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil." [7] such requirement exactly match our uses case approach and turn our workload from monolith infrastructure culture into a modernized way of thinking by giving more importance to availability and scalability on order to create give a stable service.

2.2 Non-functional requirements

Non-functional requirements become the main factor in architectural decision making regarding the constraint that the system should respect because they describe the manner of how system act as a result of that.

1. Realtime information :
   When monitoring cloud components at various levels, the monitoring system should provide real-time data not just for functional KPIs but also for application KPIs.

2. Prediction quality :
   In order to provide future information for the useful metrics, the prediction subsystem should provide time series forecasting.

3. Centralized visualisation mechanism :
   To make troubleshooting easier, the graphical dashboard for both metrics and log data should be centralized in a single instance.

2.3 Scrum Planification

Sprint planning is an event in scrum that kicks off the sprint. The purpose of sprint planning is to define what can be delivered in the sprint and how that work will be achieved. Sprint planning is done in collaboration with the whole scrum team [17]

2.3.1 Team Roles

- Scrum master : Supervisors
- Product owner : Core infrastructure Manager
- Development Team : Selim benfradj

2.3.2 Product backlog

giving much more work for the product than the features to provides bugs fixes, refactoring, process improvements. A product backlog organized around these different types of work can ensure we lift our focus off the shiny feature

and place it on other types of work necessary to deliver value to our end user experience 2.1

To do ID	Context	Action
F1	Ceremony of brain-storming	Documentation
F2	VCS : github	init project
F3	Design negotiation	Initial HLD release
F4	Design negotiation	Refactor HLD
F5	Design negotiation	Resource requirement
F6	Machine learning	ML practice/documentation
F7	Machine learning	ML advanced examples
F8	Infrastructure Cluster	orchestrators choice
F9	Infrastructure Dimen-sioning	LLD fixing
F10	Infrastructure	Infrastructure Provisioning
F11	Infrastructure	configuration
F12	Infrastructure	refactoring cluster services
F13	ML :Data-set	API integration
F14	ML: Model workaround	fixing library
F15	ML: Data-set	Transforming data (date)
F16	ML: Model optimiza-tion	checking seasonality
F17	ML: Model optimiza-tion	filling missing data
F18	ML: Model optimiza-tion	accuracy
F19	Additional functions	fixing persistance volume
F20	Additional functions	Model csv output

Table 2.1: Product Backlog

2.3.3 Sprints planification

The table below 2.2 summarizes how each collection of tasks belong to a sprint.

—Sprint—	
to do ID /task	sprint number
F1;F2;F6;F7	sprint-0
F3;F4,F5,F9	sprint-1
F8,F10,F11,F12	sprint-2
F13,F18,F20	sprint-3
F19,F20	sprint-4

Table 2.2: sprint planification

2.4 Theoretical study

In this section we theoretically discuss machine learning types,modes and architectures. Then, we conclude with ML-Ops life cycle.

2.4.1 Machine learning types

basically we need to identify which type of machine learning we focus on by adjusting it to our uses case hence the used data processing workflow . There are four types of machine learning:

- Supervised learning

also called inductive learning Training data includes desired outputs. This is spam this is not, learning is supervised.

- Unsupervised learning

Training data does not include desired outputs. Example is clustering. It is hard to tell what is good learning and what is not.

- Semi-supervised learning

Training data includes a few desired outputs.

- Reinforcement learning

Rewards from a sequence of actions. AI types like it, it is the most ambitious type of learning

[0]Artificial Intelligence

2.4.2 Batsh learning vs online learning

Batch learning represents the training of machine learning models in a batch manner. In other words, batch learning represents the training of the models at regular intervals such as weekly, bi-weekly, monthly, quarterly, etc. The data gets accumulated over a period of time. The models then get trained with the accumulated data from time to time at periodic intervals. Batch learning is also called offline learning. The models trained using batch or offline learning are moved into production only at regular intervals based on the performance of models trained with new data.

Building offline models or models trained in a batch manner requires training the models with the entire training data set. Improving the model performance would require re-training all over again with the entire training data set. These models are static in nature which means that once they get trained, their performance will not improve until a new model gets re-trained. Offline models or models trained using batch learning are deployed in the production environment by replacing the old model with the newly trained model.[11]

while in online learning, the training happens in an incremental manner by continuously feeding data as it arrives or in a small group. Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives.

Online learning is great for machine learning systems that receive data as a continuous flow (e.g., stock prices) and need to adapt to change rapidly or autonomously. It is also a good option if you have limited computing resources: once an online learning system has learned about new data instances, it does not need them anymore, so you can discard them (unless you want to be able to roll back to a previous state and "replay" the data) or move the data to another form of storage (warm or cold storage) if you are using the data lake. This can save a huge amount of space and cost. The diagram given below represents online learning as illustrated in 2.1 it depend in our uses case on many factors where choosing which mode is better since every mode has it's own pros and cons ,regarding the resource limitation (GPU,CPU,...) .

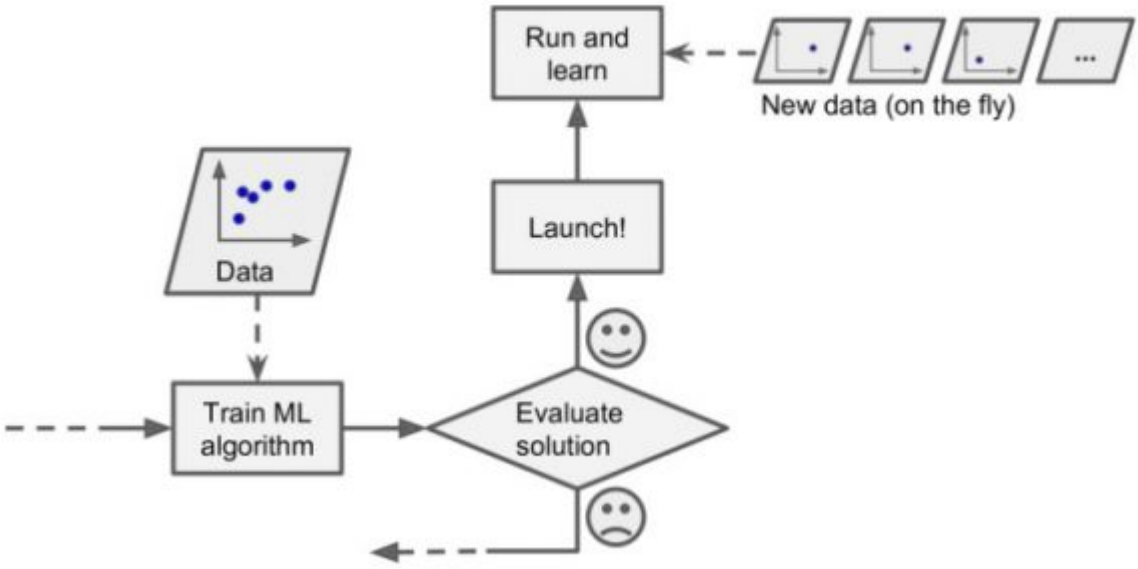


Figure 2.1: Online learning system

2.4.3 Machine learning architecture

In documentation phase we gather a ton of information in relation with machine learning in sysops field and we found that many researches has identify which architecture match there requirement , while ML mode discuss the way the data will be feeded the architecture discuss the whole data processing design :

- Lambda architecture :

Lambda architecture is a data-processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream-processing methods. This approach to architecture attempts to balance latency, throughput, and fault-tolerance by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data. The two view outputs may be joined before presentation. The rise of lambda architecture is correlated with the growth of big data, real-time analytics, and the drive to mitigate the latency of map-reduce.

Lambda architecture depends on a data model with an append-only, immutable data source that serves as a system of record.It is intended for ingesting and processing timestamped events that are appended to existing events rather than overwriting them. State is determined from the natural time-based ordering of the data. Lambda architecture describes a system consisting of three layers: batch processing, speed or real-time processing, and a serving layer for responding to queries. The processing layers ingest from an immutable master copy of the entire data set like mentioned in 2.2

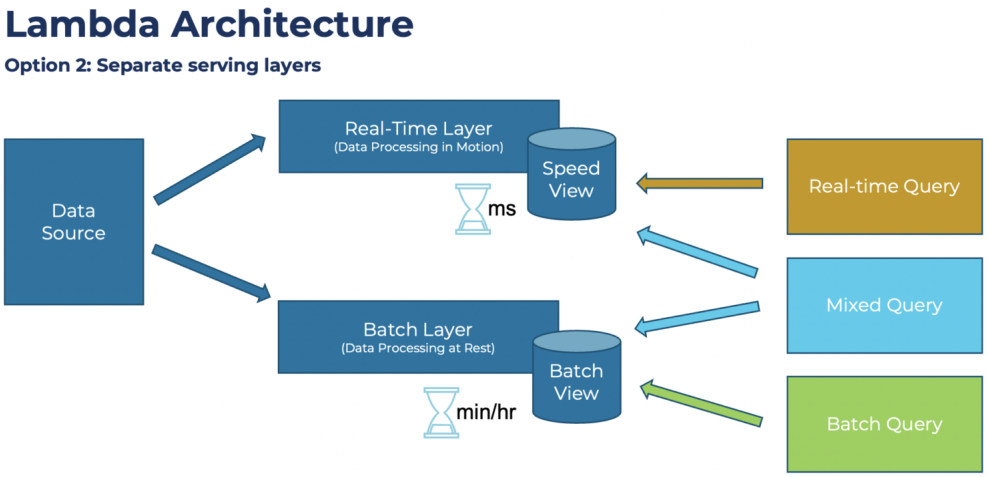


Figure 2.2: Online learning system

- Kappa architecture:

The Kappa architecture is a software architecture that is event-based and able to handle all data at all scale in real-time for transactional and analytical workloads. The central premise behind the Kappa architecture is that you can perform both real-time and batch processing with a single technology stack. The heart of the infrastructure is streaming architecture. First, the event streaming platform log stores incoming data. From there, a stream processing engine processes the data continuously in real-time or ingests the data into any other analytics database or business application via any communication paradigm and speed, including real-time, near real-time, batch, request-response. Unlike the Lambda Architecture, in this approach, you only do re-processing when your processing code changes, and you need to recompute your results. And, of course, the job doing the re-computation is

just an improved version of the same code, running on the same framework and taking the same input data as visualized in 2.3

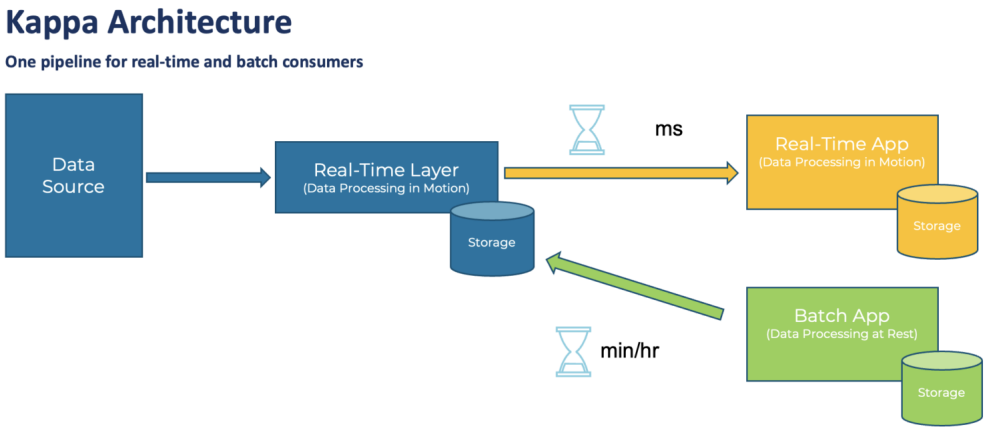


Figure 2.3: Online learning system

2.4.4 MlOps

The machine learning life cycle may be described as a multi-component flow, in which each component affects the next ones. The first component

- Prepare the dataset:

this step includes the data collection phase, transforming and cleaning the raw data. In the second component of the machine learning life cycle.

