Sippo wac administration guide

Release 4.1.0

Quobis

Jun 03, 2020

CONTENTS

Quobis
P.I A Granxa - Rua D - Paralela 1
“Casa de Pedra”
36475 PORRIÑO - PONTEVEDRA
Galicia - Spain
Tel.: +34 986 911 644
info@quobis.com - www.quobis.com

CHAPTER

ONE

INTRODUCTION

Contents

1.1 About Quobis

Quobis is a leading European company in the delivery of carrier-class unified communication solutions for communication service providers and enterprises with a special focus on security, interoperability and identity management. Quobis is headquartered in Vigo (Spain) with trusted partners throughout the world.

Quobis product portfolio includes Sippo wac, Sippo collaborator and Sippo hub, which are deployed in production in a number of telecom service providers, cloud providers, ITSP and contact centers around the world. Quobis is well-known as one of the leaders in the deployment of WebRTC technology after being involved in the standardization process and taking part in the industry-firsts implementations in more than 40 countries, including most of the top 20 telcos worldwide.

1.2 About this guide

The aim of this guide is to help system administrators to deploy, configure, troubleshoot and manage a Sippo wac. The information is written for experienced system administrators who are familiar with VoIP and UC networks. This guide is valid for Sippo v 4.1 only, for previous versions of this guide, please contact your sales representative.

1.3 Comments and suggestions

Quobis is committed to developing quality documents for its customers. If you have any suggestions, questions, comments or concerns about our documentation, please contact your sales representative in Quobis or send an email to support@quobis.com.

1.4 Revision history

This section contains a revision history for this guide. Please note that the product name has changed since version v3.0 from Sippo WebRTC Application Controller to “Sippo wac”.

Table 1: Revision history

Date	Revision	Description
December 2013	Revision 0.1	First version of guide published v1.0
May 2014	Revision 0.2	General revision
September 2014	Revision 0.3	Update to WAC version 2.0.2
December 2014	Revision 0.4	Update to WAC version 2.1
January 2015	Revision 0.5	Installation procedure and configuration review
February 2015	Revision 0.6	Update to WAC version 2.2
July 2015	Revision 0.7	SSH keys annex added
August 2015	Revision 0.8	Update to WAC version 2.3
September 2015	Revision 0.9	Credentials & profiles detailed explanation.
November 2015	Revision 1.0	Stable. Added WebRTC stats appendix (QoS)
March 2016	Revision 1.1	Stable. Complete review for Sippo v2.4
October 2016	Revision 1.2	Stable. Complete review for Sippo v3.0
December 2017	Revision 1.3	Stable. Complete review for Sippo v3.1
May 2019	Revision 2.0	Doc engine changes. Complete review for Sippo v4.0
April 2020	Revision 3.0	Release version update. Complete review for Sippo v4.1

CHAPTER

TWO

SIPPO INTRODUCTION

2.1 About Sippo wac

Sippo wac is a software solution developed by Quobis that allows telephone companies, CSPs (Cloud Service Providers), MVNOs (Mobile Virtual Network Operator) and large enterprises, to create real time multimedia and collaborator applications. These applications can be used on desktop web-browsers, mobile browser and o mobile application on Android and iOS smartphone. The original acronym “WAC” stands for *WebRTC Application Controller*.

Sippo wac has been designed to be a carrier-class solution with special focus on scalability, reliability and interoperability with existing UC platforms and networks.

Sippo wac has been developed by Quobis since 2011 and it is distributed worldwide through a network of first-class partners and UC vendors.

2.1.1 About WebRTC technology

Sippo product family has its roots in the to WebRTC technology, which is a free, open project that enables web browsers with Real-Time Communications (RTC) capabilities via simple JavaScript APIs. The WebRTC components have been optimized to best to serve this purpose. The mission of WebRTC is to enable rich, high quality, real time applications to be developed in the browser via simple JavaScript APIs and HTML5.

WebRTC offers web application developers the ability to write rich, real-time multimedia applications on the web, without requiring plugins, downloads or installs. It's purpose is to help build a strong RTC platform that works across multiple web browsers, across multiple platforms. Nowadays, WebRTC technology can also be used on smartphones and also other embedded devices.

