DEVOPS EXPERTS
OUR KNOWLEDGE, YOUR FUTURE

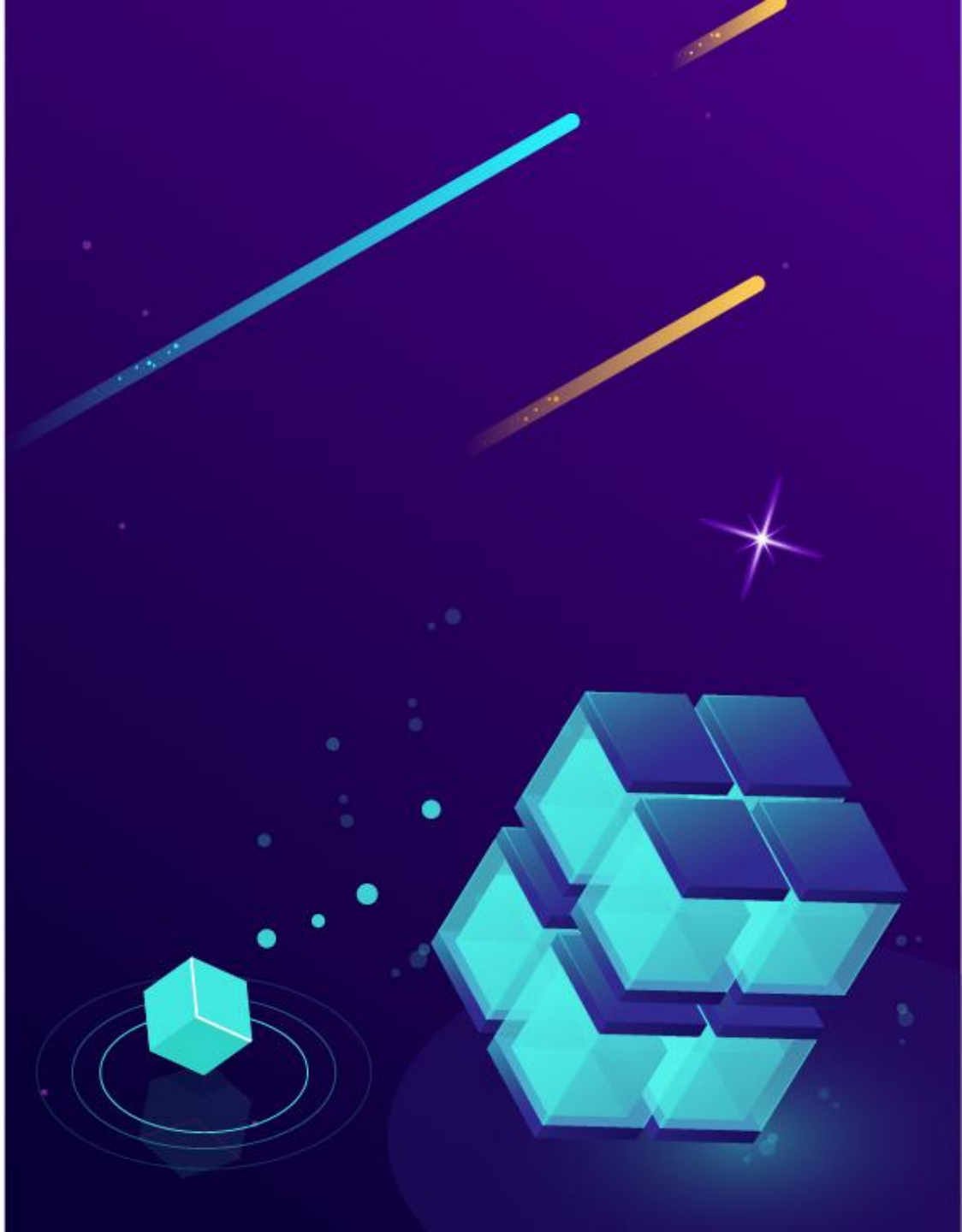
Software Architecture

CyberFort Nexus

COURSE PROJECT

Avraham Poupko

devopsexperts.co.il



CyberFort Nexus

Protect Your
Enterprise Network

General Requirements

CyberFort Nexus

- Background: You are an architect in a company whose expertise is cyber defense, launching their next gen product to a **comprehensive enterprise cyber defense solution**
- CyberFort Nexus will be a **detection system**, installed **on premises** on **servers**, and feeding of information from the **gateway** of the enterprise network
- The system collects all information from different **sensors** to a **central analysis center**
- It will provide the **Cyber Analysts** to the **SoC** (Security Operations Center):
  - Large amount of information from the sensors
  - Enable them to process it with the help of an automatic investigator
  - Send alerts one cyber attacks
- Enable the analysts with drill down to events and forensics
- The system will not provide mitigation or other actions in response to the detected attacks