- Experiments are conducted

the data-set is explored and features are extracted for further investigation.

- Train component

a model is selected and the training process is started to fine-tune its parameters, during this process the model is monitored and evaluated at the end. Once a model satisfies the requirements set precision/accuracy is selected, in the last component.

- Deploy

[0]Machine learning operations

the chosen model is deployed for inference and monitored.

the previous steps are organized in the following figure 2.4

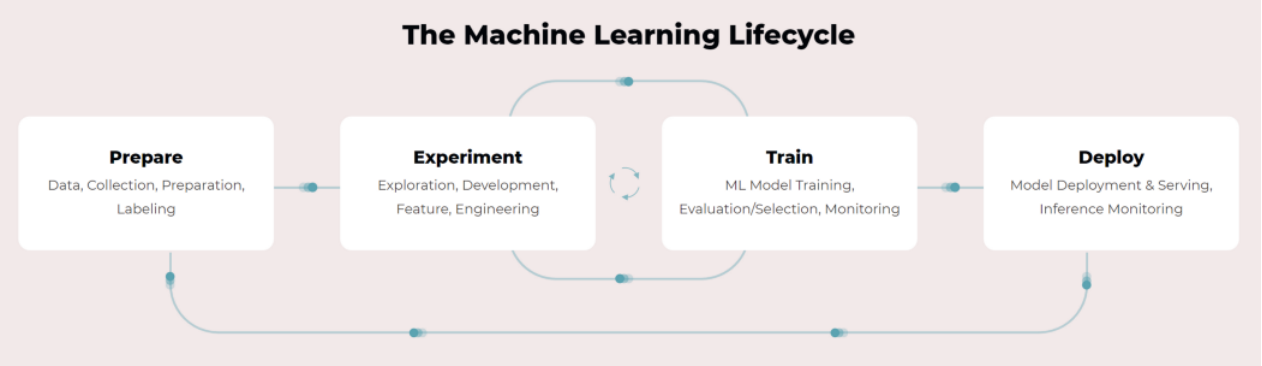


Figure 2.4: MLops data lifecycle

2.5 Conclusion

In this chapter we have set the road map for the whole project adopting scrum master methodology as the best fit to our work then we put a cookbook introduced in high level design and low level design before taking a deep dive into machine learning from a theoretical scientific view .

Chapter 3

Release-1 Design and Deployment

In this chapter, we discuss Design decision regarding two levels high and low levels design .In other hand we deploy our cluster before diving into monitoring stack setting up a some workaround where we refactor cluster components manifests and charts.

3.1 Release-1 Product backlog

The fist release composed of two main axes : Design and Deployment we have prepare this backlog table 3.1 which explain the design sprint (sprint 1) and the Infrastructure sprint (sprint 2)

—Release 1—	
to do ID /task	sprint number
F3;F4,F5	sprint-1
F8,F9,F10,F11,F12	sprint-2

Table 3.1: Release 1 planification

3.2 Sprint 1 : Design

3.2.1 High level Design

Purpose

During the preliminary stages of a development, the need of the project is to identify those parts of the project that might be at risk or time consuming. HLD provides a brief description of how the various sub-systems and components of the system fit together. As mentioned in

Design

Implementing solution in production environment is too risky because it may impact cloud stability that's why we use to work on a local test-bed environment to avoid production impact.

There are many types of computes used in Openstack private cloud such as OVS (Open vSwitch), SRIOV (single root input/output virtualization),AVRS (Accelerated Virtual Routing Switch).
however the common between all those types of compute is that they are Linux based Kernel hence our choice to use Centos virtual machine which act as a " logical compute node" in order to feed solution with metrics and logs ,from other hand deploying the whole Openstack solution require too much resources and security level that we could not warrant during development phase.

After identifying the principal logical zone we deal with the cloud native solution itself by defining global components and the purpose behind design decision. There are three main logical entity in the solution as below :

- **Monitoring zone**: composed of a real time monitoring stack composed from metrics scrapper tools, persistence storage,data (log and metrics) collector,and visualization.
- **Machine Learning zone**: consist of the online machine learning model and it's dependencies and libraries in a containerized form
- **Data Routing Engine**: an API used to scrap metrics data from Prometheus server web interface to feed machine learning models

The philosophy behind this is inspired from Artificial Intelligence of things "AIoT" workload where we use our sensors measurements to predict future results. From other hand, we fit all components to match Cloud Native approach by modernizing all monolith system in order to increase solution availability and scalability. After design negotiation, this is the final HLD target : 3.1

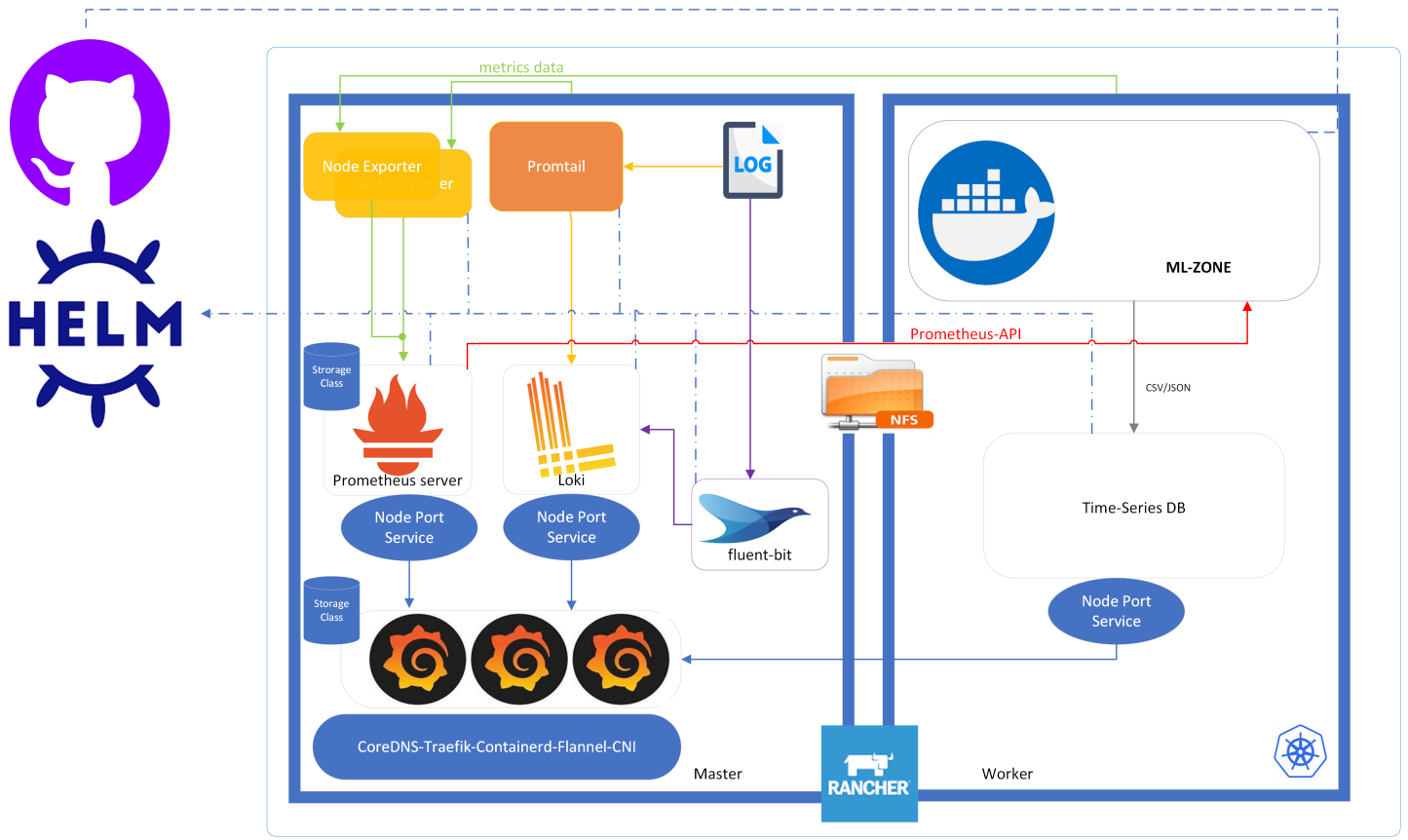


Figure 3.1: High Level Design

3.2.2 Low Level Design

Low Level Design, or "LLD," refers to the component-level design process and is similar to Detailing HLD. It provides the real logic of each system component and drills down to the specification of each module, and it offers a comprehensive description of each individual unit. Micro/detail design is another term for it. This section will be discussing networking configuration and the building process organization . briefly LLD will serve a cookbook for our Core solution in deployment phase and to keep historical documentation for future use.

Dimensioning

We need to set the dimension for the cluster entities to meet the high availability requirement. Furthermore, we should prioritize a vertical scale limited resource we use in terms of hardware capacity to ensure high performance. Machine learning necessitates greater computing power than the other components hence the physical separation between ML zone located in worker node and Monitoring zone located in master node , the table below 3.2 describe initial dimension for master node since worker node still empty at this level in it will be included in A.1 :

—Monitoring Zone —				
Namespaces	Node	Initial Pods	Actual/ReplicaSets	Services
default(Monitoring)	Server (Master)	prometheus-node-exporter	1/1	Cluster IP
default	Server (Master)	prometheus-alertmanage	1/2	NodePort
default	Server (Master)	prometheus-pushgateway	1/1	NodePort
default	Server (Master)	prometheus-server	1/2	NodePort
default	Server (Master)	Prometheus-kube-state-metrics	1/1	Cluster IP
default	Server (Master)	loki-fluent-bit-loki	1/1	Cluster IP
default	Server (Master)	loki-promtail	1/1	Cluster IP
default	Server (Master)	promtail	1/1	Cluster IP
default	Server (Master)	grafana	1/1	NodePort
default	Server (Master)	loki	1/1	NodePort
—K3s sysyem pods —				
kube-system	Server (Master)	svclb-traefik	1/1	Cluster IP
kube-system	Server (Master)	Coredns	1/1	Cluster IP
kube-system	Server (Master)	metrics-server	1/1	Cluster IP
kube-system	Server (Master)	local-path-provisioner	1/1	Cluster IP
—Machine Learning Zone —				
Machine learning	Worker	model and dependencies	2/2	LoadBalancer
Machine learning	Worker	database	1/2	NodePort

Table 3.2: Cluster Dimension

IP plan

we have configure the IP plan for both Virtual machines in Vagrant file as shown below :

- Mask : 255.255.255.0
- NAT
- Gateway : 192.168.253.0
- DNS : 192.168.253.135
- master : 192.168.253.135
- worker : 192.168.253.134

In addition to that, when going more deep in cluster layer and we realize that fortunately a similar mechanism is running on cluster as a top agent will automatically use a range of address to identify network policy : DNS provider, Ingress controller service, and loadbalancer services which are mostly come with base installation in lightweight Kubernetes to auto configure cluster networking as shown in the cluster IP plan illustrated in 3.3 where we describe pods internal/external IP address and internal/external pod's port . the namespaces summarize the logical separation between Kube-system components and monitoring zone components using namespaces ,the whole services is ilustrated in A.2

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
default	kubernetes	ClusterIP	10.43.0.1	none	443/TCP
kube-system	kube-dns	ClusterIP	10.43.0.10	none	53/UDP,53/TCP ,9153/TCP
kube-system	metrics-server	ClusterIP	10.43.149.96	none	443/TCP
default	prometheus-node-exporter	ClusterIP	None	none	9100/TCP
default	prometheus-kube-state-metrics	ClusterIP	10.43.232.50	none	8080/TCP
default	prometheus-pushgateway	ClusterIP	10.43.183.16	none	9091/TCP
default	prometheus-server	NodePort	10.43.116.209	none	80:31539/TCP
default	prometheus-alertmanager	NodePort	10.43.50.144	none	80:31840/TCP
kubernetes-dashboard	kubernetes-dashboard	ClusterIP	10.43.140.122	none	443/TCP
kubernetes-dashboard	dashboard-metrics-scraper	ClusterIP	10.43.131.144	none	8000/TCP
default	loki-headless	ClusterIP	None	none	3100/TCP
default	my-release-influxdb	NodePort	10.43.169.53	none	8086:30083/TCP, 8088:32732/TCP
default	loki3-headless	ClusterIP	None	none	3100/TCP
default	loki3	NodePort	10.43.16.148	none	3100:31317/TCP
grafana	grafana	NodePort	10.43.244.135	none	80:31381/TCP
kube-system	traefik	LoadBalancer	10.43.70.26	192.168.253.135	80:31047/TCP, 443:31276/TCP

Table 3.3: IP/PORT plan

Monitoring Zone

Basically the whole monitoring stack will be the first subsystem turned alive to collect as much as possible of data and to monitor infrastructure behavior during model test and validation process . we have to mention also that this zone is running on master node. however it gather the worker node metrics also .

