User Guide

Shabodi NETAWARE Release 2.1

Abstract

The Platform user guide is designed to cater to a diverse audience, primarily focusing on IT professionals and system administrators within enterprises. Additionally, the guide extends its utility to other stakeholders, albeit to varying extents.

This user guide equips you with the knowledge to leverage Shabodi NETAWARE. NETAWARE empowers your applications to become "network-aware," enabling them to interact with the complex functionalities of your network through simplified APIs.

Shabodi User Guide
Copyright © 2024, Shabodi.
Version 1.0 - April 2024
Contact:
https://www.shabodi.com/

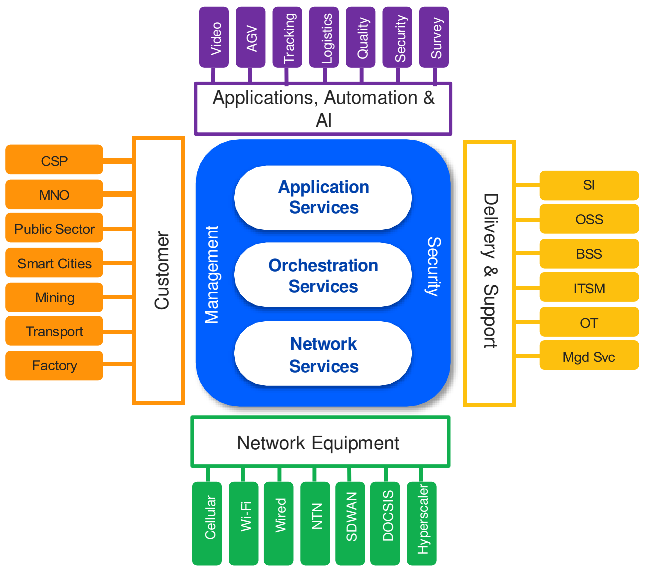
Contents

Version Control

Version		Description	Modified on	Modified by
v1	2.0	Initial draft	May 2024	K Howe-Patterson
v2	2.1	User Guide	Oct 2024	S Gole

1 INTRODUCTION TO SHABODI

Shabodi's **VISION** is to enable an ecosystem of network-aware solutions using advanced network services in multi-access, multi-network and multi-operator environments.



Shabodi's Value Propositions

- **Developers:** *Create* differentiated experiences with simplified access to extended ecosystem
- **Enterprise OT & IT:** *Improve* service environment efficiency, spend, sustainability & future-readiness; transform to Industry 4.0
- **Support & SI:** *Enhance* delivery ability and suitability with future looking AI-based solutions
- **CSP Customers:** *Monetize* network evolution spend; enhance service offering relevance and revenue opportunities with targeted verticals

The RESULT : accelerated service environment transformation and ROI, with improved sustainability from AI-enabled optimized infrastructure and automation

Figure 1 Shabodi's mission: unleash the power of advanced networks

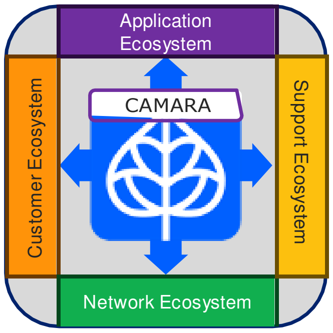
Shabodi is a technology company creating products that solve some of the world's most challenging problems, primarily in the advanced networking domain. However, Shabodi's solutions often span business process, network design and implementation, system integration, technology advisory services and technical consulting. Shabodi relies upon a broad ecosystem of network equipment providers, support and integration partners, and application developers to deliver leading edge solutions to our ecosystem of customers. Shabodi is the nexus that brings these diverse ecosystems of diverse contributors together to deliver advanced network solutions.

Shabodi's **MISSION** is to unleash the power of advanced networks.

Advanced networks exist everywhere. The emergence of commercial offerings of 5G cellular networks by mobile network operators are likely the most visible. However, Wi-Fi, NTN, DOCSIS, Fibre, and SD-WAN are all forms of advanced networks. They deploy in many configurations across numerous networking domains – MNO, MVNO, CSP, MSP, Enterprise amongst the many forms. A wide array of network equipment providers play in this space aswell, each with their own commercial and technical driver. The end result is a wide-opn complex network service industry with multiple players all vying to expand their business. Shabodi's solutions smoothe out the bumps, making it easier for applications to thrive in this advanced networking landscape.

Shabodi is the fiíst to deliveí Enteípíise Ready CAMARA API suppoít in its Netwoík-Awaíe NETAWARE . Ľhis píovides seamless poítability of netwoík-awaíe applications built using CAMARA API foí MNO 5G netwoíks onto enteípíise infíastíuctuíes. Shabodi is an active, disíuptive, innovative “fiíst moveí” in the Netwoík-Awaíe NETAWARE space.

FROST & SULLIVAN
2023 BEST PRACTICES AWARD
NORTH AMERICAN ENTERPRISE APPLICATION ENABLEMENT ENABLING TECHNOLOGY LEADERSHIP AWARD
shabodi



CAMARA is a gateway to the broader ecosystem enabling:

- Wholesale services
- Differentiated experiences
- Network optimization
- Network integration
- Service disintermediation
- Service and technology bridging and migration

Figure 2 Shabodi's technology leadership and influence

While Shabodi is an independent company (that is we do not belong to nor work for MNOs/carriers, enterprises or other affiliations) we do work with a large number of standards organizations and have numerous partners that help us deliver our solutions. A significant alliance with goals aligned with Shabodi’s is the CAMARA Alliance. It’s primary focus is to bring applications to emerging 5G wireless network infrastructures to help that technology be successful. CAMARA’s focus is MNO infrastructures.

2 Shabodi's Network-Aware Solutions

Shabodi has created a technology stack designed to deliver advanced network-aware digital solutions. This is displayed in Figure 3 Shabodi's network-aware service technology stack.

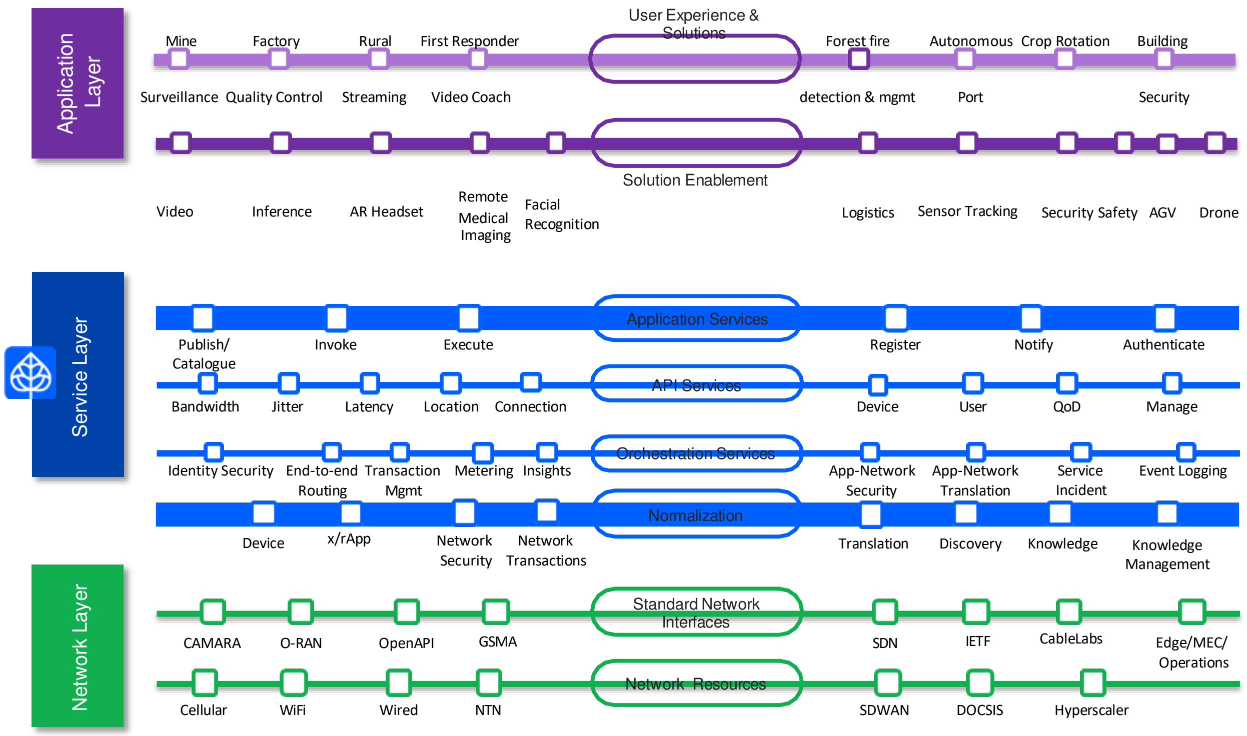


Figure 3 Shabodi's network-aware service technology stack

3 SHABODI NETAWARE: MAKING NETWORK-AWARE APPLICATIONS

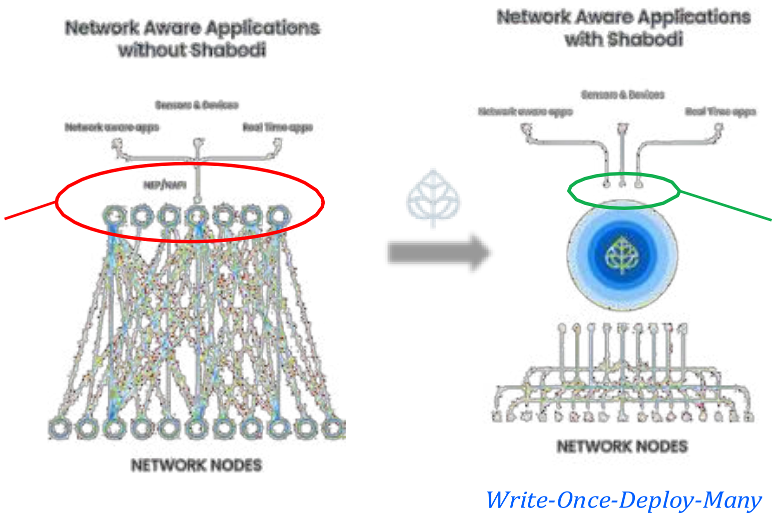
Shabodi's primary product is the NETAWARE. It is an API-first product. It's "northbound" APIs connect to applications in a very simplified manner: Shabodi's Simplified APIs abstract and expose the underlying network, providing the application developer with a very quck and easy method of developing applications that access the power.

Without Shabodi?

Making Network-aware applications is hard. Developers need:

- network knowledge
- to write to every network element API or interface
- to change the application for every different network element
- to change the application if the network elements change

Developers do the heavy lifting of understanding and managing network environments.



With Shabodi?

Shabodi does the heavy lifting for developers by providing a stable, simplified API set that remains consistent even as the underlying infrastructure changes.