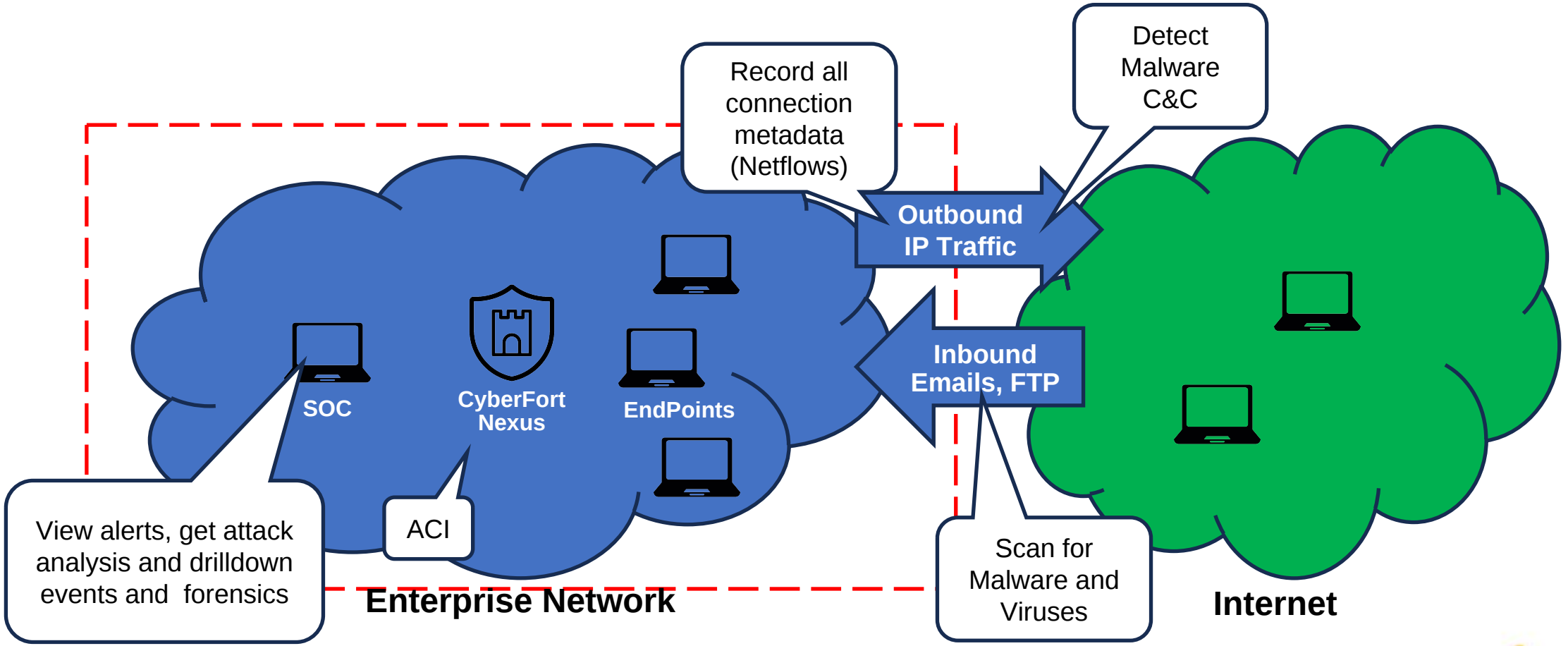
General Requirements – Cont.

CyberFort Nexus

- The system has 3 Parts
- **Threat Detection Sensors (all on the organization gateways)**
  - Inbound traffic: **File Analysis**
  - Outbound traffic: Malware Command and Control (**C&C**)
  - Duplex traffic: Netflow Collector for **Network Forensics**
- **Automatic Cyber Investigator (ACI) -** Automated cyber investigation to detect cyber campaigns
- **Investigation Portal -** To perform investigations (with the aid of the ACI), view alerts, drill down on events and network forensics