The WebRTC initiative is a project supported by Google, Mozilla and Opera. More information can be found online at the website www.webrtc.org

2.1.2 Understanding Sippo wac role

Web applications are usually built on top of application servers or web frameworks, such as JBoss, NodeJS, PHP, etc... Real-time audio and video applications are built the same way: HTML5 code rendered by your browser making a series of request to an application server.

So what is a "WebRTC Application Controller" and why I need one?

The term "WebRTC Application Controller" has been coined by Quobis after our experience deploying WebRTC projects in service providers and enterprises all around the world. In a real setup, there are a large number of features and functionalities that need to be delivered and that you cannot find at the web server for example:

- user authentication
- identity management
- multi-multiparty conferencing
- dialplan integration
- push notifications
- security control
- scalability
- QoS control
- and a large etcetera

Sippo wac fills these gaps that you need to put a application prototype into real production.

Thanks to its flexible SDKs and configuration parameters, Sippo wac supports a large number of business cases, from a simple click-to-dial button to advanced scenarios like UC collaboration, unified messaging, remote contact-center agents, video customer onboarding, etc.. in a number of verticals such as banking, health, logistics, retail, etc. . .

Sippo wac is standards compliant and has been designed and developed by engineers who participate in WebRTC standardization forums like W3C, IETF, 3GPP, SIPForum and GSMA.

2.1.3 Main functionalities

On
Boost app development
Boosts app development through a set of APIs
Hide complexity
Activates lots of use cases just with one element
Service registration
Orchestrates deployment via user and service management

Sippo wac provides a number of functionalities that boost the development of real time applications. The most relevant functionalities provided by Sippo wac are detailed in the list below:

HQ Voice and Video Sippo SDKS provides the ability to embed voice and video calls with standard codecs such as VP8 and H264. SippoSDK is able to control the quality of the video streams.

User Management Sippo wac handles the management of subscribers, its capabilities and its classification into domains, supporting multiple tenants into the same deployment.

Identity Management Sippo wac provides and internal authentication system to authenticate the subscribers, but it also can delegate the authentication of the users into a third party element that supports OAuth2 protocol, such as Microsoft Active Directory FS. Subscribers can have several identities and user a number of different authentication models.

Policy control subscribers can perform different actions depending on the permissions and policies setup by the system administrator.

Contact management Sippo has an internal Network Addressbook (NAB) that can be used to manage the agenda of the subscribers. It takes into account that subscribers and belong to diferent domains, organizations and groups, so a number of combination of private contacts, shared contacts and group contacts is already available. This NAB can also be synchronized with external systems via a REST API.

Recording The recording capability of Sippo SDKs allows the implementation of use cases where there is a legal requirement for recording every interaction in a secure and encrypted way.

SIP interconnection Beyond its internal signalling server, Sippo is able to connect to existing SIP and IMS networks with flexible SIP schemas, from a SIP trunk to complex registration scenarios and SIP manipulation rules. That enables a range of innovative applications to extended traditional telephony applications into the cloud.

Conferencing Beyond one-to-one calls, Sippo wac allows the setup of multiparty audio and video rooms with fine-grained functionalities, such as public conferences, private conferences, one-use conferences, etc. . . depending on the use case, billing strategy, etc. . .

Messaging Beyond audio and video communications, Sippo wac provides a messaging server to support use cases such as a basic chat, multi party chats, persistent chats, etc. . . That messaging capabilities can be combined with the user management capabilities as well

Notifications Sippo wac handles the logic needed to properly notify users of incoming calls, new messages, new groups and other events, both for subscribers using a mobile device (via a push notification server) but also browser-based notifications.

Interoperability Sippo wac has been designed with a open interoperability mindset. That means that part of the functionalities provided by Sippo wac can be replaced by third party elements without impacting the rest of functionalities. This is specially important when customers wants to integrate any existing WebRTC element such media servers, WebRTC-to-SIP gateways, etc…

Whitelabelling and customization Sippo wac is fully white-labeled so each of our customers can provide it's own unique solution to the market.

Integration Most features are available through REST APIs so easy OSS/BSS integration and service provisioning

Deployment Sippo wac is a software-only solution that can be deployed in standard bare-metal server or in a Docker and/or Kubernetes environments, ready for cloud providers but also traditional on-premises deployments.