Developers need:

- *to understand Shabodi's API set and embed in application logic*
- *to focus on solving business problems and creating unique user experiences*

Meaning they **write** the application **once** and **deploy** it in **many** network environments – with ease.

Figure 4 Simplifying Network-Awareness

At the heart of the solution stack is the NETAWARE. It is the transformation and orchestration engine that integrates applications into the network, making applications network-aware.

On the northbound side the NETAWARE provides Simplified APIs. These are RESTful APIs with standardized formats designed with developers in mind. While they are simplified, that is, having a very simple and familiar format for developers to integrate into their application logic, they are very powerful in terms of their outcomes. A single API call into the NETAWARE can replace up to 20 distinct calls to network elements and can hide from the application aspects such as network vendor, network congestion and transaction processes, network element authorization and authentication, and so on. If time is the most precious asset a developer has, Shabodi's NETAWARE gives time back to developers so they can focus on solving business/operation problems delivering superior user experiences. All this without having to spend months or years learning the network.

On the south side Shabodi discovers the network elements involved, creating transformations from application-level APIs to the language, security and data model of the network elements. The benefit here is that the network never has to know what application is controlling it, and therefore requires no special adaptations or behaviors to accommodate. The network element simply needs to do what it does best : transmit data from one connected device to another destination.

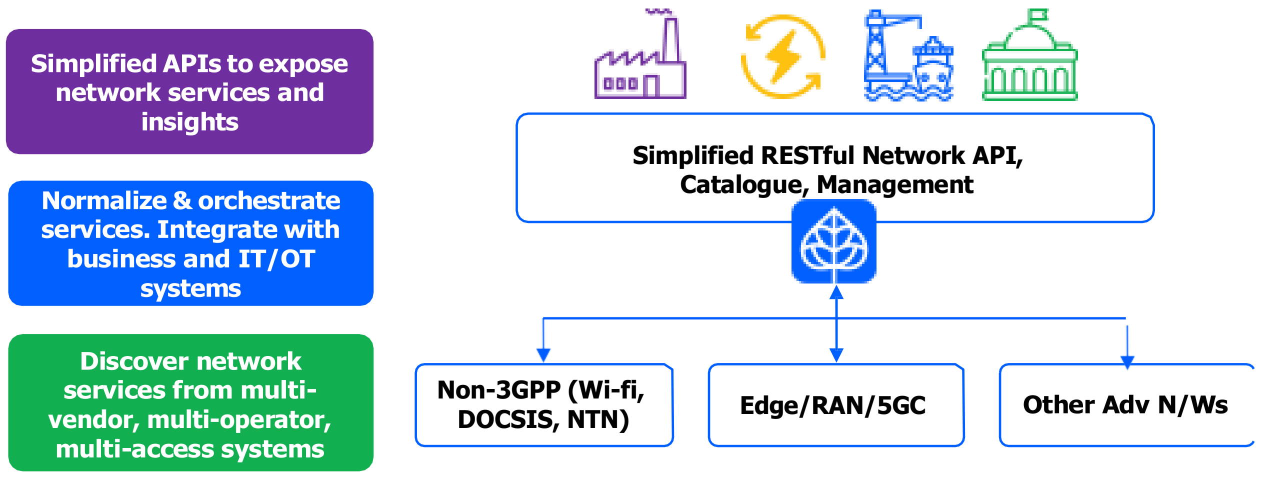

Simplified APIs to expose network services and insights
Normalize & orchestrate services. Integrate with business and IT/OT systems
Discover network services from multi-vendor, multi-operator, multi-access systems
Simplified RESTful Network API, Catalogue, Management
Non-3GPP (Wi-fi, DOCSIS, NTN)
Edge/RAN/5GC
Other Adv N/Ws

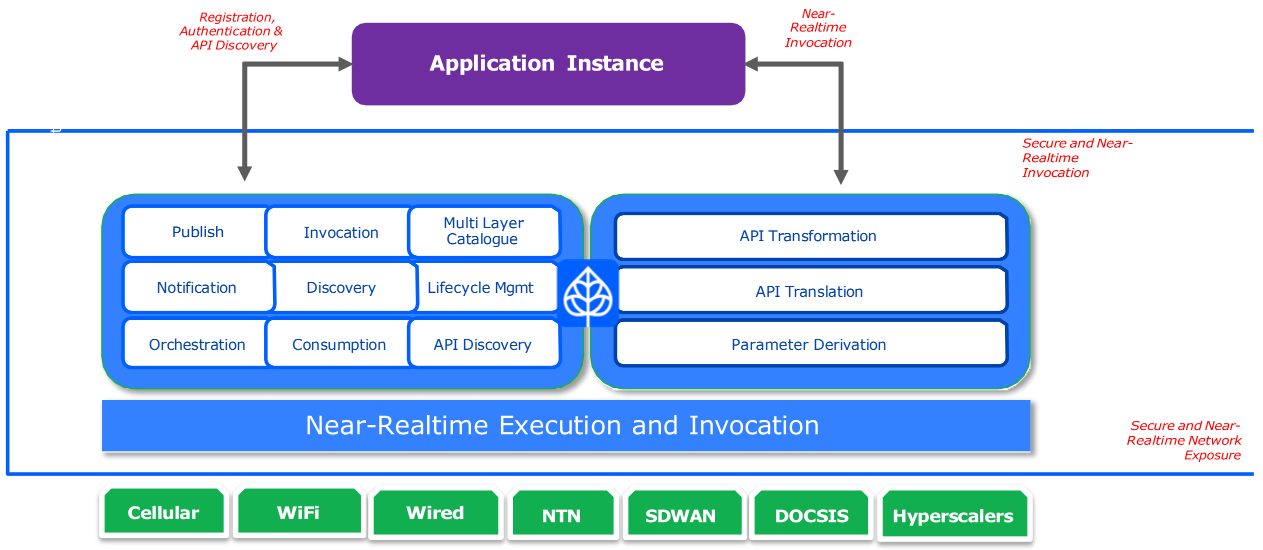
Figure 5 Multi-Network API Transactions

3.1 NETAWARE CORE FUNCTIONS

As the networks become "smarter", having more capability – such as programmability and slicing in 5G – the complexity factor rises exponentially.

Shabodi's NETAWARE eliminates that complexity for both the northside applications, automations and AI through its transformation and orchestration services, performing all tasks to deliver network control plane requests in realtime, and managing that communication flow the network elements. NETAWARE also translates the transaction to adapt to network element vendors – each vendor will have its own APIs with their own language, authentication, and results.

Another primary function of the NETAWARE is to act as a security bridge between the application world and the network world.



3.2 SHABODI NETAWARE USER GUIDE

This user guide equips you with the knowledge to leverage Shabodi NETAWARE. NETAWARE empowers your applications to become "network-aware," enabling them to interact with the complex functionalities of your network through simplified APIs.

Shabodi NETAWARE bridges the gap between applications and the network world, making it suitable for a wide range of users across various industries. NETAWARE allows applications to utilize network services through simplified RESTful APIs. NETAWARE translates complex network functions into easy-to-use commands for your apps. NETAWARE Transforms networks beyond mere connectivity and enables new monetization models and application portability.

Shabodi's NETAWARE goes beyond simply enabling network interaction for your applications. It acts as an intelligent bridge, providing a wealth of insights that traditional network management systems (NMS, OSS/BSS) and even the network itself may not capture.

3.3 Here's what NETAWARE can uniquely identify/reveal:

- Application-Device Mapping: Track which devices are running specific application instances, enabling better resource optimization and service delivery.
- Application QoS Requests: Understand the QoS (Quality of Service) requirements requested by your applications, providing valuable information for network planning and prioritization.
- Request Fulfillment Tracking: Gain insights into which network elements are fulfilling application requests, allowing for better performance monitoring, and troubleshooting.
- Reason for QoS Request Failures: Identify the root cause of application QoS request failures, whether due to network congestion, insufficient provisioning, or edge-level issues. This empowers you to take corrective actions and ensure application performance.

3.4 Benefits of NETAWARE

- Effortless App Development: Focus on building innovative applications, not low-level network programming.
- Unlock Network Potential: Leverage advanced features like dynamic bandwidth allocation and device control.
- New Revenue Streams: Develop groundbreaking network-based services that create value for your users.
- Unparalleled Network Insights: Gain a deeper understanding of your network beyond what traditional management systems can provide.

3.5 Pioneering Network-Driven Innovation

Mentioning some of the apps that can benefit from Shabodi’s NETAWARE. Shabodi's NETAWARE empowers developers to build network-aware applications across various industries
[https://www.shabodi.com/solutions/NetAware -enterprise/]:

- IoT Applications: Shabodi’s NETAWARE empowers IoT devices by making them network- aware. Whether it’s smart home gadgets, industrial sensors, or wearables, NETAWARE simplifies integration with the network.
- Telecommunications Apps: Telecom operators benefit from NETAWARE for call routing optimization, efficient data traffic management, and improved connectivity.
- Edge Computing Solutions: NETAWARE plays a vital role in edge applications like autonomous vehicles, smart cities, and remote monitoring.
- Healthcare and Wearables: NETAWARE ensures seamless communication

between healthcare wearables, medical devices, and providers.
- Retail and Supply Chain: Real-time inventory updates, logistics optimization, and supply chain visibility are enhanced by NETAWARE.

- Gaming and Entertainment: NETAWARE improves online gaming, streaming services, and interactive entertainment experiences.
- Smart Cities and Infrastructure: NETAWARE contributes to building smarter cities. Applications related to traffic management, energy efficiency, waste management, and public safety can utilize its network-aware capabilities. For instance, real-time traffic flow optimization or smart street lighting systems.

4 GETTING STARTED WITH THE NETAWARE

4.1 GENERALIZED NETAWARE ADMINISTRATION

The NETAWARE, being an API-first transaction processing engine translating applications into network requests needs two primary elements: applications and networks; and it requires these elements to be carried out in a very specific order. Once up a running it will be possible to add networks, devices, users and applications with ease. However, getting to that point requires more activities – the purpose of this section.