Machine Learning Zone

As mentioned before we will give a high priority in term of resources for this zone following a "continuous everything" life cycle based on GitOps workflow to keep a backup for every version so in case of bugs we rollback to a previous version. this zone is running on worker node only where it send output prediction to Grafana which is located in the master node . we start this application development in local environment using Anaconda. Mostly, we need to turned into a microservices because we have to run it inside the cluster hence the following 3.2 modernization workaround to adapt it to our need.

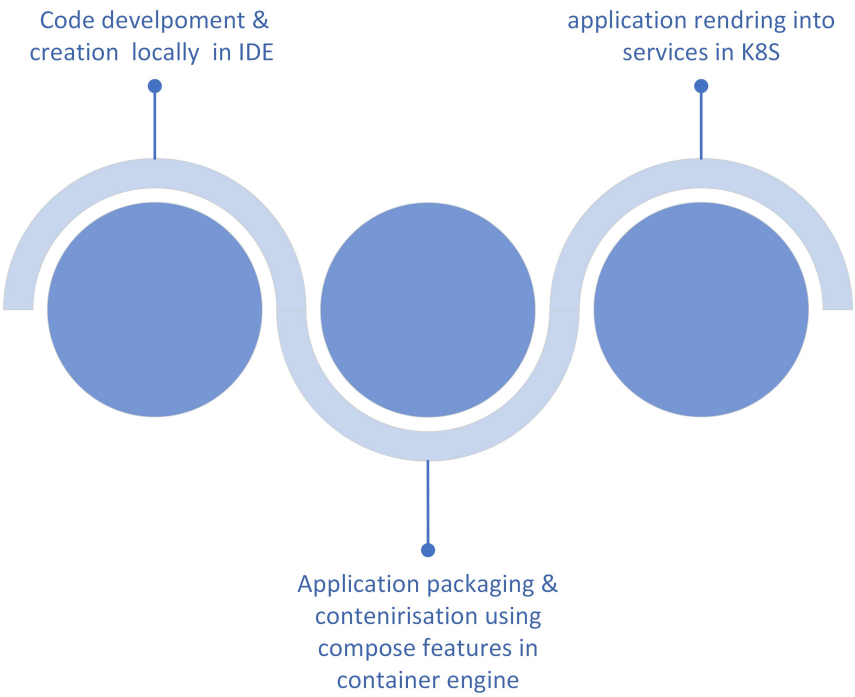


Figure 3.2: ML application building process

Data Redirection

Initially this part was initiated by implementing MQTT to route log message from log file into python model however when we realise that forecasting textual data is a waste of time we decided to move forward and use an Application program interface "API" which work side by side with data collector engine to feed Machine learning Model with metrics data

Turning this microservices into API minimize ressources consumption and secure the tunnel between the data producer and data consumer compared to MQTT workflow . In addition to that, the API zone is encapsulated in previous zone which doesn't require any physical apparition in architectural design .

3.3 Sprint 2 : Infrastructure

3.3.1 Hardware Environment

Identifying the hardware environment is an important thing in order to set a minimal requirements in future project reintegration .meanwhile, In our case we use a laptop with the specifications below:

- CPU : Core i7 8th vpro
- Memory: 16 Gb
- Hard disk: 512 SSD

3.3.2 Infrastructure Provisioning

Dealing with infrastructure provision is an important step for a goal to build a hosting stable mechanism for the entire application using best available technology's

Server provisioning

Server provisioning is the process of configuring a server for use in a network based on the resources that are required. Provisioning can involve all of the actions required to construct a new machine and bring it alive, as well as defining the system's desired state. Based on this we introduce infrastructure as a code using Vagrant hence a dynamic workflow for future deployments. Meanwhile, we use to work with VMWare desktop as VM provider.
This combination illustrated in 3.3 is a key factor behind bringing this project live :

Virtual machine Provider

We have assist earlier in a technical benchmark of VirtualBox versos VMware desktop which ranked Vmware in the first place in term of performance for main reasons that it has a good Network mechanism which turn VM communication easier and its friendly with our already implemented technology's in Core cloud.

Figure 3.3: VMware Desktop logo

Iac tools

Infrastructure as a code tool provide a dynamic provision where the deployment and configuration process of virtual machine are both involved to render a scaled output . Vagrant as an Iac fit our need in this mission using its simple vagrantfile syntax for VM provisioning where we define hostname and IP addresses also for defining VM provider and call back for provision shell script to install indeed tools automatically. The Vagrantfile is a Ruby file used to configure Vagrant on a per-project basis. The main function of the Vagrantfile is to described the virtual machines required for a project as well as how to configure and provision these machines. as illustrated in this example 3.4 we have bring the infrastructure using "vagrant up"

```
Vagrant.configure("2") do |config|

  config.vm.provision "shell", path: "install-docker.sh"

  config.vm.provision "shell", path: "custom-linux.sh"


  config.vm.define "master" do |master| #master node |
    kubemaster.vm.box = "centos/8"
    kubemaster.vm.hostname = "master"
    kubemaster.vm.network "private_network", ip: "192.168.253.135"
    kubemaster.vm.provider "VMaware Desktop" do |v|
      v.name = "master"
      v.cpus = 4
      v.memory = 4096
    end
    kubemaster.vm.provision "shell", path: "install-nfs-server.sh"
    kubemaster.vm.provision "shell", path: "kube.sh"
  end
```

Figure 3.4: VagrantFile syntax

3.3.3 Cluster provisioning

A cloud native approach need an orchestration software to increase availability among hosted application and services however choosing the best match for our case is a critical process hence the following benchmark in order to identify which one match our need

minikube

Minikube could run a local kubernetes cluster unfortunately it run a single node cluster for non production use only . which make it a bad choice regarding availability and scalability requirements [16]

Figure 3.5: Minikube logo

K8S

is an open-source system for automating deployment, scaling, and management of containerized applications and offer a high availability ,However bringing a whole kubernetes cluster locally could not be a good idea regarding limited resources.[15]

Figure 3.6: Kubernetes logo

Rancher K3S

K3s is an official CNCF sandbox project that delivers a lightweight yet powerful certified Kubernetes distribution designed for production workloads across resource-restrained, remote locations or on IoT devices. Meanwhile it offer high availability and easy upgrading mechanism for both master and workers nodes . those features made from K3s the best fit for our uses cases .[14]

Figure 3.7: K3S rancher logo

3.3.4 Monitoring stack Deployment and Configuration

After setting up the infrastructure and being sure that it work properly we proceed with deploying and configuring the monitoring stack to provide a clear vision about system behavior in realtime way .

3.3.5 Helm package manager

Helm is the first Kubernetes-based application package manager. It allows us to describe the app's structure with simple helm-charts and manage it with simple commands. Instead of creating manifest file from scratch for every namespaces resource which is a complex and long process the helm automatically deploy any tools into K3s using charts [13]

Figure 3.8: helm package manager

3.3.6 Metrics-operator

In order to organize the system we use to gouped the components by the type of output data. As a result of that we create two namespaces "metrics"

and "logs" .when Prometheus operator chart it come with all in one deployment package including Node-exporter,Alert-manager,Pushgatway and Prometheus-server.

Prometheus-server

the Prometheus server is the data collector for metrics KPI it come with a persistence volume and deployed into stateless mode in order gather data from multiple source .

Figure 3.9: Prometheus logo

By default Prometheus service was accessible only from inside the cluster that's why we need to edit the service in order to expose it's port to the public and access web user interface with node address and forwarded port instead of creating an external IP . if every is truly configured the user interface will appear like that 3.10

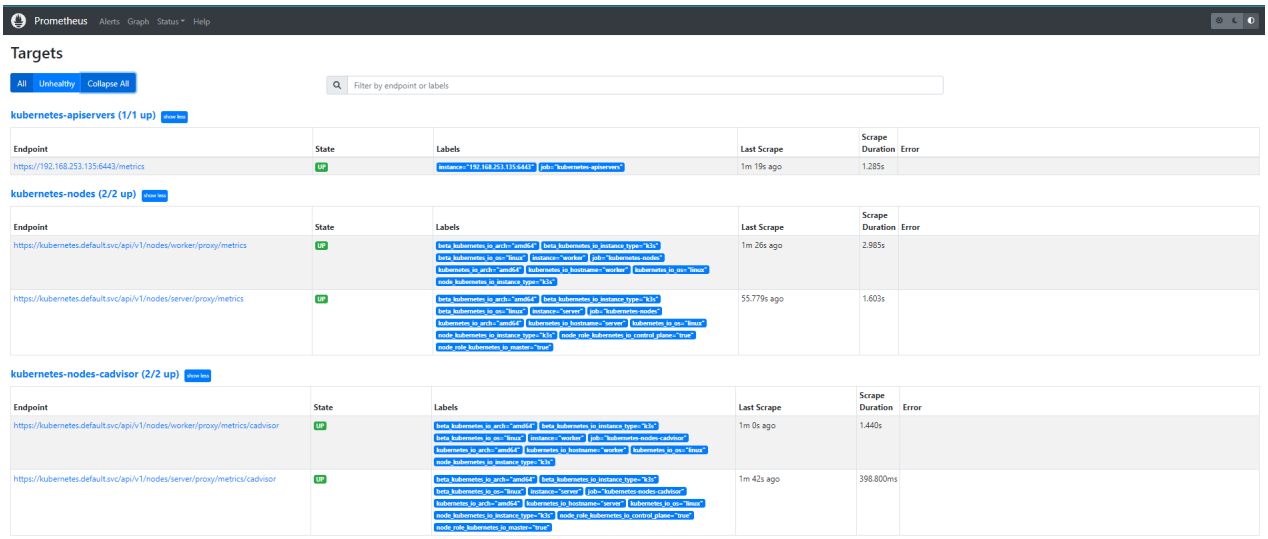


Figure 3.10: Prometheus User Interface

Node-exporter

Node exporter is an official Prometheus exporter for capturing all the Linux system-related metrics. It collects all the hardware and Operating System level metrics that are exposed by the kernel. we can use the node exporter to collect the system metrics from all your Linux systems.

Prometheus-pushgatway

The Prometheus Pushgateway allows you to push time series from short-lived service-level batch jobs to an intermediary job which Prometheus can scrape. Combined with Prometheus's simple text-based exposition format, this makes it easy to instrument even shell scripts without a client library.