There are also a number of low-level functionalities that are described in detail in the following subchapters that make Sippo a quite unique solution to deliver complex applications to life in a very short time frame in a trusted, secure and scalable way.

2.1.4 Interfaces and APIs

We'll explain in this section the main protocols and interfaces supported by Sippo wac and which are the connections points to other network elements before explaining how Sippo wac fits into a existing architecture.

There are three main interfaces that connect a Sippo wac instance with the rest of the network and/or Internet:

SippoSDKs These are a set of software development kits (in short, SDKs), available on different programming languages, that allow developers to create their applications and leverage all of the Sippo wac functionalities using WebRTC standards. Sippo clients are the applications that use the SippoSDKs -in the browser on in mobile apps- to provide modern services such as audio and video calling capabilities, in-app messaging, multiparty video conferencing, etc… The SippoSDK are later introduced in detail in this guide and the complete documentation can be found online at Quobis support site.

SIP interface Beyond app-to-app or browser-to-browser communications, Sippo wac supports SIP protocol to interconnect with existing telephony and VoIP networks, both in enterprise and IMS-based service provider networks. That opens up a new range of use cases to make browser-to-SIP calls and app-to-SIP calls in addition to standard WebRTC-to-WebRTC calls, bridging both the telephony world and the internet world. That means that customers can extend the existing services provided on regular phones, PBXs, call-centers and other SIP-platforms and take them to websites and mobile applications.

REST API interface A powerful and flexible REST-based API is provided for OSS/BSS integration, system administration, user provisioning system configuration and monitoring. This REST API can be easily and securely consumed from third-party systems but also as from web and mobile applications, complementing the SippoSDKs. The most common uses for this API are user provisioning and management, adress book integration, call detail records collection, authentication and usage . This REST API is later introduced in another chapter and the documentation can be also found online at Quobis support site

Beyond the three interfaces above, Sippo wac supports other interfaces as listed below and that will be explained in the following chapters.

WAPI interface Interface used by SippoSDK applications to communicate, provide info and other runtime operations with the Sippo wac

Authentication interface Sippo wac supports OAuth2 protocol to delegate authentication into third parties

XMPP interface Sippo wac supports standard XMPP messages for IM

SMTP Sippo wac supports standard SMTP protocol for e-mail exchanging

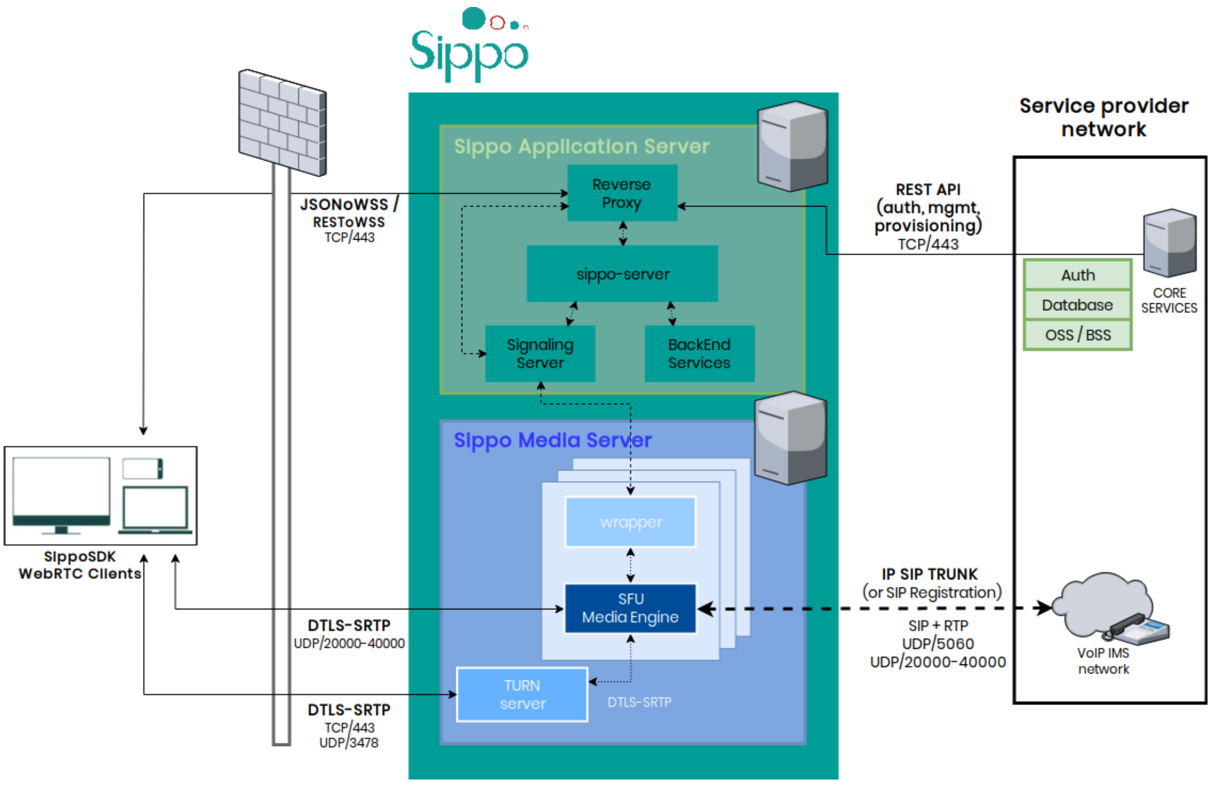
SNMP Standard monitoring and service availability"