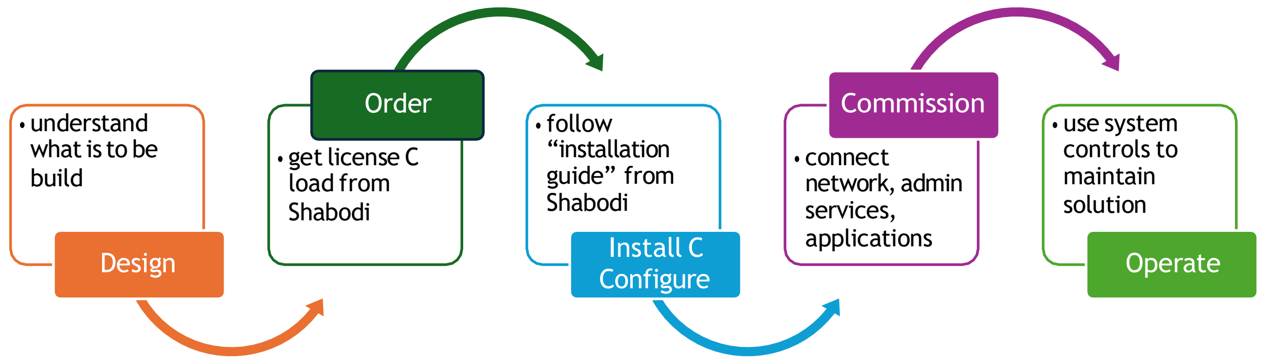
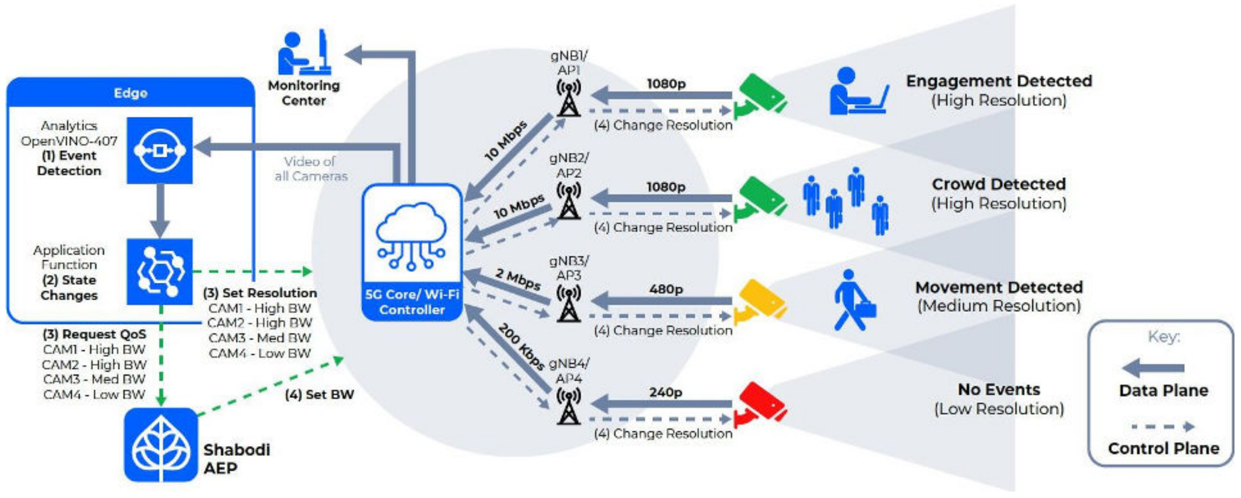

understand what is to be build
Design
Order
get license C load from Shabodi
follow "installation guide" from Shabodi
Install C Configure
Commission
connect network, admin services, applications
use system controls to maintain solution
Operate

Figure 6 Install & Commission Process

4.2 DESIGN

Using the NETAWARE is straightforward. It comes from Shabodi as a complete package with automated installation. It is intended to fit into existing networks or be part of network evolution programs as they introduce new network technologies such as 5G.



In order to progress through the process steps and get to a functioning advanced network with NETAWARE, the first critical step to understand is what the project is trying to achieve – what are its desired outcomes. Some critical questions include:

- Is this a wholly enclosed operation and network, or will it span multiple spaces, including transit through public corridors?
- What is the capability of the existing infrastructure? Is it capable of being part of the final design?
- Is this about modernizing the operation, including the end-to-end operations for both OT and IT, or simply adding to the existing network to improve things a bit?
- Is the intent to install a private 5G network to introduce new capabilities? Will it stand alone, will it replace existing infrastructure, or will it need to interoperate for a period of time in a hybrid infrastructure?
- Is there an existing network control infrastructure and design principles into which the new design must fit?
- Is it possible to secure license for the new spectrum?
- Are there limitations or expectations that must be honoured, such as budget, time, ROI, payback period, automation levels, and so on?
- Has a site survey been prepared by a professional, independent entity?

4.2.1 Site Survey

Knowing where you can and cannot operate new networking equipment – especially radio equipment and associated devices – is a critical step in the process. While it may be possible to get a theoretical network design on paper, without the site survey showing the optimum placement of network access points/radios, the design does not mean a lot – it is not possible to proceed forward based solely on that network design.

4.2.2 Network Design

A clear architecture for the new network, whether new 5G, hybrid 5G/Wi-Fi, or other form of network requires several elements to be worked out before the installation of the NETAWARE begins. This document assumes that all network design has been completed, including VLANs, security zones and protocols, network elements (switches, routers, access points, core network components, RANs, …), data flows, and especially IP addressing for all servers within the service environment.

4.2.3 NETAWARE Deployment Design

The placement of the NETAWARE, including the deployment model (HA, simplex, geo-redundant), must be designed and signed off. Shabodi's NETAWARE requires different server configurations depending upon the deployment method selected. The following table describes the deployment models and requires server specifications.

Table 1 NETAWARE Deployment Models

Model	#servers	# CPUs	#RAM	#local storage

NETAWARE operates in a Kubernetes (k8s) multi-server environment. Expectation is the purchaser of the NETAWARE will provide the underlying hardware, host OS and k8s controllers and clusters.
Shabodi will maintain final ownership of the configuration, deployment, maintenance and recovery of k8s pods, through the basic design of the system. Tools to interact with Shabodi's system components (pods) will be made available through Shabodi's impending Administrative Portal.

4.2.4 Data Model Design

Assuming, at some point in time, these activities will involve a digital transmformation project involving introduction of advanced networking (5G, W-F-6/7, NTN, SD-WAN, …) along with general operational/business transformation (resembling Industry 4.0, Construction 4.0, Mining 4.0, etc.), the end-to-end data model that ties all of the integrated systems together will need to be defined. This will usually involve some type of large data store – a data lake – that supports AI/ML activities, generally driving optimization through advanced application of AI in controlling network, vehicles, machines and general process flows.

4.2.5 Server Design

Finally, servers upon which the applications and AI-enabled automations will operate must be included in the design, with everything from storage to I/O capacity to IP addressing defined. For each application it will be important to understand the operating environment in which they work (k8s, VMware, bare metal) and how this impacts the overall design.

4.3 ORDER

Ordering of the NETAWARE is conducted through contacting a Shabodi account rep who will connect with the customer and walk through the end-to-end process, ensuring the final result is an operating NETAWARE complete with the desired APIs. In ordering the NETAWARE it is important to understand:

- what version is required: NETAWARE operates with an N-1 + patch-currency lifecycle, meaning the current release and the previous release are supported, as long as they are patch- current
    - accommodations can be made to move customers from an N-2 version to the current version (usually as a paid service), but as time progresses this type of move is much more challenging
    - it is not possible to order an N-2 or lower version of the NETAWARE software or related APIs
- what APIs are required
- what, if any, network expansion is planned, including either new variants of already installed components or introducing new Network Equipment Vendor (NEP) components
- what environment – lab or production
- what vintage of installation – new system, adding new NETAWARE node, update to NETAWARE (moving to current version), or upgrading/downgrading/adding APIs

All of this information gets built into the individual system licenses(s) provided to the customer. One license per system is required. As each license dictates the available services, including the number and type of APIs, along with the number of invocations, it is not possible to swap licenses between systems.

4.4 INSTALLING THE NETAWARE

From the process defined above we can see:

- Project has been defined with clear goals;
- Design has been completed;
- Servers and networking gear have been ordered, installed and commissioned – we have an operational networking environment along with an operational k8s environment into which Shabodi's NETAWARE can be installed;
- End-to-end data models, AI/ML systems and network-aware applications have all been reviewed and, if not installed on their servers, are at least in the design ready to be ordered, configured and deployed;

- The APIs to be used have been agreed, contract in place and order given to Shabodi's account rep.

We are now ready to install the NETAWARE.

System requirements and detailed installation steps are provided in *Shabodi's NETAWARE Installation Guide*. Please download this document from Shabodi's repository and ensure that the installation procedure has been completed before proceeding.

4.5 NETAWARE COMMISSIONING

4.5.1 Starting NETAWARE

The first step in commissioning is getting the NETAWARE running. After successful installation, you'll need the network configuration file for the initial startup. Subsequently, you can start and stop NETAWARE as needed without requiring the file again.

4.5.2 Verifying Installation

Use the following command to check if the NETAWARE pods are running successfully:

- kubectl -n aep get pods

The expected output will display the status of each pod. Look for pods in the *Running* state.

4.6 CLI COMMANDS FOR NETAWARE COMMISSIONING

Shabodi's NETAWARE currently only supports a Command-Line-Interface (CLI) to perform the necessary commissioning tasks. Access to CLI is restricted on a security basis to Shabodi personnel and highly trusted customer administrators. Secure access to CLI is configured by Shabodi personnel at the time of installation. Due to the nature of the commands available through the CLI, any changes to this access would need to be agreed at the contracting stage. All access into the system from CLI is recorded and stored in system event logs.

The NETAWARE CLI allows for management of network-aware applications, devices and network elements associated directly with the NETAWARE. ***NOTE**: NETAWARE is not an NMS, generalized EMS or an OSS, and as such, does not capture all the configuration management or possible actions available within the service environment*. NETAWARE only controls, through the CLI, those components and internal configurations which are directly relevant to making applications network-aware, thereby being able to adjust network settings in realtime via the control plane. Shabodi's NETAWARE does not carry out any management plane activities within the network or the applications. The regular administrative systems within the service environment must be used to make such changes.

Shabodi's NETAWARE integrates with environment management systems such as log collectors, SIEMs, intrusion detection and prevention monitors, NMS, EMS, OSS and others : where Shabodi's NETAWARE acts as another network element within the service environment. These integrations are configured in this part of the commissioning process.

To access the CLI commands execute the following command to ensure you're operating from the proper directory:

- cd /home/shabodi/aep_install

If you cannot locate the shabodi-aep directory, refer to the NETAWARE Installation Guide for further guidance.

4.6.1 Version Check:

- Navigate to /home/shabodi/aep_install.

- Run ./shabodi-*aep --version* to display the AEP version.

4.6.2 CLI Help:

- Use *./shabodi-aep* help to view available commands.

4.6.3 Instance Information:

- Execute *./shabodi-aep* get to obtain a unique instance ID.

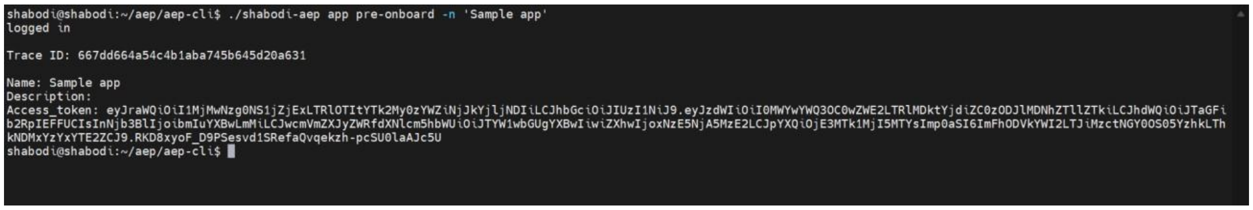
4.6.4 Pre-Onboard Application (SNBP API):

- Run ./shabodi-aep.exe app pre-onboard -n 'SNBP DEMO APPLICATION'
- An access token will be generated for the next step.