Alert-manager

The Alertmanager handles alerts sent by client applications such as the Prometheus server. It takes care of duplicating, grouping, and routing them to the correct receiver integration such as email, Teams,slack or any supported notification channel . It also takes care of silencing and inhibition of alerts. reproducing the same workload for service exposing we could the access the user interface 3.11:

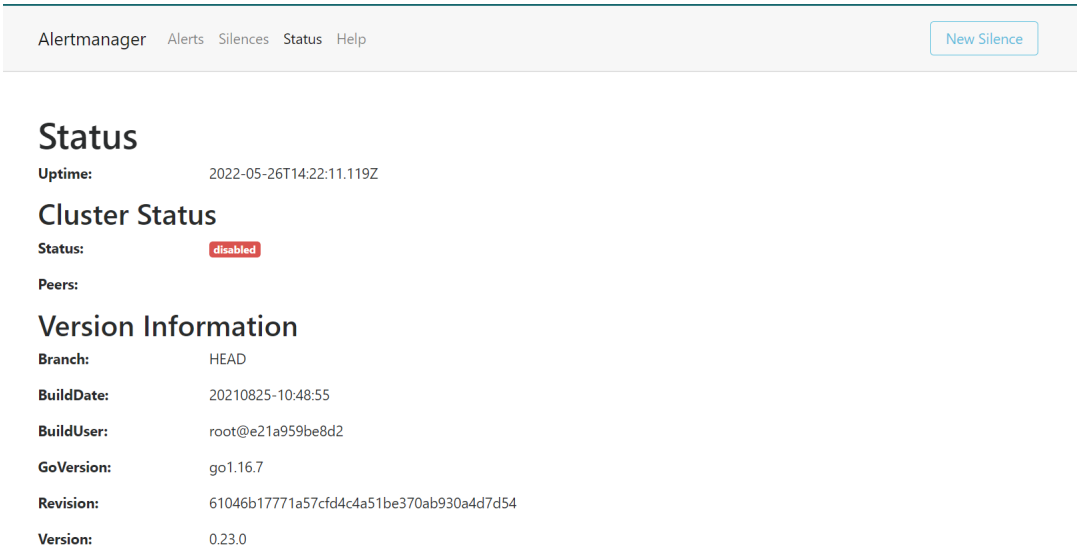


Figure 3.11: AlertManager User interface

3.3.7 Logs-operator

The logs are nedeed information triggered among the system services and scrapped to log collector engine in order to be visualized latter in human readable way there are a several log scrapper for Loki such as Filebeat with logstach , Promtail ,Fluent-bit, however in the installation guide grafana team recommend Promtail to ship logs to Loki as the configuration is very similar to Prometheus. This allows us to ensure that labels for metrics and logs are equivalent by re-using the same scrape config and relabeling configuration.

Promtail

Promtail is an agent which ships the contents of local logs to a private Grafana Loki instance or Grafana Cloud. It is usually deployed to every machine that has applications needed to be monitored.

It primarily:

- Discovers targets
- Attaches labels to log streams
- Pushes them to the Loki instance

Loki

Loki is a horizontally scalable, highly available, multi-tenant log aggregation system inspired by Prometheus. It is designed to be very cost effective and easy to operate. It does not index the contents of the logs, but rather a set of labels for each log stream. as illustrated in 3.12

The Loki project was started at Grafana Labs in 2018, and announced at KubeCon Seattle. Loki is released under the AGPLv3 license. [3]

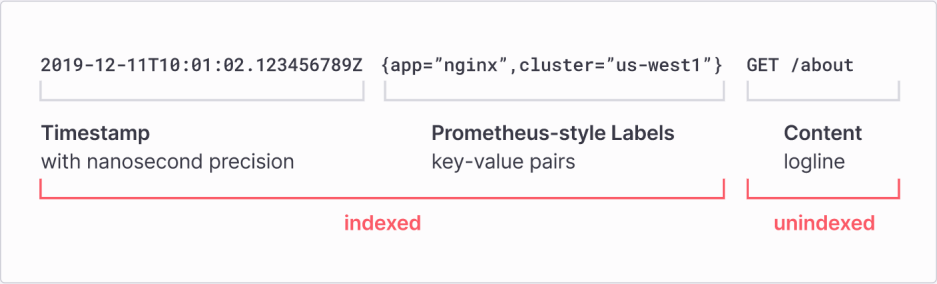


Figure 3.12: Loki metadata indexing

we have choose loki as log engine collector instead of many others alternatives such the famous Elasticsearch because Loki takes a unique approach by only indexing the metadata rather than the full text of the log lines instead of indexing the whole data which cause a huge resource consumption like in Elasticsearch case .

In our case we will deploy both promtail and fluent-bit as log scrapper by adding two "–set" argument in the installation level summarized in result A.3

Before putting Loki as a data source in Grafana we have expose port by changing Loki service type to NodePort instead of ClusterIP .

Fluent-bit

Fluent Bit is a super fast, lightweight, and highly scalable logging and metrics processor and forwarder. It is the preferred choice for cloud and containerized environments..

On 2014, the Fluentd team at Treasure Data forecasted the need of a lightweight log processor for constraint environments like Embedded Linux and Gateways, the project aimed to be part of the Fluentd Ecosystem and

we called it Fluent Bit, fully open source and available under the terms of the Apache License v2.0. [2]

Meanwhile to build a persistence log data gathering services we implement Fluent-bit with Promtail so in case of failure one of alive component will still send log to Loki .

Figure 3.13: Fluent-bit logo

In order to scrap syslog config we have to change manifset yaml file for fluentd after deploying by adding syslog file path into ,In other hand if Loki isn't marked as an output in manifest it could generate an error that cause pod crash because fluent-bit goes into a loopBackCrash situation

3.3.8 Centralized visualization

Grafana is an open source solution for running data analytics, pulling up metrics that make sense of the massive amount of data to monitor our apps with the help of cool customized dashboards.

it serve to connects with every possible data source, commonly referred to as databases such as Graphite, Prometheus, Influx DB, ElasticSearch, MySQL, PostgreSQL etc.

Being an open source solution also enables us to write plugins from scratch for integration with several different data sources.

Figure 3.14: Grafana logo

The tool helps us study, analyse , monitor data over a period of time, technically called time series analytics.

It helps us track the user behaviour, application behaviour, frequency of errors popping up in production or a pre-prod environment, type of errors popping up and the contextual scenarios by providing relative data.

A big upside of the project is it can be deployed on-prem by organizations which do not want their data to be streamed over to a vendor cloud for security purpose. [1]

Same as other operator Grafana has it's own helm chart and offer many customization features when deployed on top of cluster,by default it does not come with a persistence storage which may cause configuration lose in case of pod crash ,hence the need for a custom configuration in grafana charts to avoid this issues using bellow arguments 3.15:

`persistence.enabled`	Use persistent volume to store data	`false`
`persistence.type`	Type of persistence (`pvc` or `statefulset`)	`pvc`
`persistence.size`	Size of persistent volume claim	`10Gi`
`persistence.existingClaim`	Use an existing PVC to persist data (can be templated)	`nil`
`persistence.storageClassName`	Type of persistent volume claim	`nil`
`persistence.accessModes`	Persistence access modes	`[ReadWriteOnce]`
`persistence.annotations`	PersistentVolumeClaim annotations	`{}`

Figure 3.15: Persistence volume

which cause another unattended issue in the Container networking interface where grafana user interface is no longer accessible by master IP instead of that it become accessible using worker IP .by the way exposing services make it accesbile using combination of worker IP and external port .

As a result for all previous workload we have build many useful dashboard for Loki data source such as below example 3.16 where the dashboard act as a search engine for any specific pattern that could appear in log file

Figure 3.16: Loki log dashboard

although the big number of log events we observe that there is no latency where loading dashboard , Loki achieve this accomplishment because it only index metadata instead of indexing the entire log as illustrated in this dashboard A.4.

Hence the low resource consumption compared to other log search engine alternative tools such Elasticsearch regrading the experimental result illustrated in Loki stack dashboard 3.17 where the dedicated resources isn't fully used

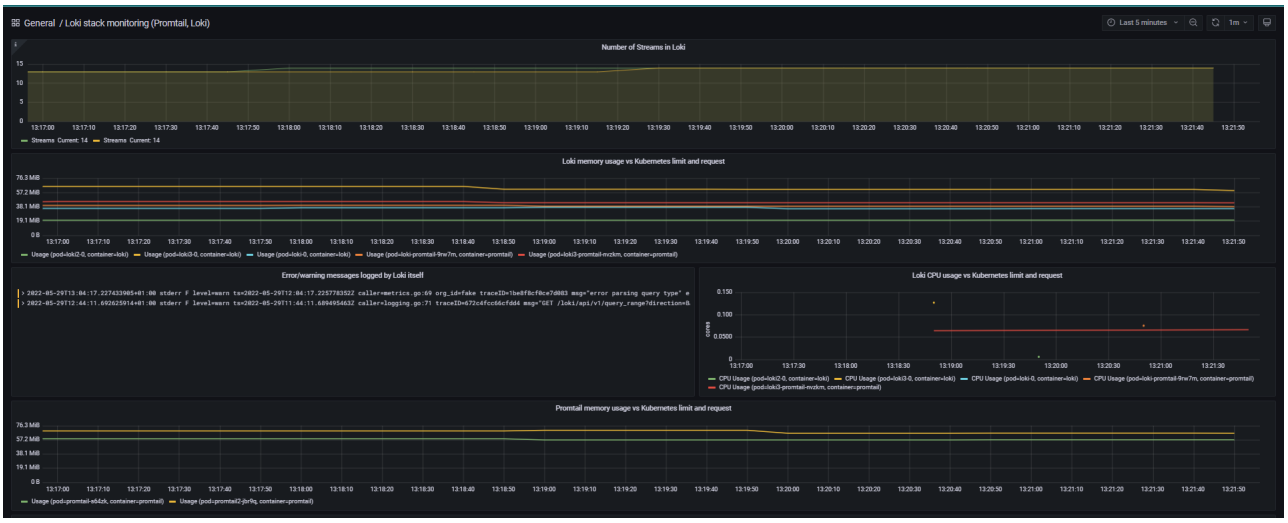


Figure 3.17: Loki resource consumption

as a result of that we have built a combined data source dashboard to optimize our vision using the same Grafana instance for all data entry A.5

This workload really makes problem troubleshooting faster because it takes our team to a higher level of supporting cloud components and changes the way we troubleshoot system issues from manually searching in log files containing tone of events to an easy human readable metrics dashboard and a fast log search engine in a centralized visualization platform for both log and metrics as illustrated in A.6

3.4 conclusion

In this chapter, we set up the entire infrastructure, which consists of two nodes, and then successfully deploy and configure the monitoring system, which is divided into two stacks: metric and logs, eventually linking all stacks with Grafana. We've also added the appropriate workaround to fix configuration lost at the Grafana level.

Chapter 4

Release 2 : AiOps and Resiliency Workload

In this chapter We take a deep dive into machine learning in this chapter, where we do some data transformation after importing it using a based API engine, then we implement two model, to proceed after that with some optimization in order to modernize a ready to use python application in the end.

4.1 Release-2 Product backlog

In this release we implement the AiOps workload including the process of building a machine learning modernized application where we summarize the process of sending the output to Grafana by the end of workload .However, we have to mention also that we will add some additional functions to the whole solution in a separated sprint (sprint-4) as illustrated in the table 4.1

—Release 2—	
to do ID /task	sprint number
F13,F14,F15,F16,F17,F18	sprint-3
F19,F20	sprint-4

Table 4.1: Release 2 planification

4.2 Sprint 3 AiOps workload

After identified the best practices for our machine learning application in a previous theoretical study we use to implement the models that fit our requirement following a clear Aiops workload regarding data lifecycle in the system.

4.2.1 Software requirement

before going into data importing ,transformation and loading process we have to introduce that during the development phase we have use Anaconda where Conda is an open-source package and environment management system that runs on Windows, macOS, and Linux. Conda quickly installs, runs, and updates packages and their dependencies. It also easily creates, saves, loads, and switches between environments on your local computer. It was created for Python programs, but it can package and distribute software for any language [12]

Figure 4.1: Anaconda logo

4.2.2 Data Gathering

We already have metrics data collected in the previous sprint by Prometheus server therefore we didn't need anew mechanism to collect data the again.Hence using an API to scrap the same data while the Prometheus Service is exposed to public . then we have to submit the GET requests and we'll get the data

we looking for , The procedure is straightforward. To transmit the request and the JSON module, we only need to import the get method from the requests library.