2.1.5 Network architecture reference

Sippo wac has a flexible architecture which allows the setup of a wide range of network setups, depending on the use case that is going to be deployed. The power of Sippo wac is to work in coordination with other elements of the network, as it complements or improve some other network elements. This flexibility is one of the strengths of the product.

Sippo wac usually sits in the edge of the network, in close collaboration with other network elements both on the telephony area (such as PBXs, softswitches, but also on the IT area (web-servers, web-proxyes, firewalls, etc. . .).

The following picture describes a basic Sippo deployment. Please note that there is no "right" architecture as it's highly dependant on the use case. Quobis's presales team provides professional services to help in the design of the most appropiate architecture in terms of security, scalability, etc. . .



The main elements of this network diagram are:

Sippo wac Usually sitting in the edge of the network, this element is the core of the real-time application to be deployed.

SIP network This block is present when we need to connect to a existing PBX, ACD, softswitch, SBC or any other SIP element. Sippo wac has built-in SIP capabilities and thus can route call to the SIP network, via a standard SIP trunk or via a SIP REGISTER method. SIP authentication and SIP credentials management are also supported.

IT network This block is present when Sippo wac needs to interact with existing elements on the customer network for user authentication, user provisioning, CDR collection, addressbook integration, access control, monitoring, etc. . .

Web browsers The Sippo WebRTC applications are downloaded into the web browser. From the point of view of the subscriber or end-user, this is the only application that he/she will need to use. Sippo applications have HTTPS and Websocket connectivity with the Sippo wac.

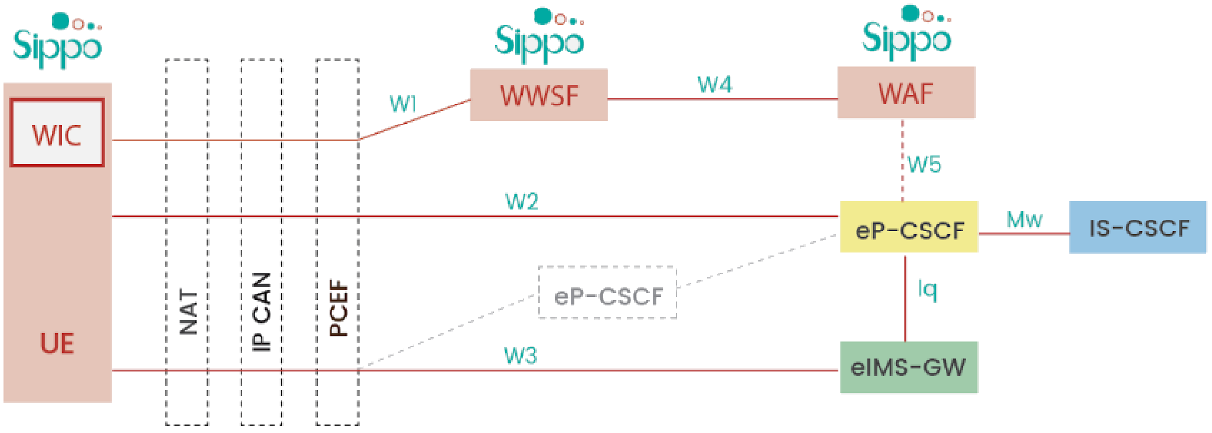
In a real deployment there are a number of additional network elements involved such a Session Border Controller, firewalls, STUN/TURN servers, SIP routers, etc… which will interact in some way with the WebRTC services and applications.