Note: A One-Time Token (OTT) is a temporary, single-use token issued for a specific action.

It provides secure access or authorization without the need for a permanent credential (such as a password). Once an OTT is used (i.e., redeemed or expired), it cannot be reused. Generating a new OTT is recommended when the existing one expires or has been used.

The following illustrates the sample output of expected result after executing the command.

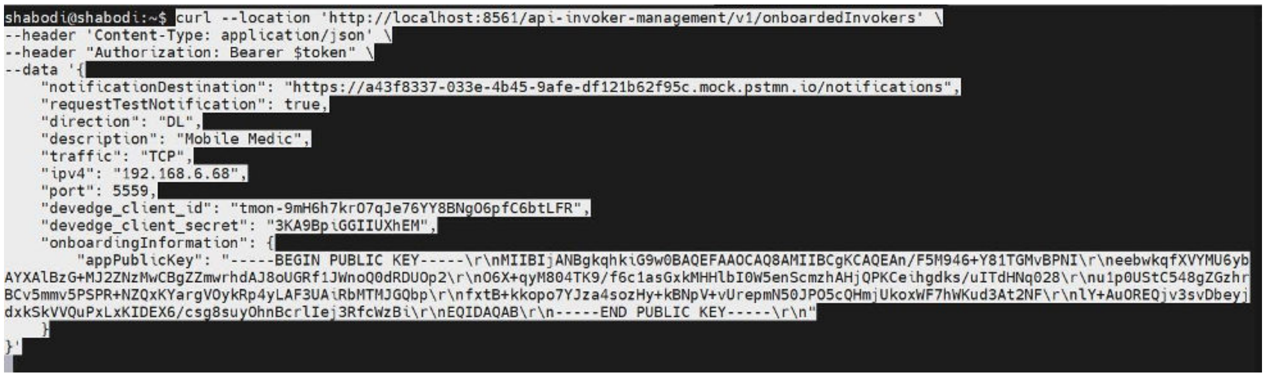


```
shabodi@shabodi:~/aep/aep-cli$ ./shabodi-aep app pre-onboard -n 'Sample app'
logged in

Trace ID: 667dd664a54c4b1aba745b645d20a631

Name: Sample app
Description:
Access_token: eyJraWQiOiI1MjMwNzg0NS1jZjExLTRlOTItYTk2My0zYWZiNjJkYjljNDIiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI0MWYwYWQ3OC0wZWE2LTRlMDktYjdiZC0zODJlMDNhZTllZTkiLCJhdWQiOiJTaGFi
b2RpIEFFUCIsInNjb3BlIjoibmIuYXBwLmMiLCJwcmVmZXJyZWRfdXNlcm5hbWUiOiJTYW1wbGUgYXBwIiwiZXhwIjoxNzE5NjA5MzE2LCJpYXQiOjE3MTk1MjI5MTYsImp0aSI6ImFhODVkYWI2LTJiMzctNGY0OS05YzhkLTh
kNDMxYzYxYTE2ZCJ9.RKD8xyoF_D9PSesvd1SRefaQvqekzh-pcSU0laAJc5U
shabodi@shabodi:~/aep/aep-cli$ 
```

Figure 7 Sample Output 1

4.6.5 Onboard Application (SNBP API):

Replace **aepIp** with your **VM IP** and **access token** with the token from the previous step (step 4). Use the provided curl command to onboard the application.

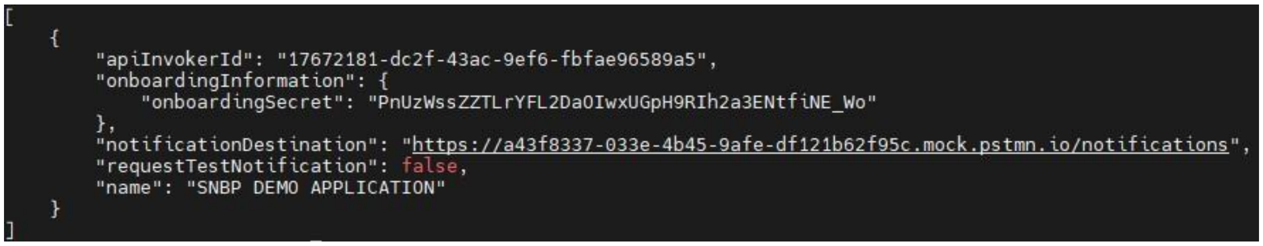


```
shabodi@shabodi:~$ curl --location 'http://localhost:8561/api-invoker-management/v1/onboardedInvokers' \
--header 'Content-Type: application/json' \
--header "Authorization: Bearer $token" \
--data '{
    "notificationDestination": "https://a43f8337-033e-4b45-9afe-df121b62f95c.mock.pstmn.io/notifications",
    "requestTestNotification": true,
    "direction": "DL",
    "description": "Mobile Medic",
    "traffic": "TCP",
    "ipv4": "192.168.6.68",
    "port": 5559,
    "devedge_client_id": "tmon-9mH6h7kr07qJe76YY8BNgO6pfC6btLFR",
    "devedge_client_secret": "3KA9BpiGGIIUXhEM",
    "onboardingInformation": {
        "appPublicKey": "-----BEGIN PUBLIC KEY-----\r\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAn/F5M946+Y81TGMvBPNI\r\neebwkqfXVYMU6yb
AYXAlBzG+MJ2ZNzMwCBgZZmwrhdAJ8oUGRf1JWnoQ0dRDUOp2\r\nO6X+qyM804TK9/f6c1asGxkMHHlbI0W5enScmzhAHjQPKCeihgdks/uITdHNq028\r\nu1p0UStC548gZGzhr
BCv5mmv5PSPR+NZQxKYargVOykRp4yLAF3UAiRbMTMJGQbp\r\nfxtB+kkopo7YJza4sozHy+kBNpV+vUrepmN50JPO5cQHmjUkoxWF7hWKud3At2NF\r\nlY+AuOREQjv3svDbeyj
dxkSkVVQuPxLxKIDEX6/csg8suyOhnBcrlIej3RfcWzBi\r\nEQIDAQAB\r\n-----END PUBLIC KEY-----\r\n"
    }
}'
```

Figure 8 Sample Code Onboard Application

After successfully onboarding your application, you'll receive an Apiinvoker ID and an onboarding secret. These credentials are essential for further steps, such as making authenticated API calls or configuring your application within the system.

The following illustrates the sample output of expected result after executing the command.



```
[
    {
        "apiInvokerId": "17672181-dc2f-43ac-9ef6-fbfae96589a5",
        "onboardingInformation": {
            "onboardingSecret": "PnUzWssZZTLrYFL2DaOIwxUGpH9RIh2a3ENtfiNE_Wo"
        },
        "notificationDestination": "https://a43f8337-033e-4b45-9afe-df121b62f95c.mock.pstmn.io/notifications",
        "requestTestNotification": false,
        "name": "SNBP DEMO APPLICATION"
    }
]
```

Figure 9 Sample Output 2

4.6.6 Add Network Configuration:

- Store the network configuration file (network.json) based on core in a specified location.
- Run ./shabodi-*aep.exe add network -f "C:\Users..."* to generate a network ID with integer value.

Sample output should be "The southbound network has been configured.\n Network id= 9"

4.6.7 Add Devices Configuration:

- Store the device configuration file (device.json), based on ue in a specified location.
- Execute ./shabodi-aep.exe add devices -f "C:\Users..." -n <networkid> to configure the network device.

Note: The networkid value should be updated to match the extracted ID from step 7. Replace networkid with the actual ID obtained in the previous step.

4.6.8 Show Network Information:

- Use *./shabodi-aep.exe show network* to display network IDs and information.

4.6.9 Show Device Information:

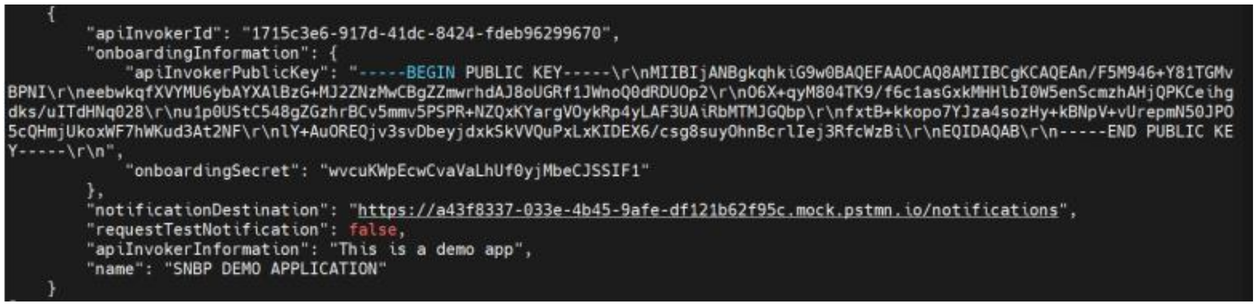
- Run ./shabodi-aep.exe show devices -n <networkid> to view device IDs and information.

Note: To proceed, update the Networkid value with the ID extracted from step 7. Replace Networkid with the actual ID obtained earlier.

4.6.10 Show onboarded App (SNBP API):

- Execute *./shabodi-aep* show *onboardedApp* to display the API invoker ID and onboarding secret.

The following illustrates the sample output of expected result after executing the command.



```
    {
        "apiInvokerId": "1715c3e6-917d-41dc-8424-fdeb96299670",
        "onboardingInformation": {
            "apiInvokerPublicKey": "-----BEGIN PUBLIC KEY-----\r\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAn/F5M946+Y81TGMv
BPNI\r\neebwkqfXVYMU6ybAYXAlBzG+MJ2ZNzMwCBgZZmwrhdAJ8oUGRf1JWnoQ0dRDUOp2\r\nO6X+qyM804TK9/f6c1asGxkMHHlbI0W5enScmzhAHjQPKCeihg
dks/uITdHNq028\r\nu1p0UStC548gZGzhrBCv5mmv5PSPR+NZQxKYargVOykRp4yLAF3UAiRbMTMJGQbp\r\nfxtB+kkopo7YJza4sozHy+kBNpV+vUrepmN50JPO
5cQHmjUkoxWF7hWKud3At2NF\r\nlY+AuOREQjv3svDbeyjdxkSkVVQuPxLxKIDEX6/csg8suyOhnBcrlIej3RfcWzBi\r\nEQIDAQAB\r\n-----END PUBLIC KE
Y-----\r\n",
            "onboardingSecret": "wvcuKWpEcwCvaVaLhUf0yjMbeCJSSIF1"
        },
        "notificationDestination": "https://a43f8337-033e-4b45-9afe-df121b62f95c.mock.pstmn.io/notifications",
        "requestTestNotification": false,
        "apiInvokerInformation": "This is a demo app",
        "name": "SNBP DEMO APPLICATION"
    }
```

Figure 10 Sample Output 3

Add Device Application mapping (SNBP API):

- Use ./shabodi-aep add device-mapping -i <app.id> -d <device.id> to map devices and applications.