After that, we add the required information to the API URL, in this case,Prometheus has a ready to use API ,by parsing all metrics using predefined function "Prom.all-metrics()" which parse all existing metrics as mentioned in 4.2:

```
from prometheus_api_client import PrometheusConnect
prom = PrometheusConnect(url ="http://192.168.253.135:31539/", disable_ssl=True)
# Get the list of all the metrics that the Prometheus host scrapes
prom.all_metrics()
```

```
['aggregator_openapi_v2_regeneration_count',
 'aggregator_openapi_v2_regeneration_duration',
 'aggregator_unavailable_apiservice',
 'aggregator_unavailable_apiservice_total',
 'apiextensions_openapi_v2_regeneration_count',
 'apiserver_admission_controller_admission_duration_seconds_bucket',
 'apiserver_admission_controller_admission_duration_seconds_count',
 'apiserver_admission_controller_admission_duration_seconds_sum',
 'apiserver_admission_step_admission_duration_seconds_bucket',
 'apiserver_admission_step_admission_duration_seconds_count',
 'apiserver_admission_step_admission_duration_seconds_sum',
 'apiserver_admission_step_admission_duration_seconds_summary',
 'apiserver_admission_step_admission_duration_seconds_summary_count',
 'apiserver_admission_step_admission_duration_seconds_summary_sum',
 'apiserver_audit_event_total',
 'apiserver_audit_requests_rejected_total',
 'apiserver_client_certificate_expiration_seconds_bucket',
 'apiserver_client_certificate_expiration_seconds_count',
 'apiserver_client_certificate_expiration_seconds_sum',
```

Figure 4.2: Prometheus all-metrics

Currently we are sure that data gathering is going well and the API is working fine we move to parsing parameter such as "start time" or which metrics to scrap its values and chunk-size using two important class "metricSnapShotDataFrame" and "metricRangeDataFrame" if a table is shown as illustrated in 4.3 we could proceed to the data transformation phase. using the chosen metrics for the entire upcoming process.

```
from prometheus_api_client.utils import parse_datetime
from prometheus_api_client import PrometheusConnect,  MetricSnapshotDataFrame, MetricRangeDataFrame

from datetime import timedelta

start_time = parse_datetime("5w")
end_time = parse_datetime("now")
chunk_size = timedelta(days=1)

metric_data = prom.get_metric_range_data(
    "node_memory_MemAvailable_bytes",  # this is the metric name and label config
    start_time=start_time,
    end_time=end_time,
    chunk_size=chunk_size,
)
```

```
metric_df = MetricSnapshotDataFrame(metric_data)
metric_df.head()
```

_io_managed_by	chart	component	heritage	instance	job	namespace	node	release	service	timestamp	value
Helm	prometheus-15.8.5	node-exporter	Helm	192.168.253.135:9100	kubernetes-service-endpoints	default	server	prometheus	prometheus-node-exporter	1.652452e+09	805883904
Helm	prometheus-15.8.5	node-exporter	Helm	192.168.253.135:9100	kubernetes-service-endpoints	default	server	prometheus	prometheus-node-exporter	1.652533e+09	741416960
Helm	prometheus-15.8.5	node-exporter	Helm	192.168.253.135:9100	kubernetes-service-endpoints	default	server	prometheus	prometheus-node-exporter	1.652575e+09	813064192
Helm	prometheus-15.8.5	node-exporter	Helm	192.168.253.135:9100	kubernetes-service-endpoints	default	server	prometheus	prometheus-node-exporter	1.652800e+09	835305472
Helm	prometheus-15.8.5	node-exporter	Helm	192.168.253.135:9100	kubernetes-service-endpoints	default	server	prometheus	prometheus-node-exporter	1.652813e+09	877137920

Figure 4.3: API parameter

4.2.3 Data Transformation

The process of modifying the format, structure, or values of data is known as data transformation,this require a pre-processing phase in our case where we extract only needed column and ignore others . we prioritize date column regarding our final goal which is forecasting time series .However the gathered data in its first version is not supported by machine learning model's.in fact, turning it into supported input from initial timestamp format to timedate format before get feeded to the model.

An other mathematical transformation process will take place for the metrics itself in order to avoid showing exponential number ,taking for example the available memory metrics known as " node-memory-MemAvailable-bytes " at least it give a number of 9 digits ,logically turning it to normal integer need to convert the bytes to Gbytes using this expression :

node-memory-MemAvailable-bytes(bytes)/1000000000 = x (Gbytes)

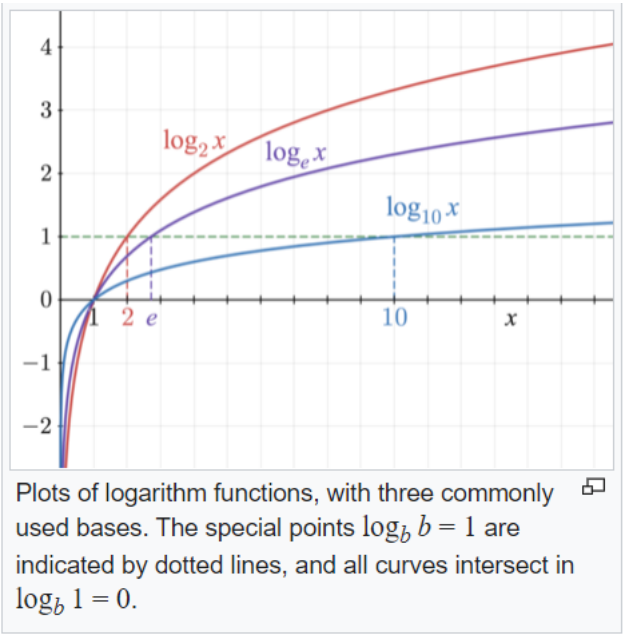
Unfortunately, due to the diversity of measurement units, we need a custom conversion statement to convert all metrics units to a human readable

number. Also we have observed that all of the initial values are given in a tiny units, therefore using logarithm the inverse function of exponentiation in mathematics.where the logarithm of a given number x is the exponent to which another fixed number, the base b, must be raised in order to obtain that number x.

The logarithm counts the number of times the same factor appears in repeated multiplication in the simplest case, When there is no possibility of confusion, the logarithm of x to base b is expressed as logb (x) , or even without the explicit base using Napierian logarithm (base 10) in our case :

Metrics [small unit] = Log(metrics) [log small unit]

The observation of plot is more meaningful now with log(metrics) instead of metrics exponent n value where the y axes has readable number by using logarithm in base 10 4.4:



Plots of logarithm functions, with three commonly used bases. The special points $\log_b b = 1$ are indicated by dotted lines, and all curves intersect in $\log_b 1 = 0$.

Figure 4.4: plot log equation

4.2.4 Darts Library

After a deep search into the world of machine learning Darts is a Python library for easy manipulation and forecasting of time series. It contains a variety of models, from classics such as ARIMA to deep neural networks. The models can all be used in the same way, using fit() and predict() functions, similar to scikit-learn. The library also makes it easy to backtest models, combine the predictions of several models, and take external data into account. Darts supports both univariate and multivariate time series and models. The ML-based models can be trained on potentially large datasets containing multiple time series, and some of the models offer a rich support for probabilistic forecasting. [10]

4.2.5 Fixing missing data issues

When dealing with missing data, we have two options for resolving this issue imputation or data removal.

For missing data, the imputation process generates credible predictions. When the percentage of missing data is low, it's the most beneficial. If the percentage of missing data is too high, the results will be devoid of natural variation, which will make it difficult to build an effective model.

The other option is to delete information. To eliminate bias when dealing with data that is absent at random, relevant data can be erased. If there aren't enough observations to do a reliable analysis, removing data may not be the best solution. Observation of specific events or factors may be essential in some circumstances.

Before settling on a strategy we must say that in our case the data gap is due to the fact that when system is shut down the measurements are not considerable .

Usually we don't face this problem in production environment because such a situation isn't reproduced regarding zero down time monitoring system with a small annual incertitude which turn the gap invisible .

Moreover ,filling missing data will use the seasonality parameter to generate a fake data in order to avoid model dysfunction . the below graph 4.5 illustrate the source cause beyond this problem during 12H observation.

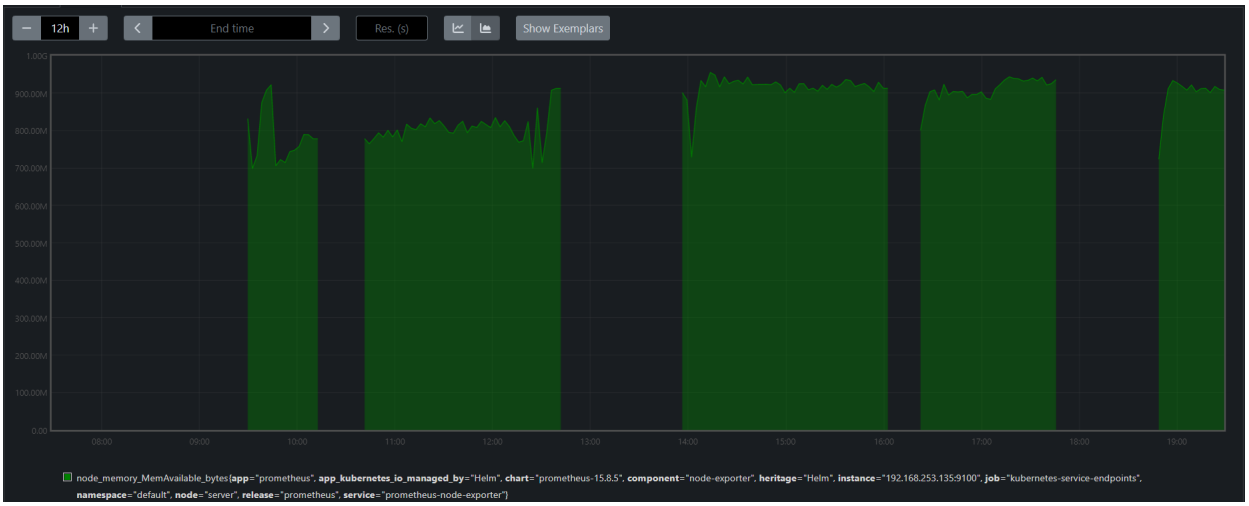


Figure 4.5: Missing data graph

4.2.6 Seasonality

Seasonal fluctuation may be present in time series data. Seasonal variation, often known as seasonality, refers to cycles that occur on a regular basis over time. Seasonal variation refers to a recurring pattern within each year, though the phrase is more broadly applied to recurring patterns within any set period.

Darts offer a mechanism for checking seasonality if its possible to capture so we set an extra parameter to parse the seasonality frequency .in this figure 4.6 the blue area present the possible value while the small separators are the seasonality frequency which where they must be equals for the whole used period

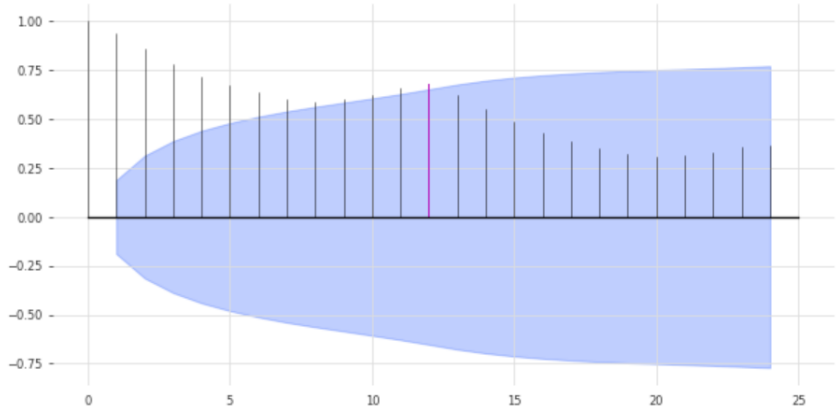


Figure 4.6: checking seasonality plot

4.2.7 Model accuracy

Darts aim a big range of time series forecasting model's , choosing which model have the best accuracy with our dataset involve many factors such as training time,model accuracy ,resource consumption ...etc

choosing the best model in our case is too relative regarding the non-fixed dataset where the API append a new rows to the initial data in every request by the way we have choose Naive-Seasonal and Theta introduced by Darts as a good model accuracy using.