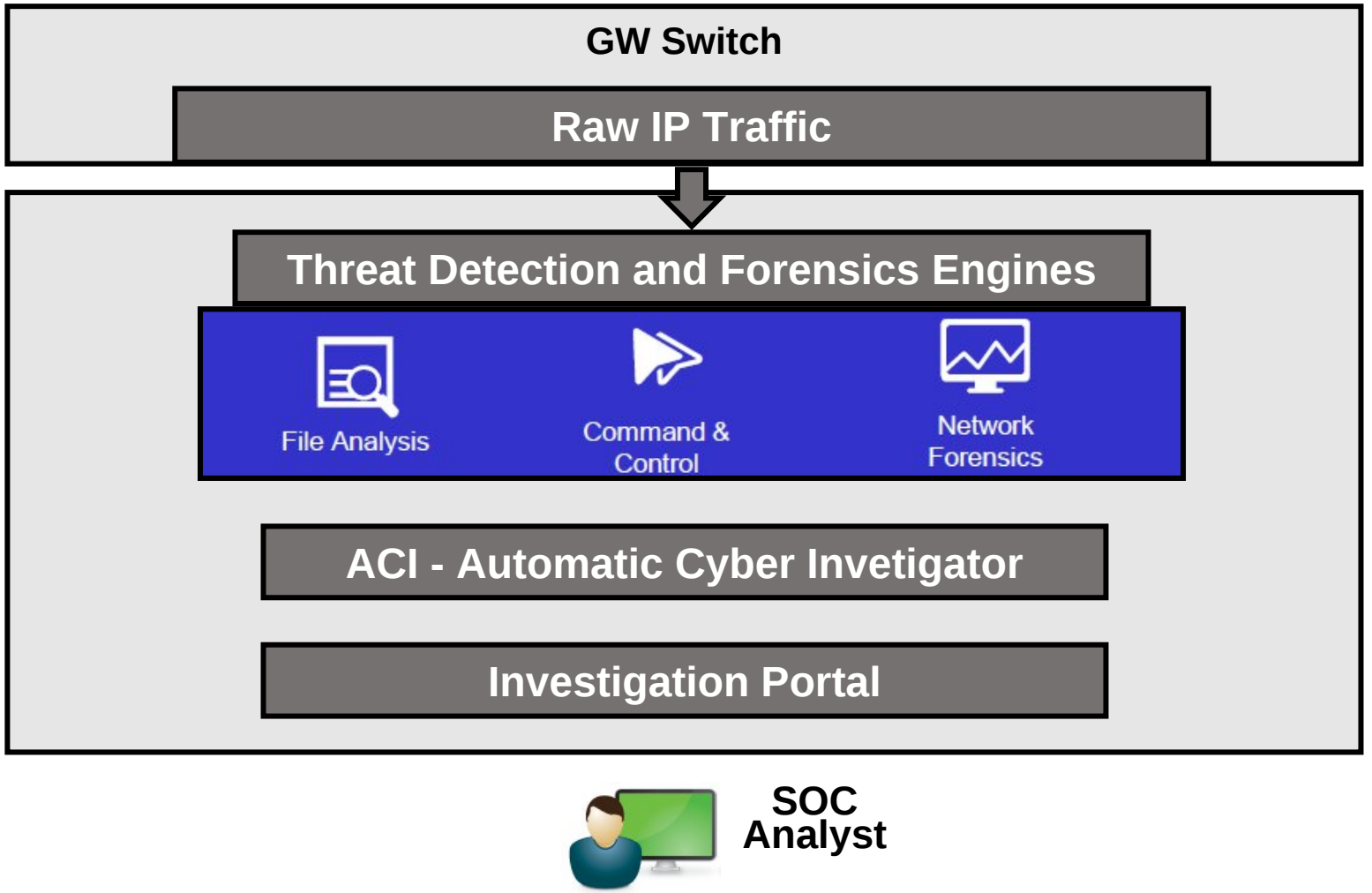
High Level Functionality

CyberFort Nexus



System Concept

CyberFort Nexus



Detection Engines

CyberFort Nexus

- **File Analysis**
  - A 3rd party software component that runs anti-virus engines to scan for malware and viruses in inbound files and emails (POP3 decoding)
  - REST API – send file path in json and retrieve a json report in the response
  - Generates alert when malware/virus is found
- **Command and Control**
  - Home brew
  - Runs ML logic on the metadata of raw packets to detect malware outgoing C&C traffic
  - Assume an ML algorithm is available as python code, no need to consider the training part
  - Generates alert when suspected C&C traffic is detected

Forensics Engine

CyberFort Nexus

- **Network Forensics**
  - Index all gateway traffic from the organization to the internet
  - IP metadata is indexed to search the traffic as a netflow – timestamp, Source+Dest IP, Source+Dest Port, and protocol (tcp/udp)
  - Data retention is 3 months
  - Assumption: 100 IP flows per minute per endpoint on average