Note: To proceed, change the app.id and device.id values as follows:

- Set app.id (also known as apiInvokerId) to the extracted ID from step 10.
- Set device.id to the extracted ID from step 9.

You will see the output "The devices and application has been mapped."

4.6.11 Set Permission (SNBP API):

- Run *./shabodi-aep set permission -l -i <app_Id>* to set permissions for using Shabodi APIs.

Note: To continue, update the app.id (also known as apiInvokerId) value as follows:

- Set app.id to the extracted ID from step 10.

The expected output is to successfully set permission for the license-enabled application to use all Shabodi APIs. The sample output should be: "Successfully set permission."

4.6.12 Invocation token (EF API):

- The purpose is to generate an access token for further steps.
- Obtain an access token using the curl command provided.

```
>> curl -X POST 'http://:8561/security/v1/token' \ --header 'Content-Type: application/json' \ --data '{"client_id": "", "client_secret": ""}'
```

Figure 11 Sample Code Invocation Token

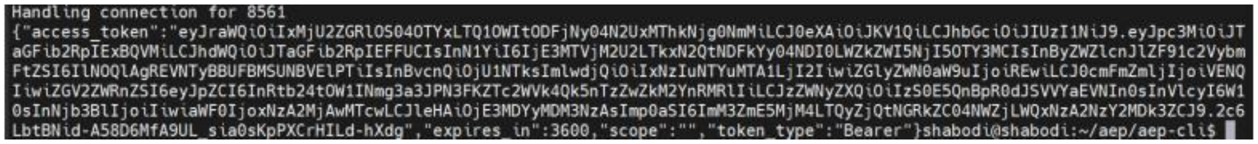
Note:

- Replace <aep-ip> with the actual IP address of your VM.
- Replace <clientid> with the specific ID you obtained during Step 10 [The clientid, also called API Invoker ID is obtained from Step 10 of the setup process. Extract an ID associated with the API invoker. This ID serves as the client ID for authentication.]
- Replace <clientsecret> with the actual secret key you obtained during Step 10 [The clientsecret or Onboarding Secret is also extracted from Step 10. In the same step, you obtain a secret key associated with the onboarding process.]

Remember to replace the placeholders with the correct values specific to your AEP configuration. These parameters are crucial for authenticating and authorizing API requests within the NETAWARE.

The expected output is an access token, which will be used in subsequent 'step15'.

Following is the Sample Output.

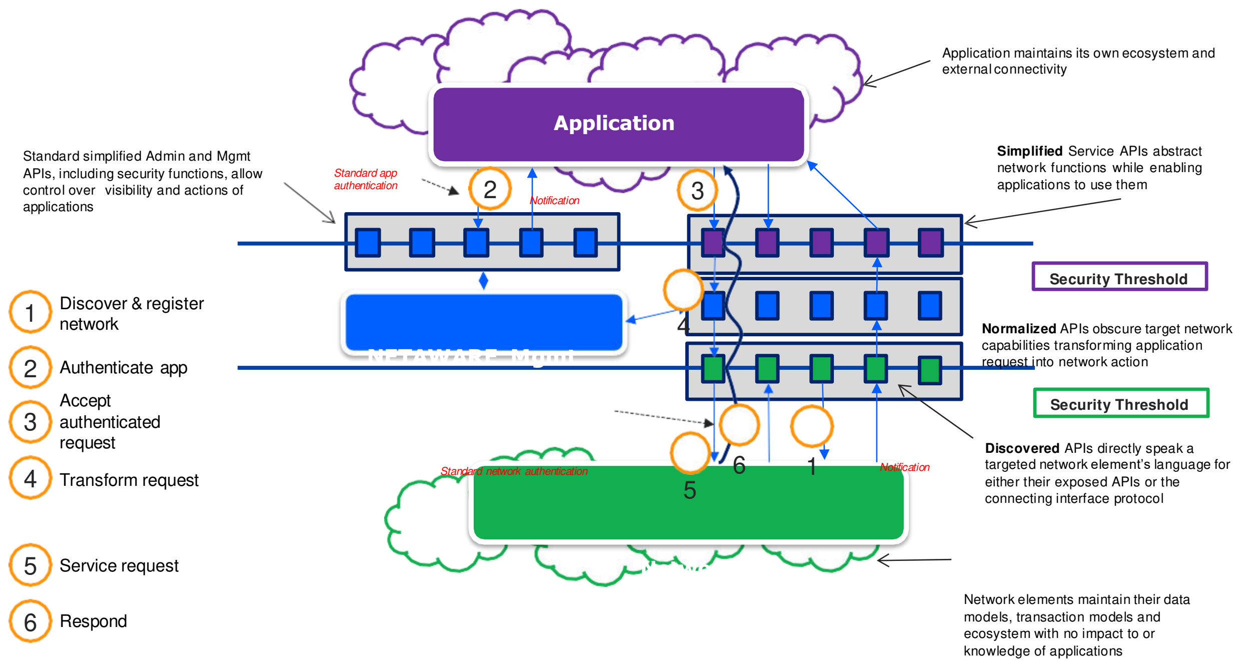


```
Handling connection for 8561
{"access_token":"eyJraWQiOiIxMjU2ZGRlOS04OTYxLTQ1OWItODFjNy04N2UxMThkNjg0NmMiLCJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJT
aGFib2RpIExBQVMiLCJhdWQiOiJTaGFib2RpIEFFUCIsInN1YiI6IjE3MTVjM2U2LTkxN2QtNDFkYy04NDI0LWZkZWI5NjI5OTY3MCIsInByZWZlcnJlZF91c2Vybm
FtZSI6IlNOQlAgREVNTyBBUFBMSUNBVElPTiIsInBvcnQiOjU1NTksImlwdjQiOiIxNzIuNTYuMTA1LjI2IiwiZGlyZWN0aW9uIjoiREwiLCJ0cmFmZmljIjoiVENQ
IiwiZGV2ZWRnZSI6eyJpZCI6InRtb24tOW1INmg3a3JPN3FKZTc2WVk4Qk5nTzZwZkM2YnRMRlIiLCJzZWNyZXQiOiIzS0E5QnBpR0dJSVVYaEVNIn0sInVlcyI6W1
0sInNjb3BlIjoiIiwiaWF0IjoxNzA2MjAwMTcwLCJleHAiOjE3MDYyMDM3NzAsImp0aSI6ImM3ZmE5MjM4LTQyZjQtNGRkZC04NWZjLWQxNzA2NzY2MDk3ZCJ9.2c6
LbtBNid-A58D6MfA9UL_sia0sKpPXCrHILd-hXdg","expires_in":3600,"scope":"","token_type":"Bearer"}shabodi@shabodi:~/aep/aep-cli$
```

Figure 12 Sample Access Token

5 USING NETAWARE APIS

Need API User Guide


Application maintains its own ecosystem and external connectivity
Application
Standard simplified Admin and Mgmt APIs, including security functions, allow control over visibility and actions of applications
Standard app authentication
2
Notification
3
Simplified Service APIs abstract network functions while enabling applications to use them
Security Threshold
4
NETAWARE Mgmt
Normalized APIs obscure target network capabilities transforming application request into network action
Security Threshold
Discovered APIs directly speak a targeted network element's language for either their exposed APIs or the connecting interface protocol
Standard network authentication
5
6
1
Notification
Network elements maintain their data models, transaction models and ecosystem with no impact to or knowledge of applications
1 Discover & register network
2 Authenticate app
3 Accept authenticated request
4 Transform request
5 Service request
6 Respond

6 NETAWARE ADMINISTRATIVE FUNCTIONS

6.1 NETAWARE PORTAL

NETAWARE Portal is under development as of this release. This space is reserved for a future release to describe the functions of the administration portal for NETAWARE.

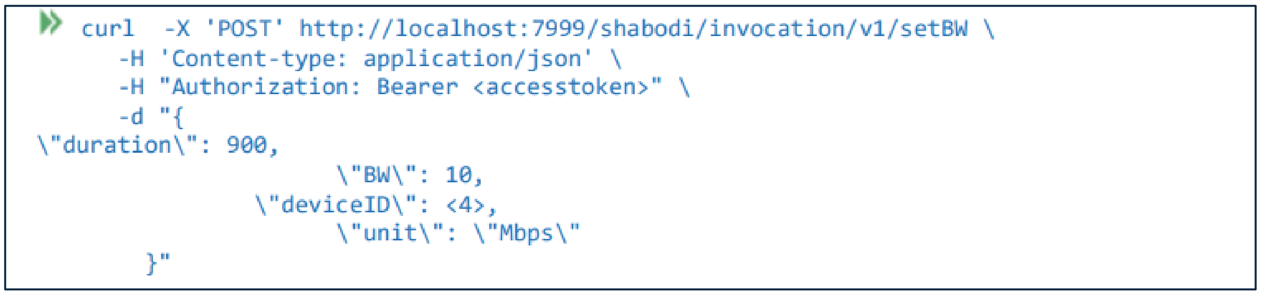
6.2 NETAWARE CLI FUNCTIONS

6.2.1 Set Bandwidth:

This step involves setting up the bandwidth for specific end Device.

Notes:

- Before proceeding, ensure that both the access token and device ID are updated accordingly.
- Follow these steps to modify them:
    - Access Token: Extract the token obtained from Step 13 of the setup process.
    - Device ID: Extract the device ID obtained from Step 9 of the setup process.
- As explained, replace the placeholder values with the actual access token and device ID extracted in the previous steps.



```
curl  -X 'POST' http://localhost:7999/shabodi/invocation/v1/setBW \
      -H 'Content-type: application/json' \
      -H "Authorization: Bearer <accesstoken>" \
      -d "{
\"duration\": 900,
                        \"BW\": 10,
                  \"deviceID\": <4>,
                        \"unit\": \"Mbps\"
        }"
```

Figure 13 Sample Code

- The command used is a curl POST request to the Shabodi API endpoint for setting bandwidth.
- The request includes parameters such as duration, bandwidth (BW), deviceID, and unit.
- Replace the placeholders with actual values and use the curl command to set the bandwidth.
- A 'success' response is expected, indicating that the bandwidth has been set to the specified value for the end Device.

Below is a sample output:

```
{
 "SuccessCode": 200,
 "payload": {
 "SuccessCode": 200,
 "payload": {
 "SuccessCode": 200,
 "payload": [
 {
 "id": 637
 }
 ]
 }
 }
}
```

Figure 14 Sample Output

Please replace <placeholders> with actual values when using these commands.