4.2.8 Theta Model

The Theta model of Assimakopoulos and Nikolopoulos (2000) is a simple method for forecasting the involves fitting two lines, forecasting the lines using a Simple Exponential Smoother, and then combining the forecasts from the two lines to produce the final forecast. The model is implemented in steps:

- Test for seasonality
- Deseasonalize if seasonality detected
- Estimate by fitting a SES model to the data and by OLS.
- Forecast the series

- Reseasonalize if the data was deseasonalized

Ultimately only plays a role in determining how much the trend is damped. If is very large, then the forecast of the model is identical to that from an Integrated Moving Average with a drift, Finally, the forecasts are reseasonalized if needed.[5]

Theta has a very good rank based on Darts community ,however when trying this model in our real data it seem the model need more rows to give a comprehensible result 4.7 :

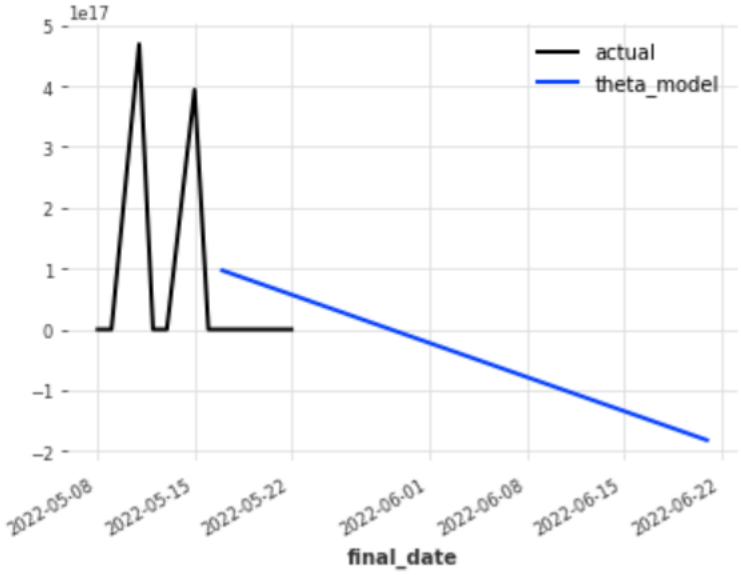


Figure 4.7: Theta prediction

4.2.9 Naive Seasonal

is a simple forecasting method that can be considered as a good benchmark in many situations. Similar to Naïve, Seasonal Naïve relies only on one observation, but instead of taking the most recent value, it uses the value from the same period a season ago. For example, for producing a forecast for January 1984, we would use January 1983. Mathematically this is written as:

$$yt = ytm$$

where m : is the seasonal frequency. This method has an underlying model, Seasonal Random Walk:

$$yt = ytm + t$$

Similar to Naïve, the higher variability of the error term "t" is the faster the data exhibit changes. Seasonal Naïve does not require estimation of any parameters and thus is considered one of the popular benchmarks to use with seasonal data.[18]

after splitting data into training and validation the model give the following result 4.8 tested on Available memory measurements

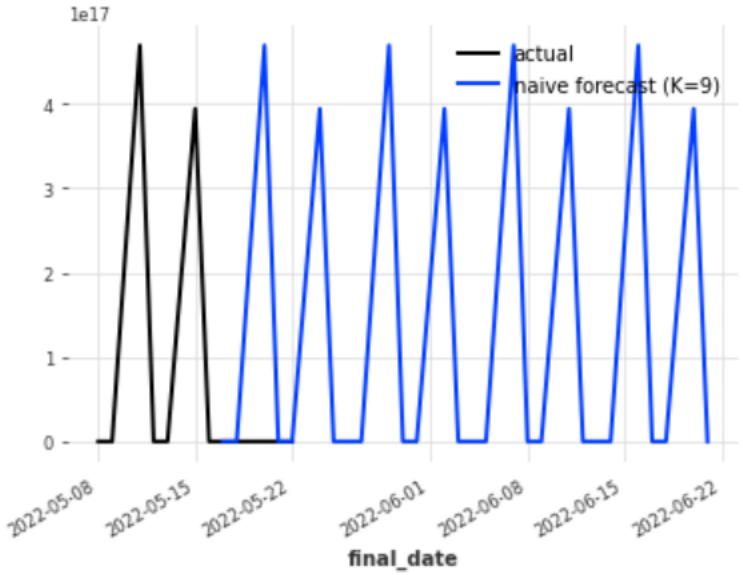


Figure 4.8: Naive seasonal prediction

4.2.10 Solution modernization

basically the monitoring zone already modernized using helm in all deployment we proceed for modernising the python application and its database.

4.2.11 Database Deploying

Before dealing with modernizing the python application we have to set which database we are going to use to store predicted data which are outputted into CSV format.

We involved Influxdb as a time series database management system in the early stages of our project. We successfully deployed influxdb and Mimir as alternatives in a lab environment, where we discovered that those open source solutions are built to be scalable, highly available, multi-tenancy durable storage, which are now optional needs. On the other hand, the output only covers the next three months "12w," which is a very short period. Given the size of our data, we opted to choose the smallest database option possible, such as mysql, which is also familiar with Grafana.

basically we apply the manifest file then we expose the database port and make it accessible for others application, meanwhile Grafana need a read only access where the python application need read write privilege so it could insert new predicted data every time .

Figure 4.9: database migration plan

4.2.12 Application Modernization

Modernization means taking our application environment in the form that it's in today most likely, legacy and monolithic and transforming it into

something that is more agile, elastic, and highly available. We have already mention before that when modernizing Any monolith application developed locally we have to consider two stages : turning the application into an image using Dockerfile then building a manifest file using Podman ,we have involve two containerisation engine in this workload because they are familiar hence we install docker-cli and used side by side with Podman like illustrated in 4.10

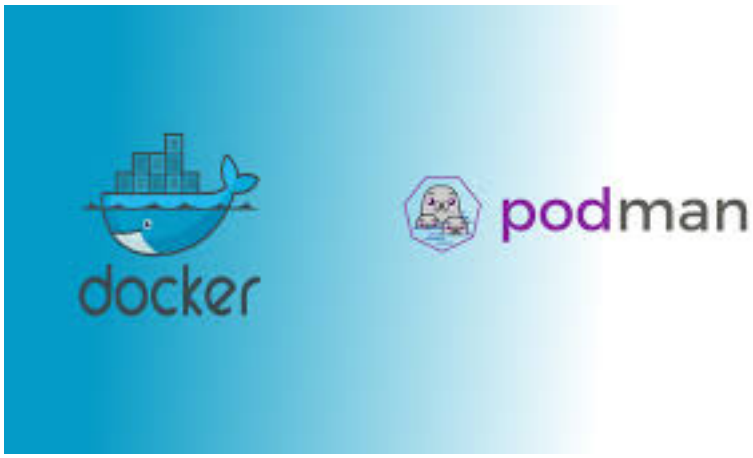


Figure 4.10: Docker and Podman

Docker file

To create a container image we use Docker Compose that allows us to define and operate multi-container Docker applications.We start with configuring our application's services via Compose using a YAML file. Then we build and start all of the services from our setup with a single command. "Docker-compose up"

In his phase we collect all used libraries and there dependencies in a requirement text file under in a sub path under the application environment,where we use a base image of Linux Ubuntu image instead of using Conda image in order to make container interactive mode easy to manage after running it . assuming the image is builded successfully we push it the to our registry in order to keep a backup.

Manifest file

when running a container in the cluster using docker image we have to build a manifest file with manually creating all application components in the cluster such as Service ,Deployment,Pod and a Configmap if needed . However, this features already enhanced by Podman which offer the possibility to create a manifest file based on a running container, this feature automatically create a default manifest file containing all basic components .

4.3 Sprint 4 : Additional Functions

After finishing all the solution deployment we move in this sprint and try some resiliency tests to identify weakness in the entire infrastructure and enforce solution reliability in real world. The key factor behind this accomplishment is introducing Chaos engineering .Before dealing with experimental workaround we summarize the meaning of Chaos engineering.

4.3.1 Principals

Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production.

Advances in large-scale, distributed software systems are changing the game for software engineering. As an industry, we are quick to adopt practices that increase flexibility of development and velocity of deployment. An urgent question follows on the heels of these benefits: How much confidence we can have in the complex systems that we put into production?

Even when all of the individual services in a distributed system are functioning properly, the interactions between those services can cause unpredictable outcomes. Unpredictable outcomes, compounded by rare but disruptive real-world events that affect production environments, make these distributed systems inherently chaotic.

We need to identify weaknesses before they manifest in system-wide, aberrant behaviors. Systemic weaknesses could take the form of: improper fallback settings when a service is unavailable; retry storms from improperly tuned timeouts; outages when a downstream dependency receives too much traffic; cascading failures when a single point of failure crashes; etc. We must address the most significant weaknesses proactively, before they affect

our customers in production. We need a way to manage the chaos inherent in these systems, take advantage of increasing flexibility and velocity, and have confidence in our production deployments despite the complexity that they represent.

An empirical, systems-based approach addresses the chaos in distributed systems at scale and builds confidence in the ability of those systems to withstand realistic conditions. We learn about the behavior of a distributed system by observing it during a controlled experiment. We call this Chaos Engineering.[8]

4.3.2 workflows

Chaos Workflows are a set of actions strung together to achieve desired chaos impact on a Kubernetes cluster. Workflows are an effective mechanism to simulate real world conditions gauge application behaviour in an effective manner

Deployment

deploying the litmus require helm3 and a persistence volume of 20G at least . the deployment could be installed using Yaml file or using helm as illustrated in figure 4.11

```
[root@server ~]# helm install chaos litmuschaos/litmus --namespace=litmus
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /root/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /root/.kube/config
NAME: chaos
LAST DEPLOYED: Mon Jun 13 21:42:43 2022
NAMESPACE: litmus
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing litmus 😀

Your release is named chaos and it's installed to namespace: litmus.

Visit https://docs.litmuschaos.io to find more info.
```

Figure 4.11: deploying Litmus with helm3

Analytics

We perform tests on infrastructure reliability where every workflow's have been scored . so we could explore system weakness in case of huge CPU requests or memory excited ,it could be either a CoreDns down ,this is a summary of statics for the current and scheduled workflow's 4.12. In addition

to that in the analytics section we summurize the workflow detail in B.1 and the Prometheus metrics and B.2

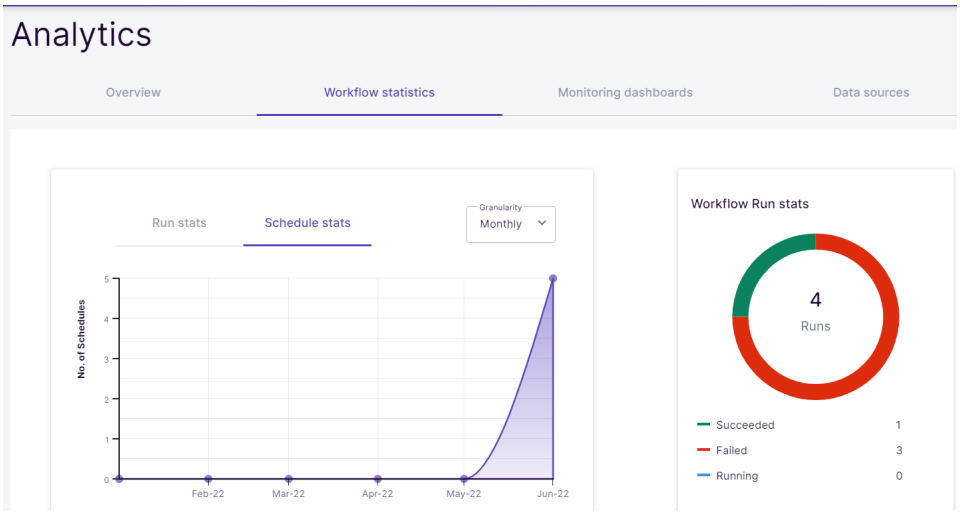


Figure 4.12: workflows statics

Results

Chaos engineering put the light on our solution weaknesses in term of resources limits and networking capabilities . it stand for identifying which component need more attention when implementing the solution in production environment.