- Assume the Organization Gateway switch can mirror all IP traffic to a probe to intercept it

ACI - Automatic Cyber Invetigator

CyberFort Nexus

- The “Brain” of the cyber defense system
- Gets all the data from all engines and performs automatic queries in order to alert on cyber incidents. Its sources are:
  - File Analysis – alerts malware or viruses detected in files/emails
  - C&C – Alerts on detection of outbound C&C traffic
  - Network Forensics – Forensics of IP metadata and traffic
- The output of ACI are alerts for the SOC analyst, appearing on their dashboard
- The investigation shows all relevant information in the portal and allow further manual investigation by the cyber analyst

Investigation Portal

CyberFort Nexus

- The organization’s cyber defense operation is managed by analysts in the SOC – the Security Operation Center. The analysts will work the CyberFort Nexus product
- The system will provide a cyber portal to the analysts in which they can:
  - View alerts from the file and C&C sensors
  - Investigate the network forensics data
  - Manage investigations which include a timeline of alerts and network recordings
  - The ACI also produces automatic investigations which the user can manage as well
  - Dashboard and reports of alerts and open investigations
- The organization may define up to 50 different users to the system, each with access only to their own investigations

Non Functional Requirements

CyberFort Nexus

- The solution should be 99.9% available
- The solution is installed on premises of the company
- The solution should be cloud native – based on containers / some VMs are OK
- Sizing should be for 3 sizes of company networks
  - Small – 200 endpoints (laptops/devices), 1 Gbps IP traffic peak
  - Medium – 1,000 endpoints, 5 Gbps IP traffic
  - XLarge – 10,000 endpoints, 50 Gbps IP traffic
- All components should be scalable appropriately
- All components should be monitored to a health dashboard and have a central log collection service

Submission Guidelines

CyberFort Nexus

- You are required to write a full High Level Design document of the system, based on the provided HLD template
- Start with assumptions and constraints
- Then write requirements
- Move to design - Identify components and flows, decide on persistency
- You can use any diagram tool. Examples: Draw.io / Diagrams.net, Excalidraw, Lucidchart, Power Point, MS Visio, Gliffy for Confluence, StarUML, WebSequenceDiagrams, Eraser.io, Archi.

Submission Milestones

CyberFort Nexus

- All students must submit the following milestones throughout the course
  - **Week 5** – **MS1**: Preliminary solution diagram (all submit + 3-4 volunteers to present)
  - **Week 9** – **MS2**: Requirements (all submit + 3 volunteers to present)
  - **Week 13** – **MS3**: High Level Design + Data + Security + Performance Diagrams (all submit + 3 volunteers to present)
  - **Final Submission-MS4** – Till the end of the course
  - To submit, please send to **ayrahampoupko9@gmail.com**

HLD

Feature Name

Document ID: AAA-BBB-12345

Revision History:

Rev.	Date	Author	Details

Abstract:

{3-4 lines about what the feature is, which system/version it is for, which sub-systems it involves.}

[Enter your writing here]

*{ A high-level design document is a high-level **description of the architecture, design, and components of a software system**.*

*The purpose of this document is to provide a **clear and concise understanding** of the system's design before more detailed design and implementation work begins.*

This document will describe the system architecture and organization, the main components and their relationships, the overall data flow and processing, and any external systems and dependencies.

The high-level design document serves as a foundation for more detailed design and implementation work and is used as a reference throughout the software development process.

It is also used to communicate the design to stakeholders, project team members, and other relevant parties, and to ensure that everyone has a clear understanding of the system's design and architecture}

1. General

1.1 Introduction

{In this section describe in high level the system's goals and objectives.}

[Enter your writing here]

1.2 Glossary, references

*{It's a good idea to create a glossary, dictionary with definitions of unusual phrases, product names etc. to create **ubiquitous language**. Moreover, it helps the reader to better understand not only documentation but also your business}*

[Enter your writing here]

2. Requirements

{This chapter should NOT include any design constraints or ideas – only the requirements, as interpreted by dev teams. All design issues must be under "High Level Design".

HLD Requirements are a transformation of Product requirements to technical domain.

Specific requirements should be crossed reference to other specifications (Product, Project etc.)