6.3 Sequence & Status for Initiating Jobs (Example: Device Discovery)

1. Show Tasks: Begin by displaying all available tasks that can be utilized to create a job.
2. Create a Job: For device discovery, select the DeviceDiscovery task.
3. Job Status: CREATED: After creating the job, it enters the CREATED status.
4. Execute Job or Set Job for Future Execution:
   - 4.1. To execute immediately, utilize the execute job command
   - 4.2. To schedule for a specific time or interval, use the set job command.
5. Job Status: RUNNING (Immediate Execution): Upon immediate execution, the job transitions to the RUNNING status. After successful completion, it's marked as COMPLETED.
6. Job Status: READY (Scheduled Execution): For scheduled jobs, the status is initially READY. Upon execution, it changes to RUNNING and then to ONGOING after completion, remaining in this status until the next scheduled execution.
7. View Job Information: Use the show jobs command to monitor job details.
8. Track Job History: Utilize the show job history command to review iteration history, including SUCCESS and FAILURE statuses for executed tasks.
9. Job Completion: Upon reaching the scheduled end time, the job transitions to the COMPLETED status.

Note: For additional information, please refer to the appendix.

6.4 Required CLI Commands:

6.4.1 Show Tasks

- Description: Displays all available cron services in the Shabodi platform for creating jobs.
- Command: ./shabodi-aep show tasks

6.4.2 Show Jobs

- Description: Shows all available jobs in the Shabodi platform.
- Command: ./shabodi-aep show jobs

6.4.3 Job History

- Description: Views the history of all iterations for a particular job.
- Command: ./shabodi-aep show jobs history <Job_Name>

6.4.4 Show Discovered Devices

- Description: Displays all discovered devices in the Shabodi platform.
- Command: ./shabodi show discovered-devices

6.4.5 Create Job

- Description: Creates new jobs in the Shabodi AEP.
- Command: ./shabodi-aep create job <task_name> <job_name>
- Options: Requires task_name (obtained from show tasks) and user-defined job_name.
- Example: ./shabodi-aep create job Task_A Weekly_Job.

6.4.6 Set Job

- Configures jobs to run at specific times or intervals.
- Command: ./shabodi-aep set job
- Options: Frequency, Day (for weekly frequency), Date, Time, Period, and Param.
- Example: ./shabodi-aep set job --frequency daily --time 08:00 --param "param_value"

6.4.7 Stop Job

- Description: Stops a running instance of a job in the Shabodi platform.
- Command: ./shabodi-aep stop <job_name>
- Example: ./shabodi-aep stop Weekly_Job

6.4.8 Delete Job

- Description: Deletes a job from the Shabodi platform.
- Command: ./shabodi-aep delete job <job_name>
- Example: ./shabodi-aep delete job Weekly_Job

6.4.9 Execute Job

- Description: Executes a job immediately in the Shabodi platform.
- Command: ./shabodi-aep execute job <job_name> --param "<network_id>"
- Example: ./shabodi-aep execute job Weekly_Job --param "network_1234"

6.5 Overview of CLI commands for Job Schedule/ Task Scheduler

- Create/configure/execute/show/delete a job.
  - shabodi-aep create job {user_defined_job_name} {shabodi_predefined_task_name} shabodi-aep set job {user_defined_job_name} --frequency {frequency} --day {day} --date {yyyy-MM-dd} --time {HH:mm:ss} --period {period}
  - shabodi-aep exec job {user_defined_job_name}
  - shabodi-aep show job {user_defined_job_name}
  - shabodi-aep delete job {user_defined_job_name}

6.5.1 AEP Tasks for Job Schedule

Table 2 AEP Jobs

Name	Description	Note
device_discovery	Discover devices attached to the Network	Invoke the NB API of Shabodi node for device discovery
utilization_reporting	Generate a report on license utilization	
log_fault_detection	Retrieve logs to detect a fault	

6.6 User Management

- To create an admin:
  - Execute: ./shabodi-aep create user-admin --id {user_admin_id} --password {user_admin_pw} --otp {one-time-password}
- To login:
  - Run: *sudo ./shabodi-aep* login and enter username and password when prompted
  - Alternatively, if executing commands without login:
  - Execute the desired command, e.g., ./shabodi-aep add network -f network.json
  - When prompted for credentials, enter username and password.

6.7 License Management

- To activate a license:
  - Run: *./shabodi-aep activate license* and provide the key file path and prvc file path.
- To update a license:
  - Execute: *./shabodi-aep update* and provide the key file path and prvc file path.
- To get instance information:
  - Run: *./shabodi-aep* get instance information
- To show license configuration:
  - Run: *./shabodi-aep* show config license

6.8 Network Configuration

- To add a network:
  - Run: ./shabodi-aep add network -f {net_config.json}
- To add a network with a knowledge file:
  - Execute: ./shabodi-aep add network -f {net_config.json} -k {knowledge_key_(not_file)} -e {encoded_knowledge_file}
- To update the network:
  - Run: ./shabodi-aep update -n {net_id} -f {net_config.json}
- To delete a network:
  - Execute: ./shabodi-aep delete network -n {net_id}
- To show the network:
  - Run: ./shabodi-aep show network

6.9 Device Configuration

- To add devices:
  - Execute: ./shabodi-aep add devices -f {devices_config.json} -n {net_id} -l {layer_name}
- To update devices:
  - Run: ./shabodi-aep update devices -n {net_id} -f {devices_config_with_ID.json} -l {layer_name}
- To delete a device:
  - Execute: ./shabod-aep delete devices -n {net_id} -d {device_id}
- To show devices:
  - Run: ./shabodi-aep show devices -n {net_id}

6.10 Event Repository

- To see all historical events:
  - Execute: ./shabodi-aep show events --all
- To query historical events:
  - Execute: ./shabodi-aep show events
  - Select a search option based on Event Id, Event Name, Event Topic, Issuer, or Issued After.

6.11 Job Management

- To create a job:
  - Execute: ./shabodi-aep create job <task_name> <job_name>
- To show jobs:
  - Execute: ./shabodi-aep show jobs
- To show job history:
  - Execute: ./shabodi-aep show jobs history <Job_Name>
- To set a job:
  - Execute: ./shabodi-aep set job --frequency "every_1m" --period "day" --param "<network_id>"
- To stop a job:
  - Execute: ./shabodi-aep stop <job_name>
- To delete a job:
  - Execute: ./shabodi-aep delete job <job_name>

7 Configuring Networks & Devices

Before diving into AEP functionalities, ensure you have the following prepared:

7.1 Network Configuration File (network.json)

Before deploying AEP, it is crucial to create Network Configuration. When you first launch AEP, it will require the network.json file to understand your network layout. This one-time configuration allows AEP to build its internal representation of the network.

7.1.1 Understand the Purpose:

- The Network.json file serves as a blueprint for AEP's network setup.
- The file includes details about network nodes.
- This file ensures consistent network behavior and simplifies AEP administration.

7.1.2 Create the Network Configuration File:

- Use a text editor or any JSON editor to create a new file named Network.json.
- The file structure resembles the following:

```
{
 "NetworkID" : 5,
 "Network": {
   "Components": [
     {
       "ComponentCategory": "5GCORE",
       "ComponentNodes": [
         {
           "NodeDomain": "EMPTY",
           "NodeIPv4": "10.55.0.1",
           "NodeIPv6": "EMPTY",
           "NodeId": 0,
           "NodePort": "8032",
           "NodeType": "UPF"
         },
         {
           "NodeDomain": "EMPTY",
           "NodeIPv4": "192.168.1.184",
           "NodeIPv6": "EMPTY",
           "NodeId": 1,
           "NodePort": "7777",
           "NodeType": "PCF"
         },
       ],
       "ComponentVendor": "open5GS",
       "ComponentVersion": "v2.6.2"
     }
   ],
 },
 "documentName": "SBNodeDetails",
 "documentVersion": "v2.0",
 "symmetricKey": "kMhNbwCsUiY2rvQPYBwVrQNrtz4Bzql2"
}
```

Figure 15 Network Configuration

- Customize the network element/function details according to your network setup.

7.1.3 Network Element/Function Configuration:

- Each network element/function should have a unique name and a type.
- Interfaces with relevant information.

7.1.4 Save the File:

- Save the Network.json file in a location accessible to AEP.

7.1.5 Deploy AEP:

- When AEP boots up for the first time, provide the Network.json file.
- AEP will use this configuration during initialization.
- Subsequently, you don’t need to re-enter the configuration each time you start or stop AEP.

7.1.6 Updating Configurations:

- To update network, use CLI ./shabodi-aep.exe update network -n <network_id> -f <path>.
- To delete use ./shabodi-aep.exe delete network -n <network_id> -a <all network>.

7.2 DEVICE CONFIGURATIONS (DEVICE.JSON)

The Device Configuration (Device.json) file is crucial for configuring end Devices within AEP. This JSON file focuses on user-specific settings.

Follow these steps to create the configuration file.

7.2.1 Understand the Purpose:

- The Device.json file introduces end devices to AEP within the network.
- The file lists all relevant devices for specific applications within the network scope.
- Additionally, the file maps these devices to the network within AEP.

7.2.2 Create the Device Configuration File:

- Use a text editor or any JSON editor to create a new file named Device.json.
- The file structure should resemble the following:

```
{
  "NetworkID": 2,
  "UserEquipments": [
    {
      "DataPlaneActive": true,
      "UEID": 1,
      "UEInformation": {
        "DeviceInfo": {
          "IMEI": "353490069873319"
        },
        "SubcriberInfo": {
          "DNN": "internet",
          "DOWNLINK_AMBR": 100000,
          "IP": "10.254.0.6",
          "SliceInfo": [],
          "UPLINK_AMBR": 100000
        },
        "Supi": "901700000000008"
      }
    }
  ]
}
```

Figure 16 Device Configuration

- Customize the end Device details according to your deployment.

7.2.3 Save the File:

- Save the Device.json file in a location accessible to AEP.

7.2.4 Updating Configurations:

- Devices can be added via the CLI Command ./shabodi-aep update devices -n <network_id> -f <path> -l <SRC_layer>.
- To remove, use ./shabodi-aep delete devices -n <network_id> -a -d <device_ids>

8 FAULT AND PERFORMANCE MANAGEMENT SYSTEM

This guide outlines FCAPS/OAMP (Fault, Configuration, Accounting, Performance, Security/Operations, Administration, Maintenance, and Provisioning) system, specifically focusing on Fault Management and Performance Management enhancements.

- The system is engineered to streamline the management of network operations by integrating a central message broker queue, which collects ShabodiEvents from various services, signaling noteworthy occurrences within the network.
- Provide network administrators and operators with a more efficient and reliable platform for monitoring and managing network health and performance.
- By leveraging advanced search engine technologies like ZincSearch, OpenObserve, Apache Lucene, and Elasticsearch, the system offers a scalable solution for full-text indexing and observability at a petabyte scale.
- These technologies are carefully selected and compared to ensure they align with the system's requirements, offering a balance between performance, cost-efficiency, and ease of operation.