We have performed four main predefined workflows in which we identify hardware weakness using CPU and memory huge tests and software weaknesses using DNS and Hypervisor tests .As a result of that we could scale the infrastructure and working with more node ,either work on site redundancy so in case of extreme disaster the entire solution raise on other site .

4.4 Conclusion

This chapter focuses on the AIops workload, in which we simulate two machine learning models after feeding them processed metrics data, and conclude with a modernization solution that transforms the system from a local monolith state to a cluster-running application.in other hand we have im-

plement Chaos engineering to test solution resiliency in order to enhance reliability in production uses case.

Conclusion

Initially in brainstorming phase we have faced the Openstack problems which make the used monitoring system incapable to accomplish the mission insight due the need for a new solution .Meanwhile we are inspired by Artificial intelligence of thing workaround where the monitoring operator aim to supervise connected devices based on a prediction mechanism. Hence, the idea of dealing with such a workload in a cloud native way. So, we start with designing the architectural map for our solution . then we split the solution into two main parts ,starting with Monitoring operator deploying which is composed of metrics stack and logging stack .In other hand we deal also with developing the machine learning application, which eventually gather its input from the previous operator using realtime API's to forecast measurements in the future. We have to mention also that the entire solution outputs are visualized in a single configured instance .
We aimed for many benefits more than what was expected at the beginning of this project. This research study not only creates new support tools for the Core infrastructure team, but also it changes the way we troubleshoot Openstack problems; previously, we used to check our cloud nodes on a variety of platforms such as Zabbix and Netact, however now we build a centralized monitoring stack for all upcoming measured data; on either side, Elasticsearch is disabled in our cloud due to its huge resource consumption where this cloud native edge monitoring stack minimize resource consumption as much as possible. Hence, the validation of deployed solution as a strong alternatives for ELK.Meanwhile we have tested many Machine learning model using metrics data in order to forecast time series measurement .In the second phase we validate the best ranked model with high performance on our Dataset to proceed for modernizing the application where we turn it into a running micro-services inside the cluster . This enhancement build in-house solution used to troubleshoot, monitor, and check the Telco cloud status in

a modern way . At the end of workload we introduce a resiliency mechanism in order to optimize the solution reliability in production environment.

Appendix A

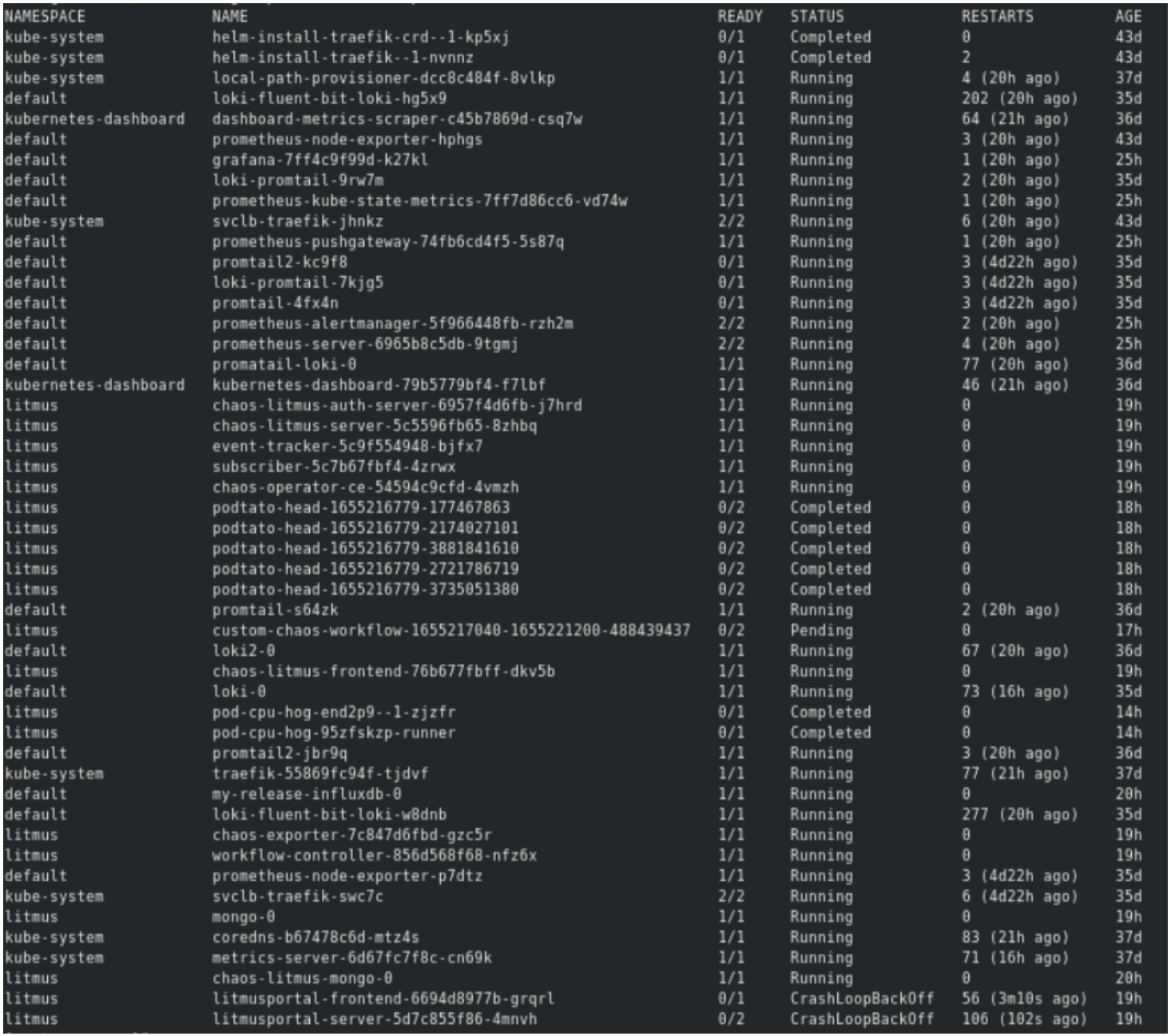
Release-1 Design and deployment

```
[root@server ~]# helm upgrade --install loki loki/loki-stack \
>     --set fluent-bit.enabled=true,promtail.enabled=true
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /root/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /root/.kube/config
WARNING: This chart is deprecated
W0529 12:13:03.196625   76059 warnings.go:70] policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
W0529 12:13:03.205656   76059 warnings.go:70] policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
W0529 12:13:03.245504   76059 warnings.go:70] policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
W0529 12:13:03.252067   76059 warnings.go:70] policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
W0529 12:13:03.258542   76059 warnings.go:70] policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
W0529 12:13:03.269094   76059 warnings.go:70] policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
W0529 12:13:03.281313   76059 warnings.go:70] policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
W0529 12:13:03.288524   76059 warnings.go:70] policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
W0529 12:13:03.298817   76059 warnings.go:70] policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
Release "loki" has been upgraded. Happy Helming!
NAME: loki
LAST DEPLOYED: Sun May 29 12:12:58 2022
NAMESPACE: default
STATUS: deployed
REVISION: 11
NOTES:
The Loki stack has been deployed to your cluster. Loki can now be added as a datasource in Grafana.

See http://docs.grafana.org/features/datasources/loki/ for more detail.
```

Figure A.3: Log-operator installation

Figure A.4: Metadata experimental dashboard



NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	helm-install-traefik-crd--1-kp5xj	0/1	Completed	0	43d
kube-system	helm-install-traefik--1-nvnnz	0/1	Completed	2	43d
kube-system	local-path-provisioner-dcc8c484f-8vlkp	1/1	Running	4 (20h ago)	37d
default	loki-fluent-bit-loki-hg5x9	1/1	Running	202 (20h ago)	35d
kubernetes-dashboard	dashboard-metrics-scraper-c45b7869d-csq7w	1/1	Running	64 (21h ago)	36d
default	prometheus-node-exporter-hphgs	1/1	Running	3 (20h ago)	43d
default	grafana-7ff4c9f99d-k27kl	1/1	Running	1 (20h ago)	25h
default	loki-promtail-9rw7m	1/1	Running	2 (20h ago)	35d
default	prometheus-kube-state-metrics-7ff7d86cc6-vd74w	1/1	Running	1 (20h ago)	25h
kube-system	svclb-traefik-jhnkz	2/2	Running	6 (20h ago)	43d
default	prometheus-pushgateway-74fb6cd4f5-5s87q	1/1	Running	1 (20h ago)	25h
default	promtail2-kc9f8	0/1	Running	3 (4d22h ago)	35d
default	loki-promtail-7kjg5	0/1	Running	3 (4d22h ago)	35d
default	promtail-4fx4n	0/1	Running	3 (4d22h ago)	35d
default	prometheus-alertmanager-5f966448fb-rzh2m	2/2	Running	2 (20h ago)	25h
default	prometheus-server-6965b8c5db-9tgmj	2/2	Running	4 (20h ago)	25h
default	promatail-loki-0	1/1	Running	77 (20h ago)	36d
kubernetes-dashboard	kubernetes-dashboard-79b5779bf4-f7lbf	1/1	Running	46 (21h ago)	36d
litmus	chaos-litmus-auth-server-6957f4d6fb-j7hrd	1/1	Running	0	19h
litmus	chaos-litmus-server-5c5596fb65-8zhbq	1/1	Running	0	19h
litmus	event-tracker-5c9f554948-bjfx7	1/1	Running	0	19h
litmus	subscriber-5c7b67fbf4-4zrwx	1/1	Running	0	19h
litmus	chaos-operator-ce-54594c9cfd-4vmzh	1/1	Running	0	19h
litmus	podtato-head-1655216779-177467863	0/2	Completed	0	18h
litmus	podtato-head-1655216779-2174027101	0/2	Completed	0	18h
litmus	podtato-head-1655216779-3881841610	0/2	Completed	0	18h
litmus	podtato-head-1655216779-2721786719	0/2	Completed	0	18h
litmus	podtato-head-1655216779-3735051380	0/2	Completed	0	18h
default	promtail-s64zk	1/1	Running	2 (20h ago)	36d
litmus	custom-chaos-workflow-1655217040-1655221200-488439437	0/2	Pending	0	17h
default	loki2-0	1/1	Running	67 (20h ago)	36d
litmus	chaos-litmus-frontend-76b677fbff-dkv5b	1/1	Running	0	19h
default	loki-0	1/1	Running	73 (16h ago)	35d
litmus	pod-cpu-hog-end2p9--1-zjzfr	0/1	Completed	0	14h
litmus	pod-cpu-hog-95zfskzp-runner	0/1	Completed	0	14h
default	promtail2-jbr9q	1/1	Running	3 (20h ago)	36d
kube-system	traefik-55869fc94f-tjdvf	1/1	Running	77 (21h ago)	37d
default	my-release-influxdb-0	1/1	Running	0	20h
default	loki-fluent-bit-loki-w8dnb	1/1	Running	277 (20h ago)	35d
litmus	chaos-exporter-7c847d6fbd-gzc5r	1/1	Running	0	19h
litmus	workflow-controller-856d568f68-nfz6x	1/1	Running	0	19h
default	prometheus-node-exporter-p7dtz	1/1	Running	3 (4d22h ago)	35d
kube-system	svclb-traefik-swc7c	2/2	Running	6 (4d22h ago)	35d
litmus	mongo-0	1/1	Running	0	19h
kube-system	coredns-b67478c6d-mtz4s	1/1	Running	83 (21h ago)	37d
kube-system	metrics-server-6d67fc7f8c-cn69k	1/1	Running	71 (16h ago)	37d
litmus	chaos-litmus-mongo-0	1/1	Running	0	20h
litmus	litmusportal-frontend-6694d8977b-grqrl	0/1	CrashLoopBackOff	56 (3m10s ago)	19h
litmus	litmusportal-server-5d7c855f86-4mnvh	0/2	CrashLoopBackOff	106 (102s ago)	19h