The breakdown below can be modified according to the specific subject of the HLD

1. *Software requirements are the specific **needs, goals, and constraints** that a software product must meet to be successful.*
2. *These requirements typically describe **what the software should do**, how it should perform, and the conditions under which it should operate.*
3. *They also define the features and functionality that the software must provide, as well as any limitations or restrictions that must be considered.*
4. *Software requirements can come from various sources, such as customers, end users, stakeholders, and project team members.*
5. *The process of defining, documenting, and refining software requirements is a critical step in the software development lifecycle and is essential for creating a high-quality and successful software product.*
6. ***Functional requirements***
    a. *describe what a software system must do and define the specific features and capabilities that the system must provide.*
    b. *These requirements specify what the software should accomplish and how it should behave in response to specific inputs and actions.*
    c. *Examples of functional requirements include user authentication, data input and output, calculations, and reporting.*
7. ***Non-functional requirements***
    a. *describe **how well the software system must perform** and define the constraints and quality attributes that must be met.*
    b. *These requirements specify the **characteristics and qualities** that are expected of the system, such as reliability, scalability, security, and usability.*

c. *Examples of non-functional requirements include response time, availability, and security standards.*

8. *Functional requirements are often described in terms of inputs, outputs, and specific user interactions with the software, while non-functional requirements are described in terms of performance and quality characteristics.*
9. *Both types of requirements are important for creating a software system that meets the needs and expectations of its users, and it's important to consider both functional and non-functional requirements when designing, building, and testing software.}*

[Enter your writing here]

1.1 Logical (System Functionality)

{Several items (number each paragraph!) which describe the feature highlights from a user standpoint - should be very similar the feature description in the product requirements specification

If relevant and does not exists in Product Spec – include use cases, state machines }

All requirements should be uniquely identifiable (different section number)

Careful attention should be given to organizing the requirements to maximize readability (short sentences, sub-paragraphs, etc.)

System functionality will break into epics and user story. The dev teams eventually break the user story into tasks}

[Enter your writing here]

1.2 User Workflow

{If relevant and possible include sample of screen shots}

[Enter your writing here]

1.3 Availability and Recovery

{Defines the proportion of time that the system is functional and working. It can be measured as a percentage. It should also state the recovery measures for any relevant failure}

[Enter your writing here]

1.4 Performance & Capacity Requirements

{Performance & capacity requirements should be expressed in measurements, not terms like "very fast". It should include responsiveness of a system to execute any action within a given time interval}

[Enter your writing here]

1.5 Scalability

{System ability to grow, by adding threads or HW. Address issues such as persistency and load balancing}

[Enter your writing here]

1.6 Security

{Address security in all relevant levels – OS, HW, Application level, interfaces (manly external), db}

[Enter your writing here]

1.7 Monitoring and Debugging

{define the monitoring capabilities of the feature. New Metrics, monitoring dashboards, log collection, and the ability of the system to provide any other information helpful for identifying and resolving issues when it fails to work correctly}

[Enter your writing here]

1.8 Deployment

{Describe the deployment environment - how this feature will be deployed, in the cloud native domain, how to deploy the container, what is the replica set, can this feature be deployed on any platforms}

[Enter your writing here]

1.9 Backward Compatibility

{Describe the required backward compatibility versions that should be supported}

[Enter your writing here]

3. High-Level Design

1.1 System Architecture

{This section should be filled for features which influence the system architecture: e.g. a new server / platform support, multi-site oriented features, etc.) Use the following views to describe your architecture:

1. *Process – run time behavior – how components communicate, concurrency, distribution, performance and scalability – use sequence diagrams, state machine diagrams, communication and activity diagrams*
2. *Development – implementation view – component diagram*
3. *Physical – deployment view – topology of software components, the physical layer, networking}*

[Enter your writing here]

1.2 Processes

{Define what the feature should do by identifying inputs, processes and outputs. Specify methods to be used. Interactions to other components & features should be specified or referred to.

This section will also identify the interfaces which will be elaborated in the detail design done by dev teams.

When writing this part, the following principles apply:

Break down the feature to user (or system) processes. Describe, for each process, the information flow between the user, and the subsystems involved. For each process, cover every aspect of the feature.

[Enter your writing here]

1.3 Design Rules and Principles

{Rules for constructing the architecture. Anything that the developers who will work on this feature need to keep in mind when they do low-level design that is not covered in the requirements or interfaces, in example – presentation layer should not direct access to persistency layer, all communication between components should be done over HTTPS. All data at reset should be encrypted.