The system's benefits include:

- Enhanced Fault Management: With a robust set of rules and configurations, the system can swiftly identify and respond to network events, minimizing downtime and ensuring continuous service delivery.
- Performance Management: Through meticulous logging and analysis, the system provides real-time insights into network performance, allowing for proactive optimizations and improved service quality.

8.1 FAULT MANAGEMENT CLI COMMANDS

The following CLI commands are essential for managing faults within the Shabodi system. Fault severity levels are categorized on a scale of **0 to 5:**

<0> Clear

<1> Critical

<2> Major

<3> Minor

<4> Indeterministic

<5> Info

Ensure that you have the necessary privilege levels for each command and adhere to the specified structure and parameters.

Table 3 Fault Management CLI Commands

Command	Syntax	Description	Response Format
List all faults in the system	shabodi-aep faults -a	Lists all the faults from all the components in the system.	ERROR:<Component>:<sub-component>:<Description>:<Severity>:<Time:Date>

List all faults by component	shabodi-aep <component> faults -a	Lists all the faults from the specific component in the system.	ERROR:<Component>:<sub-component>:<Description>: <Severity>:<Time:Date>
List all faults by sub-component	shabodi-aep <component> <sub-component> faults -a	Lists all the faults from the specific sub-component in the system.	ERROR:<Component>:<sub-component>:<Description>: <Severity>:<Time:Date>
List all faults by date & time	shabodi-aep faults -d <startdate:time-enddate:time> Note: Add ability to set enddate:time attribute as todate to use the current date and time of the server.	Lists all the faults from the system in the date range.	ERROR:<Component>:<sub-component>:<Description>: <Severity>:<Time:Date>
List all faults by date, time & component	shabodi-aep faults -d <startdate:time-enddate:time> Note: Add ability to set enddate:time attribute as to date to use the current date and time of the server.	Lists all the faults from the specific component in the date range.	ERROR:<Component>:<sub-component>:<Description>: <Severity>:<Time:Date>
List all faults by date, time & sub-component	shabodi-aep <component> faults -d <startdate:timeenddate:time> Note: Add ability to set enddate:time attribute as todate to use the current date and time of the server.	Lists all the faults from the specific sub-component in the date range.	ERROR:<Component>:<sub-component>:<Description>: <Severity>:<Time:Date>
Clear all faults in the system	shabodi-aep faults --clear	Clear all the faults in the system.	All faults cleared.
Clear all faults by component	shabodi-aep <component> faults --clear	Clear all the faults from the specific component in the system.	All <component> faults cleared
Clear all faults by sub-component	shabodi-aep <component> faults –clear	Clear all the faults from the specific sub-component in the system.	All <sub-component> faults cleared.
Clear all faults by date & time	shabodi-aep faults -d <startdate:time-enddate:time> --clear Note: Add ability to set enddate:time attribute as todate to use the current date and time of the server.	Clear all the faults from the system in the date range.	All faults for date range startdate:time-enddate:time cleared.
Clear all faults by date, time & component	shabodi-aep <component> faults -d <startdate:timeenddate:time> --clear Note: Add ability to set enddate:time attribute as todate to use the current date and time	Clear all the faults from the specific component in the date range.	All <component> faults for date range startdate:time-enddate:time cleared.

	of the server		
Clear all faults by date, time & sub-component	shabodi-aep <component> <sub-component> faults -d <startdate:time-enddate:time> --clear Note: Add ability to set enddate:time attribute as todate to use the current date and time of the server.	Clear all the faults from the specific sub-component in the date range.	All <sub-component> faults for date range startdate:time-enddate:time cleared.
List all faults in the system by severity	shabodi-aep faults -s <severity>	Lists all the faults from all the components in the system by <severity>	ERROR:<Component>:<sub-component>:<Description>:<Severity>:<Time:Date>
List all faults by component and severity	shabodi-aep <component> faults -s <severity>	Lists all the faults from the specific component in the system by severity.	ERROR:<Component>:<sub-component>:<Description>:<Severity>:<Time:Date>
List all faults by sub-component and severity	shabodi-aep <component> <sub-component> faults -s <severity>	Lists all the faults from the specific sub-component in the system by severity.	ERROR:<Component>:<sub-component>:<Description>:<Severity>:<Time:Date>
List all faults by date & time and severity	shabodi-aep faults -d <startdate:time-enddate:time> -s <severity> Note: Add ability to set enddate:time attribute as todate to use the current date and time of the server	Lists all the faults by severity from the system in the date range.	ERROR:<Component>:<sub-component>:<Description>:<Severity>:<Time:Date>
List all faults by severity, date, time & component	shabodi-aep <component> faults -d <startdate:time-enddate:time> -s <severity> Note: Add ability to set enddate:time attribute as todate to use the current date and time of the server	Lists all the faults by severity from the specific component in the date range.	ERROR:<Component>:<sub-component>:<Description>:<Severity>:<Time:Date>
List all faults by severity, date, time & sub-component	shabodi-aep <component> <sub-component> faults -d <startdate:time-enddate:time> -s <severity> Note: Add ability to set enddate:time attribute as todate to use the current date and time of the server.	Lists all the faults by severity from the specific sub-component in the date range.	ERROR:<Component>:<sub-component>:<Description>:<Severity>:<Time:Date>
Clear all faults by severity in the system	shabodi-aep faults -s <severity> --clear	Clear all the faults by severity in the system	All faults cleared.

Clear all faults by component and severity	shabodi-aep <component> faults -s <severity> --clear	Clear all the faults from the specific component by severity in the system.	All <component> <severity> faults cleared.
clear all faults by sub-component and severity	shabodi-aep <component> <sub-component> faults -s <severity> --clear	Clear all the faults from the specific sub-component by severity in the system.	All <sub-component> <severity> faults cleared.
Clear all faults by severity,date & time	shabodi-aep faults -d <startdate:time-enddate:time> - s <severity> --clear Note: Add ability to set enddate:time attribute as todate to use the current date and time of the server	Clear all the faults by severity from the system in the date range.	All <severity> faults for date range startdate:time-enddate:time cleared.
Clear all faults by severity, date, time & component	shabodi-aep <component> <sub-component> faults -d <startdate:time-enddate:time> - s <severity> --clear Note: Add ability to set enddate:time attribute as todate to use the current date and time of the server.	Clear all the faults by severity from the specific sub-component in the date range.	All <component> <severity> faults for date range startdate:time-enddate:time cleared.
Clear all faults by severity, date, time & sub-component	shabodi-aep <component> <sub-component> faults -d <startdate:time-enddate:time> - s <severity> --clear Note: Add ability to set enddate:time attribute as todate to use the current date and time of the server.	Clear all the faults by severity from the specific sub-component in the date range.	All <sub-component> <severity> faults for date range startdate:time-enddate:time cleared.

8.2 GENERALIZED FAULT MANAGEMENT

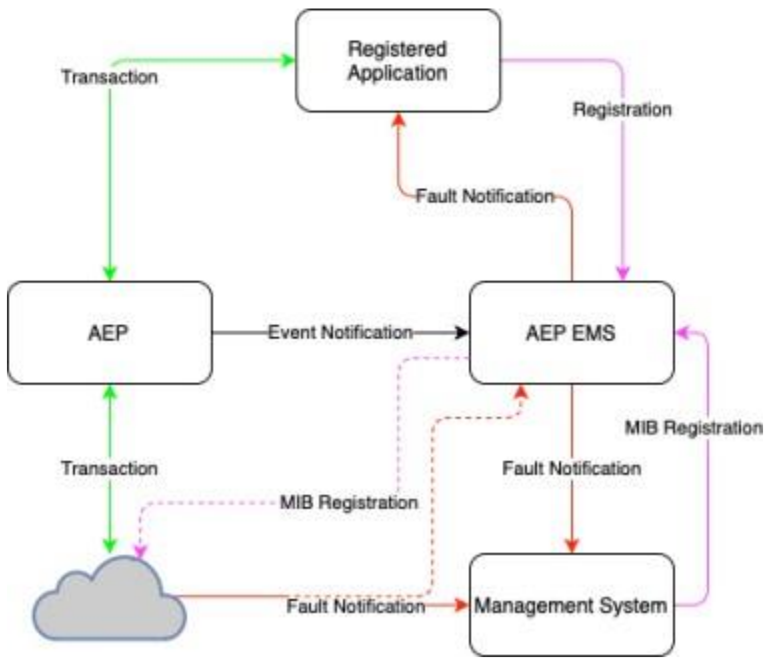


Figure 17 Fault Management Flow

Management systems and applications register for AEP's fault notifications. Transactions from registered applications flow through NETAWARE to networks. NETAWARE detects and notifies the Element Management System (EMS) of abnormal events. The EMS handles event correlations, assessments, logging, and notifications. If NETAWARE encounters a service-affecting state, it informs the NETAWARE 's EMS.

NETAWARE 's EMS may register for Network MIBs notifications to assist attached

applications. NETAWARE 's EMS publishes MIBs, alerts, and faults, monitors conditions, and

notifies stakeholders.

8.3 FAULT MANAGEMENT

- To configure SNMP trap receiver:
  - Execute: ./shabodi-aep set snmp --ip {trap_receiver_ip} --port {trap_receiver_port} --retries {retries} --timeout {timeout}
  - Retries and timeout are optional.
- To display SNMP configuration:
  - Execute: ./shabodi-aep show config snmp
- To show alarms:
  - Execute: ./shabodi-aep show alarm -s INACTIVE or ./shabodi-aep show alarm -i <alarm-id>
- To set alarm state:
  - Execute: ./shabodi-aep set alarm -s <state> -i <alarm-id>

9 Licensing & Policy Management

- Policy Configuration:
  - Policies define rules and access controls for various operations.
  - Administrators configure policies to enforce security and compliance.
- Token Generation:
  - Tokens are generated for secure application function registration and invocation.
  - These tokens play a crucial role in maintaining system integrity.

9.1 Policy Configuration

9.1.1 Setting Permissions:

- To grant permission to all APIs:
  - Execute: ./shabodi-aep set permission -i {app_id} -l
- To grant permission to specific APIs:
  - Execute: ./shabodi-aep set permission -i {app_id} -c {api_category_name} -a {api_name1, api_name2}
  - Example: ./shabodi-aep set permission -i e69d0e2b-d04b-4062-8f50-5efbd6353f4d -c shabodi -a setBW,deviceDiscovery

9.1.2 Revoking Permissions:

- To remove permissions for all APIs:
  - Execute: ./shabodi-aep revoke permission -i {app_id} -l
- To remove permission for a specific API:
  - Execute: ./shabodi-aep revoke permission -i {app_id} -a {api_name} -c {api_category_name}
  - Example: ./shabodi-aep revoke permission -i e69d0e2b-d04b-4062-8f50-5efbd6353f4d -a setBW -c shabodi

9.2 AEP CLI commands for License

The following commands enable you to effectively manage licenses within the NETAWARE environment.