Figure A.1: all pods

NAMESPACE	NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
default	kubernetes	ClusterIP	10.43.0.1	<none>	443/TCP	43d
kube-system	kube-dns	ClusterIP	10.43.0.10	<none>	53/UDP,53/TCP,9153/TCP	43d
kube-system	metrics-server	ClusterIP	10.43.149.96	<none>	443/TCP	43d
default	prometheus-node-exporter	ClusterIP	None	<none>	9100/TCP	43d
default	prometheus-kube-state-metrics	ClusterIP	10.43.232.50	<none>	8080/TCP	43d
default	prometheus-pushgateway	ClusterIP	10.43.183.16	<none>	9091/TCP	43d
default	prometheus-server	NodePort	10.43.116.209	<none>	80:31539/TCP	43d
default	prometheus-alertmanager	NodePort	10.43.50.144	<none>	80:31840/TCP	43d
kubernetes-dashboard	kubernetes-dashboard	ClusterIP	10.43.140.122	<none>	443/TCP	36d
kubernetes-dashboard	dashboard-metrics-scraper	ClusterIP	10.43.131.144	<none>	8000/TCP	36d
default	loki-headless	ClusterIP	None	<none>	3100/TCP	35d
default	loki	NodePort	10.43.191.130	<none>	3100:32328/TCP	35d
default	my-release-influxdb	NodePort	10.43.169.53	<none>	8086:30083/TCP,8088:32732/TCP	29d
default	grafana	NodePort	10.43.227.9	<none>	80:32238/TCP	43d
litmus	chaos-litmus-auth-server-service	NodePort	10.43.234.139	<none>	9003:30413/TCP,3030:32161/TCP	20h
litmus	chaos-litmus-frontend-service	NodePort	10.43.146.122	<none>	9091:32049/TCP	20h
litmus	chaos-litmus-headless-mongo	ClusterIP	10.43.131.47	<none>	27017/TCP	20h
litmus	chaos-litmus-mongo	ClusterIP	10.43.237.185	<none>	27017/TCP	20h
litmus	chaos-litmus-server-service	NodePort	10.43.178.124	<none>	9002:32747/TCP,8000:31768/TCP	20h
litmus	litmusportal-frontend-service	NodePort	10.43.238.47	<none>	9091:31873/TCP	19h
litmus	litmusportal-server-service	NodePort	10.43.125.158	<none>	9002:32584/TCP,9003:32765/TCP	19h
litmus	mongo-service	ClusterIP	10.43.64.107	<none>	27017/TCP	19h
litmus	workflow-controller-metrics	ClusterIP	10.43.253.52	<none>	9090/TCP	19h
litmus	chaos-exporter	ClusterIP	10.43.132.170	<none>	8080/TCP	19h
litmus	chaos-operator-metrics	ClusterIP	10.43.118.169	<none>	8383/TCP	19h
kube-system	traefik	LoadBalancer	10.43.70.26	192.168.253.134,192.168.253.135	80:31047/TCP,443:31276/TCP	43d

Figure A.2: all services on master

Figure A.5: Global infrastructure Dashboard

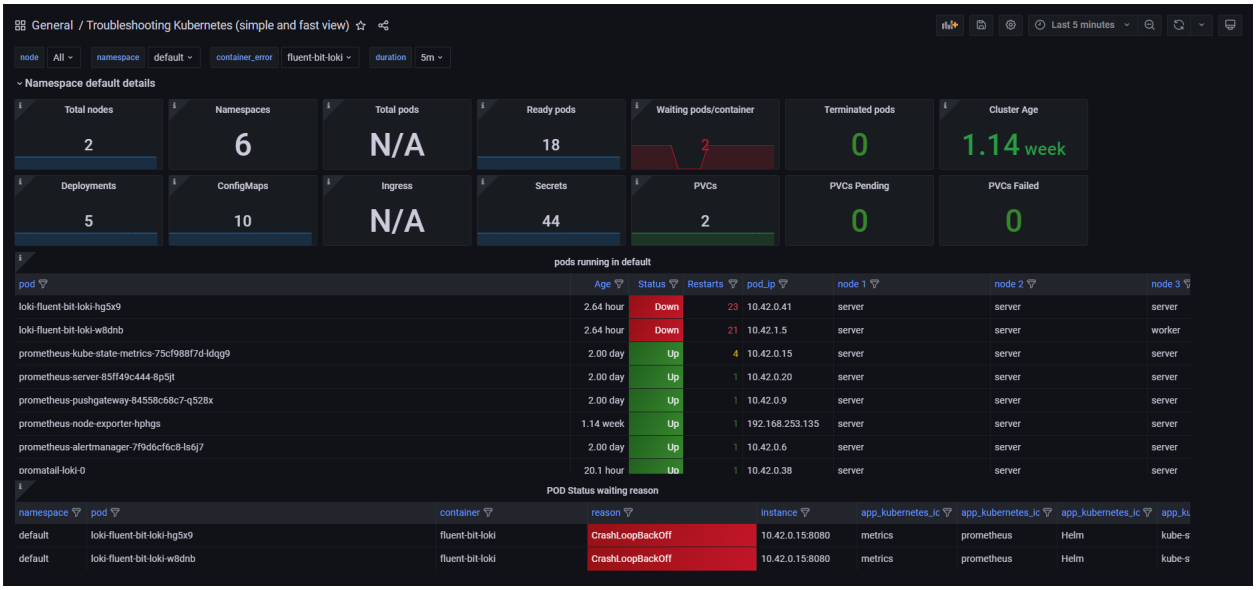


Figure A.6: Troubleshooting Kubernetes dashboard

Appendix B

Release 2: AiOps Workload

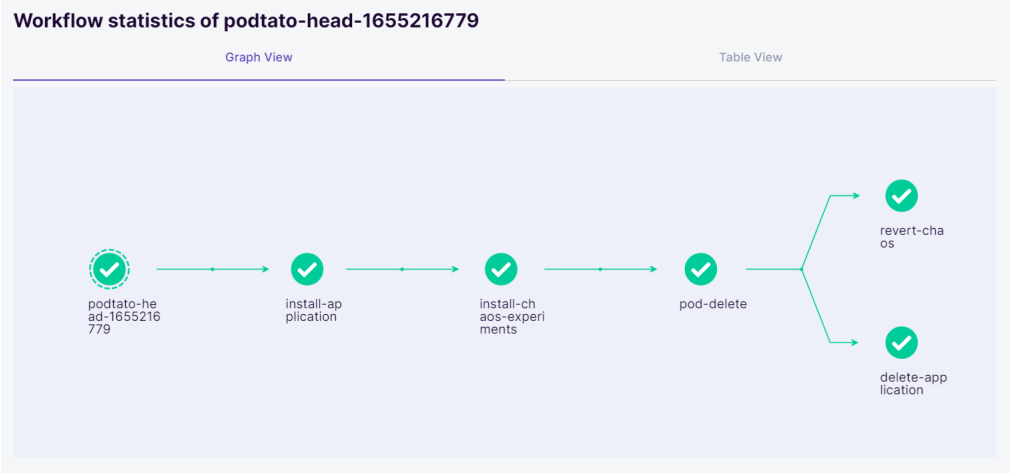


Figure B.1: full workflow process

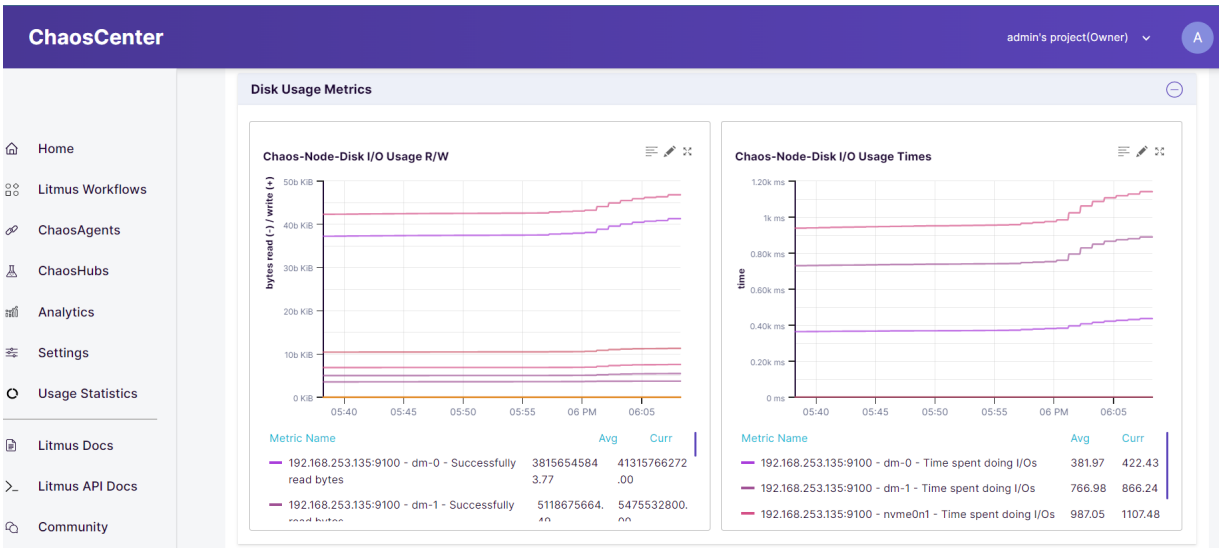


Figure B.2: Prometheus metrics in Litmus

Bibliography

[1] what-is-grafana-why-use-it-everything-you-should-know-about-it. https://www.scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/, 2020.

[2] history fluentbit. https://docs.fluentbit.io/manual/about/history, 2022.

[3] Grafana loki. https://grafana.com/oss/loki/, 2022.

[4] ooredoo graoup official website. https://www.ooredoo.com/en/l, 2022.

[5] theta mothod. https://www.statsmodels.org/devel/index.html, 2022.

[6] CNCF. an-introduction-to-metrics-monitoring-and-alerting. https://www.digitalocean.com/community/tutorials/an-introduction-to-metrics-monitoring-and-alerting, 2022.

[7] C. Community. cloud native approach. https://www.cncf.io/, 2019.

[8] C. Community. Principles of chaos engineerin. https://principlesofchaos.org/, 2019.

[9] O. community. Tripleo architecture. https://docs.openstack.org/tripleo-docs/latest/install/introduction/architecture.html, 2019.

[10] J. Herzen, F. Lässig, S. G. Piazzetta, T. Neuer, L. Tafti, G. Raille, T. V. Pottelbergh, M. Pasieka, A. Skrodzki, N. Huguenin, M. Dumonal,

J. Kościsz, D. Bader, F. Gusset, M. Benheddi, C. Williamson, M. Kosinski, M. Petrik, and G. Grosch. Darts: User-friendly modern machine learning for time series.

[11] A. Kumar. difference-between-online-batch-learning. https://vitalflux.com/difference-between-online-batch-learning/, 2022.

[12] official Anaconda website. Distrubution. https://www.anaconda.com/products/distribution, 2022.

[13] official helm website. The package manager for kubernetes. https://helm.sh/, 2022.

[14] official K3S website. Lightweight kubernetes. https://k3s.io/, 2022.

[15] official K8S website. Production-grade container orchestration. https://kubernetes.io/, 2022.

[16] official Minikube website. start. https://minikube.sigs.k8s.io/docs/start/, 2022.

[17] Scrum.org. sprint-planning. https://www.atlassian.com/agile/scrum/sprint-planning, 2020.

[18] I. Svetunkov. simpleforecastingmethods. https://openforecast.org/adam/simpleForecastingMethods.html, 2022.