This section can include guidance for the dev team – example – wherever possible, use asynch flows to improve parallelism}

[Enter your writing here]

1.4 Upgradability

{Describe how to upgrade older version of the system to support the deployment of this feature. What components needs to be upgrade, what kind of dependencies exits. Do we need to under go DB migration, or schema changes}

[Enter your writing here]

1.5 Assumptions and Dependencies

[Enter your writing here]

4. Time Estimation

{Insert the work plan time estimation for development after HLD writing }

Subsystem/team	Workdays

{Insert the work plan time estimation for development after this FRS review }

5. Limitations and Reservations

{This is a very important section. This is the place to put anything that the feature WON'T do that someone might easily think it SHOULD do. If this section is complete, there should be no surprises for the project manager or the client when they get the final product}

[Enter your writing here]

6. Risks

{For each risk, specify the resolution method (i.e., how to minimize the risk, and are there ways to minimize the "damage" of the risk if it materializes)}

[Enter your writing here]

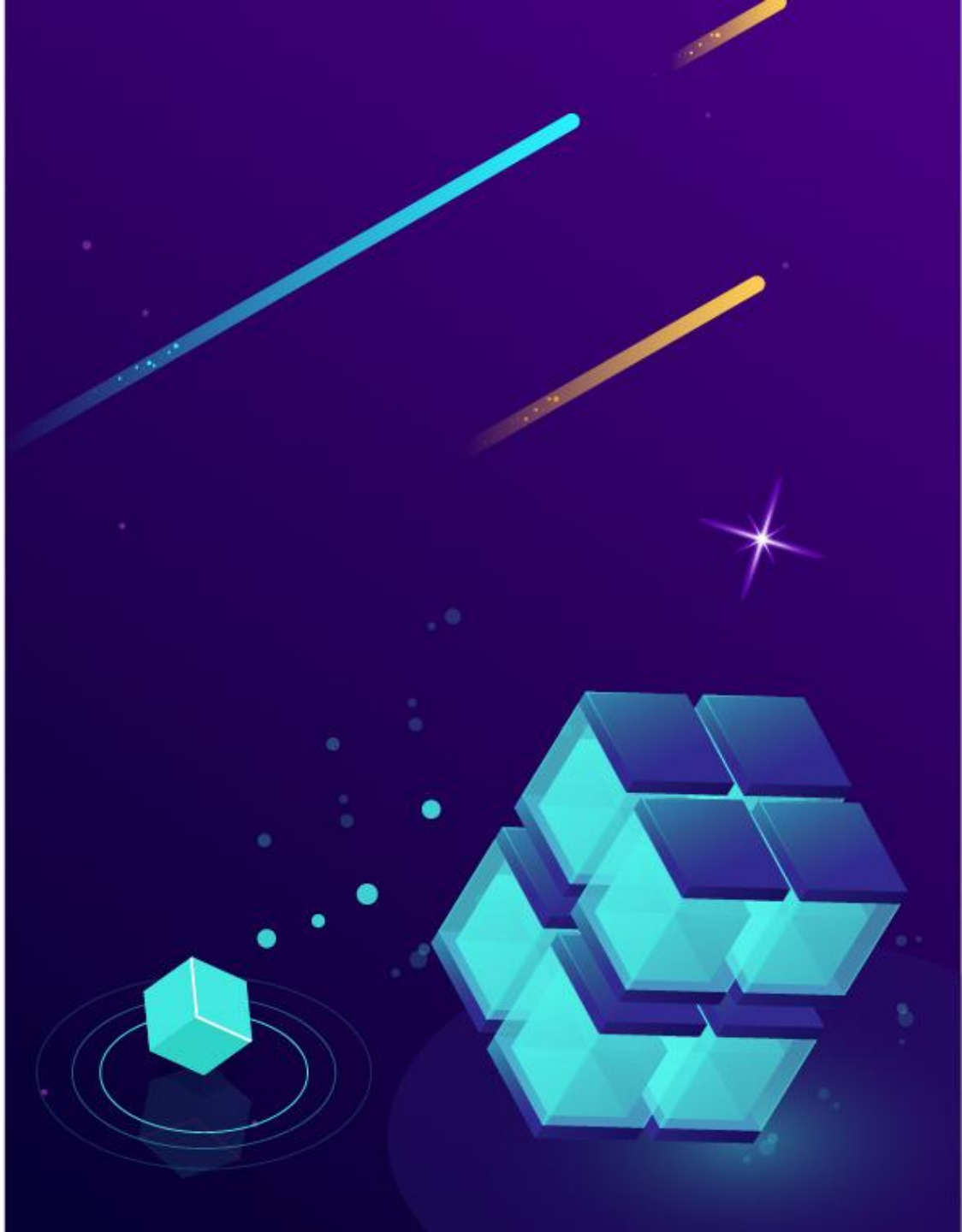
7. Open Issues

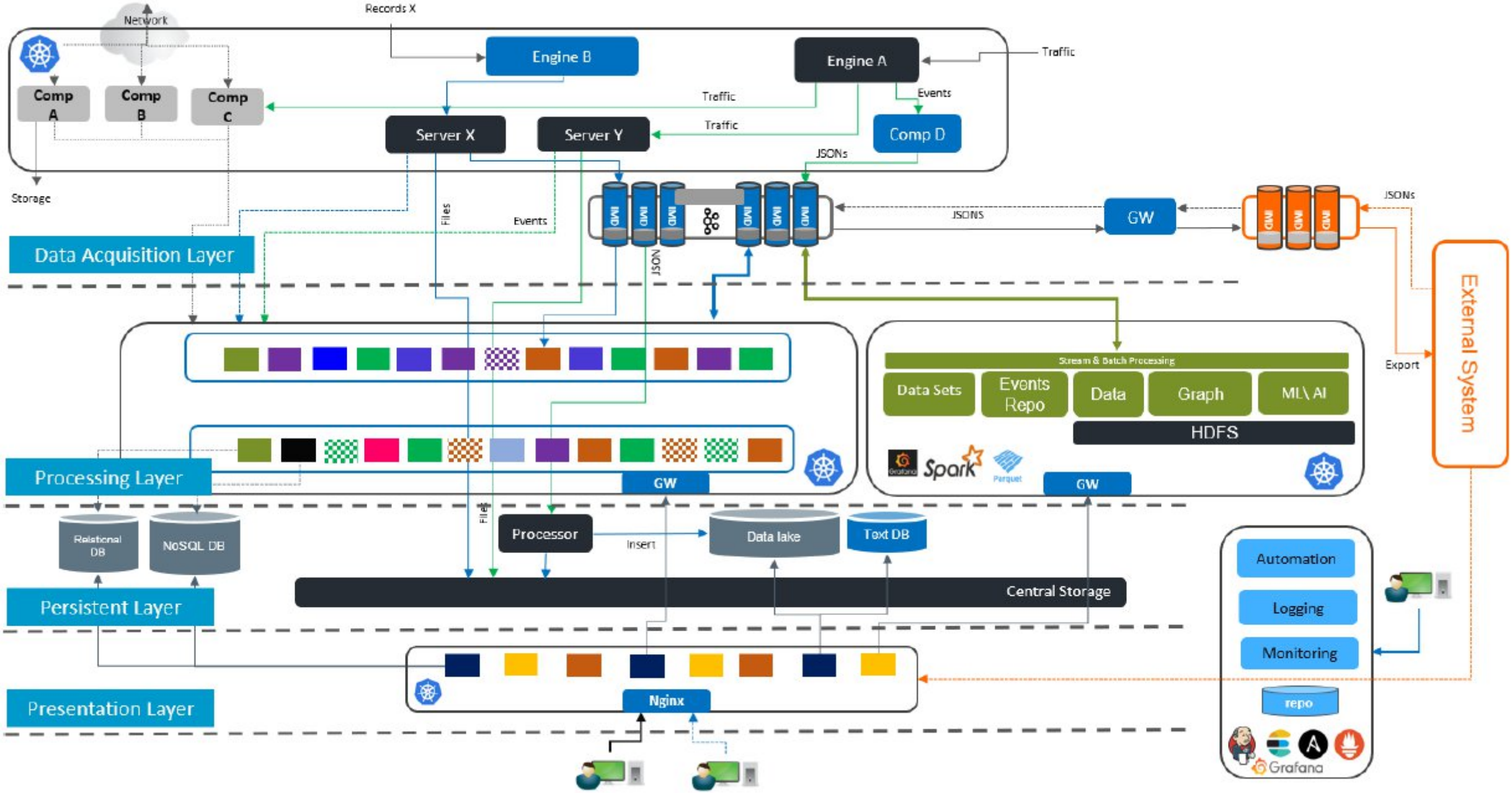
{This section should list every item that is not 100% closed in the document. Looking at this section should be enough to know what still needs to be decided or investigated. In the final FRS, this section should not exist. Any time you write in the sections above "DBS will check that..." or "AMS will perform a benchmark that" or "the text of the message will be decided", or "???", or TBD - you MUST add a paragraph to this section!