9.2.1 Renew or Update License:

- Use *shabodi-aep* update to renew or update your existing license with a new license file. Provide the license key file and license xml file as arguments. Upon success, the command will return the Instance ID and License Key for verification.

9.2.2 View License Metrics:

- Utilize *shabodi-aep* show license metrics to access license usage metrics, providing insights into account utilization.

Table 4 CLI Commands for License

Command	Arguments	Response	Description
shabodi-aep update	license key file license xml file	Instance ID License Key	Renew or update the existing license with a new license.
shabodi-aep show license metrics		Account Utilization (details below)	Displays license metrics.

9.2.3 Account (Utilization)

- Tracks user login activity, providing insights into user engagement and system access.
- Monitors system alerts and notifications, facilitating proactive management of issues and system health.
- Describes the primary functionalities and options available within the NETAWARE environment for efficient utilization.
- Presents visual representations of license usage and allocation, aiding in understanding and optimizing license utilization.
- Tracks invocations and utilization for Trivialized category services, including Bandwidth (BW) Service, Latency Service, and Jitter Service.
- Tracks invocations and utilization for Insights category services, including Application Utilization, Device Utilization, and Connectivity Utilization.

9.2.4 Access Account Utilization

- $ shabodi-aep account -id admin
- Enter password for admin: admin

9.2.5 View Account Utilization

The following alert is displayed on the CLI when thresholds are exceeded. Commands are made available to correct the issue.

- The License will expire in 27 days.
- Trivialized Category Invocations exceeded the configured threshold [70%] (710,000/700,000)
  - BW Service [90%] (450,000/450,000)
  - Latency Service [80%] (450,000/350,000)
  - Insights Category Invocations exceeded the configured threshold (720,000/700,000)
  - Application Utilization [70%] (360,000/280,000)
- What would you like to do?

1. show utilization

2. configure threshold alerts

>

9.3 NETAWARE ALERTS FOR LICENSING

This section provides sample outputs of alerts for different categories of licensing. These alerts are provided through the SNMP service and related APIs for the NETAWARE .

9.3.1 General alerts for invocation thresholds

- Simplified Category Invocations exceeded the configured threshold [70%] (710,000/700,000)
  - BW Service [90%](450,000/450,000)
  - Latency Service [80%](450,000/350,000)
- Insights Category Invocations exceeded the configured threshold (720,000/700,000)
  - Application Utilization [70%] (360,000/280,000)

9.3.2 Utilization

- License Issue Date: 2023.11.15 (Wed)
- License Expiration Date: 2024.11.15 (Fri)

9.3.3 Simplified Category

- Allowed Invocations: 1,000,000
- Invocations Utilized: 710,000
- Invocations available: 290,000
- Utilization remaining: 29%
- Threshold for Alert: 70% (700,000 invocations)

9.3.4 Bandwidth(BW) Service

- Configured Invocations: 500,0000
- Invocations Utilized: 450,000
- Invocations available: 50,000
- Utilization remaining: 10%
- Threshold for Alert: 70% (350,000 invocations)
- Threshold for Alert: 80% (400,000 invocations)
- Threshold for Alert: 90% (450,000 invocations)
    - getDownlinkBWforUE API
        - Invocations Utilized: 20,000
    - setDownlinkBWforUE API
        - Invocations Utilized: 250,000
    - getUplinkBWforUE API
        - Invocations Utilized: 10,000
    - setUplinkBWforUE API
        - Invocations Utilized: 50,000
    - getDownlinkBWforSlice API
        - Invocations Utilized: 10,000
    - setDownlinkBWforSlice API
        - Invocations Utilized: 50,000
    - getUplinkBWforSlice API
        - Invocations Utilized: 10,000
    - setUplinkBWforSlice API
        - Invocations Utilized: 50,000

9.3.5 Latency Service

- ConfiguredInvocations: 300,000
- Invocations Utilized: 240,000
- Invocations available: 60,000
- Utilization remaining: 20%
- Threshold for Alert: 70% (210,000 invocations)
- Threshold for Alert: 80% (240,000 invocations)
- Threshold for Alert: 90% (270,000 invocations)
    - getDownlinkLatencyforUE API
        - Invocations Utilized: 10,000
    - setDownlinkLatencyforUE API
        - Invocations Utilized: 150,000
    - getUplinkLatencyforUE API
        - Invocations Utilized: 10,000
    - setUplinkLatencyforUE API
        - Invocations Utilized: 20,000

- getDownlinkLatencyforSlice API
  - Invocations Utilized: 5,000
- setDownlinkLatencyforSlice API
  - Invocations Utilized: 20,000
- getUplinkLatencyforSlice API
  - Invocations Utilized: 5,000
- setUplinkLatencyforSlice API
  - Invocations Utilized: 20,000

9.3.6 Jitter Service

- ConfiguredInvocations: 200,000
- Invocations Utilized: 120,000
- Invocations available: 80,000
- Utilization remaining: 40%
- Threshold for Alert: 70% (140,000 invocations)
  - getDownlinkJitterforUE API
    - Invocations Utilized: 2,500
  - setDownlinkJitterforUE API
    - Invocations Utilized: 70,000
  - getUplinkJitterforUE API
    - Invocations Utilized: 2,500
  - setUplinkJitterforUE API
    - Invocations Utilized: 20,000
  - getDownlinkJitterforSlice API
    - Invocations Utilized: 2,500
  - setDownlinkJitterforSlice API
    - Invocations Utilized: 10,000
  - getUplinkJitterforSlice API
    - Invocations Utilized: 2,500
  - setUplinkJitterforSlice API
    - Invocations Utilized: 10,000

9.3.7 Insights Category

- Allowed Invocations: 1,000,000
- Invocations Utilized: 720,000
- Invocations available: 280,000
- Utilization remaining: 28%
- Threshold for Alert: 70% (700,000 invocations)
  - Application Utilization
    - Configured Invocations: 400,0000
    - Invocations Utilized: 360,000
    - Invocations available: 40,000
    - Utilization remaining: 10%
    - Threshold for Alert: 70% (280,000 invocations)
  - Device Utilization
    - Configured Invocations: 300,0000
    - Invocations Utilized: 180,000
    - Invocations available: 120,000
    - Utilization remaining: 60%
    - Threshold for Alert: 70% (210,000 invocations)
  - Connectivity Utilization
    - Configured Invocations: 300,0000
    - Invocations Utilized: 180,000
    - Invocations available: 120,000

- Utilization remaining: 60%
- Threshold for Alert: 70% (210,000 invocations)

9.3.8 Visualized License Metrics: Alerts are shown again.

```
What you would like to do:

1. show utilization

2. configure the allowed invocations for service category

3. configure threshold alerts

> 1
```

10 REPORTS

The report CLI command allows you to generate and schedule reports. Whether you need insights on system performance, user activity, or other metrics, the report command is used.

- Base command syntax: *./shabodi-aep report <sub-command>.*

10.1 GENERATE REPORT

The generate command allows you to create reports on-demand within the Shabodi NETAWARE environment.

- Command: ./shabodi-aep report generate
- **Options:** This command will prompt you interactively for the following information:
    - Report Name: Specify the desired name for the generated report.
    - Start Date: Enter the date and time for the report data to begin (format depends on system configuration).
    - End Date: Enter the date and time for the report data to end (format depends on system configuration).

10.2 SCHEDULE REPORT

The schedule command allows you to configure automated report generation within the Shabodi NETAWARE .

- Command: ./shabodi-aep report schedule
- Options: This command will prompt you interactively for the following information:
    - Report Name: Specify the desired name for the scheduled report.
    - Schedule Frequency: Enter the desired frequency for report generation (e.g., every 1 day, every 2 weeks, etc.).
    - Schedule Unit: Select the time unit for the chosen frequency (seconds, minutes, days, weeks, months, years).

10.3 REPORT MANAGEMENT

- To generate a report:
    - Execute: ./shabodi-aep report generate
- To schedule a report:
    - Execute: ./shabodi-aep report schedule

11 NETAWARE Commands and Alerts

11.1 Overview

This documentation provides clear steps for users to follow for setting up and managing the NETAWARE . Remember to consult the specific installation and admin guides mentioned for more detailed instructions.

11.2 AEP-IP (AEP IP)

Refers to the IP address of user's virtual machine (VM) where Adobe Experience Platform (AEP) is installed.

11.3 Possible Job Statuses

- CREATED: A new job is created.
- READY: Configured and awaiting first execution.
- RUNNING: Job is currently executing its assigned task.
- ONGOING: Job is awaiting the next scheduled execution.
- COMPLETED: Job has completed its lifecycle.
- CANCELLED: Job is forcefully stopped.
- INTERRUPTED: Job encounters an exception or failure.

11.3.1 Arguments

- Frequency: once, every_1m, every_10m, every_30m, every_hour, daily, weekly
- Day: monday, tuesday, …, sunday
- Date: yyyy-mm-dd
- Time: 24-hour format
- period: day, 24h, week, month, year, forever

11.3.2 Examples

- shabodi-aep create job druid_device_discovery1 device_discovery
- shabodi-aep create job druid_device_discovery2 device_discovery
- shabodi-aep set job druid_device_discovery1 --frequency daily --time 22:00:00 --period week --params {"ip":"19
- shabodi-aep set job druid_device_discovery2 --frequency weekly --date monday --time 07:00:00 --period forever
- shabodi-aep set job druid_device_discovery2 --frequency once --date 2024-03-01 --time 07:00:00 --params {"ip":
- shabodi-aep show job druid_device_discovery1
- shabodi-aep delete job druid_device_discovery1
- shabodi-aep exec job druid_device_discovery1
- shabodi-aep show job --exce_history druid_device_discovery1

12 Getting Support from Shabodi

Shabodi provides support to all systems as part of paid contracts terms. General terms include technical support for deployed systems with a valid license (production and lab), along with access to documentation and support tools.

While some contracts may specify special support terms, Shabodi's typical support is as follows:

12.1 Support levels

12.1.1 Tier 1 – Administrative Support

This is the first line of support available from Shabodi available to all customers. This support is intended to help customers with minor issues, or to gather information to pass along to technical support. Typical issues resolved include license problems, system access problems, Shabodi support service access problems and other related concerns.

First level support is provided through email at support@shabodi.com.

12.1.2 Tier 2 – Technical Support

In the event a technical problem has arisen with Shabodi's NETAWARE , the problem ticket will be passed on to Tier 2 technical support. This is a person with the skills and tools to dig more deeply into the NETAWARE system to resolve issues. Resolutions at this stage generally apply to a single instance or customer.

12.1.3 Tier 3 – Product Support

This level of support is required when deep technical issues exist. Involving this support level will be at the discretion of the Tier 2 support prime. Outcomes from this support level generally involve changes to product software or configuration files, and pertain to the product overall versus a single deployment (like Tier 2 support).