The final revision of the document should NOT include this section!}

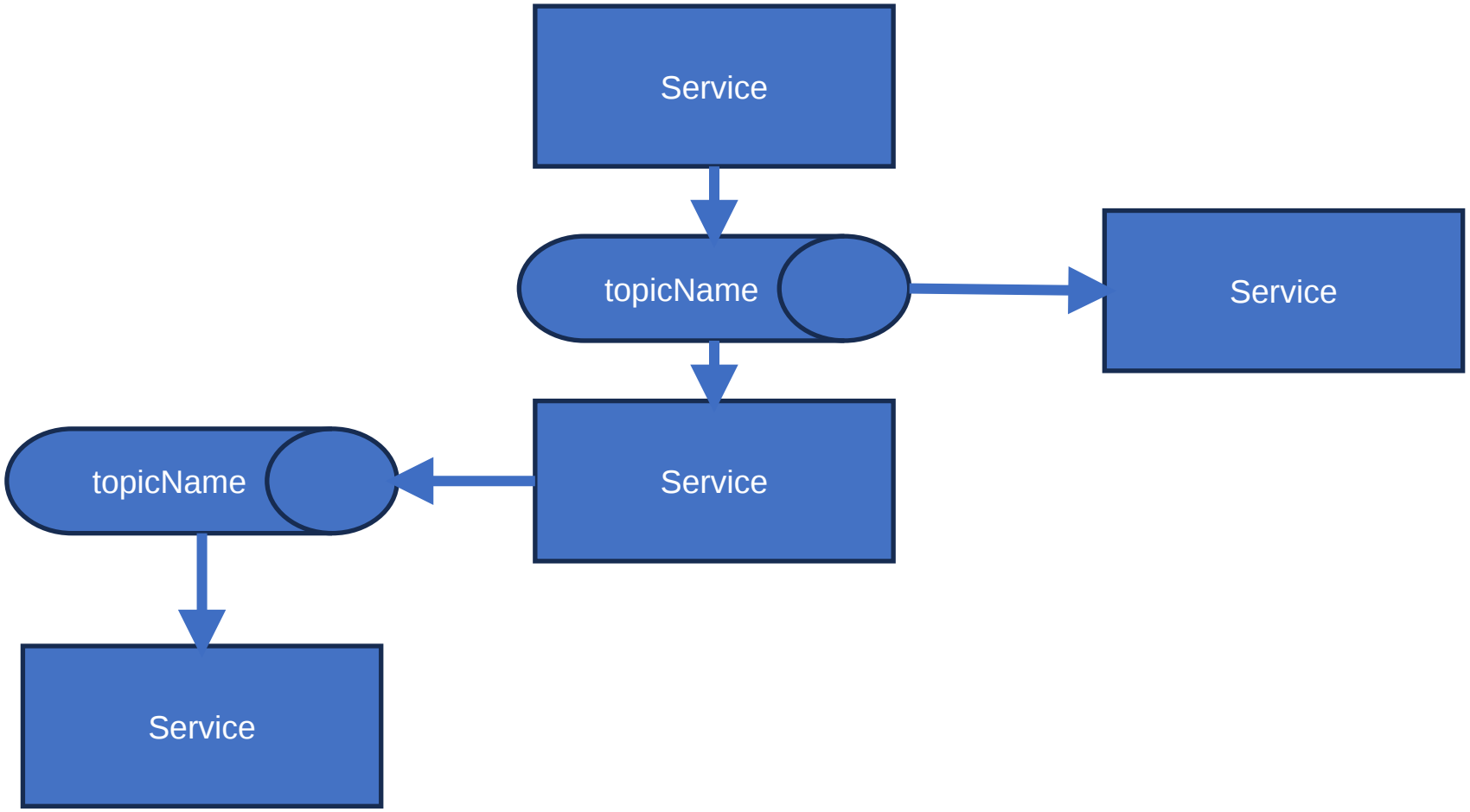
No.	Description	Subsystem	Responsible

HLD Diagram Examples




Network
Records X
Engine B
Engine A
Traffic
Comp A
Comp B
Comp C
Traffic
Events
Server X
Server Y
Traffic
Comp D
JSONs
Storage
Files
Events
IMD
JSONS
GW
JSONs
JSON
Data Acquisition Layer
External System
Export
Stream & Batch Processing
Data Sets
Events Repo
Data
Graph
ML\ AI
HDFS
Spark
Parquet
GW
GW
Processing Layer
Files
Processor
Insert
Data lake
Text DB
Relational DB
NoSQL DB
Central Storage
Persistent Layer
Automation
Logging
Monitoring
repo
Grafana
Nginx
Presentation Layer

Messaging Topics Flow Example



Ready ? Set GO!