The reference network architecture responds to the standard IMS architecture where the WebRTC technology has its own place. Keep reading to understand the roles played by Sippo wac

2.1.6 Compliance with standard 3GPP architecture

The Sippo AS with the 3GPP 'WebRTC access to IMS' architecture specified in TS 23.228 Annex U (Release 13) and TR 23.701.

The following figure represents the general architecture proposed by the 3GPP, with the different elements provided by Quobis: WIC, WWSF and WAF



WIC (WebRTC IMS Client) Is an application using the WebRTC extensions specified in WebRTC 1.0 spec, as any of the WebRTC apps built using SippoSDK

WWSF (WebRTC Web Server Function) Is the initial point of contact in the Web that controls access to the IMS communication services for the user, that is one of the features of the Sippo wac

WAF (WebRTC Authorization Function) It supports the WWSF and eP-CSCF during the authentication and identity validation process.

eP-CSCF (P-CSCF enhanced for WebRTC) Provides the signaling interworking for WebRTC clients access to IMS, while acting as the P-CSCF towards the IMS core via the Mw reference point.

eIMS-AGW (IMS Access GateWay for WebRTC) Provides the media plane interworking capabilities such as DTLS-SRTP to RTP conversion, transcoding, and NAT traversal support including ICE.

2.1.7 Data channel and data pipe

Sippo wac media server (WebRTC gateway) has support for WebRTC data channel facilitating IMS integration with MSRP instant messages in the RCS context.

It also supports raw data channels (data pipe) which are based on websocket, not WebRTC data channels, using Sippo Application Server as proxy.

In Sippo SDK, *wacDataPipe* and *DataChannelDataPipe* have the same interface, one using data pipe and the other one data channel.

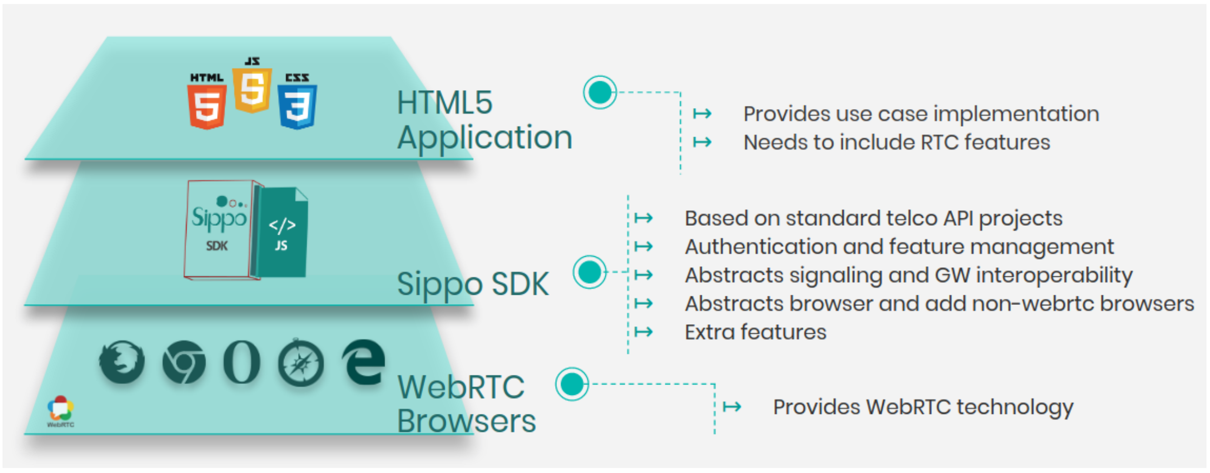
Note: Currently WebRTC data channel is only available with SIPoWS based deployment. See *Deployment introduction* for reference.

2.1.8 Developing applications with Sippo wac

Services providers and enterprises can deploy their own WebRTC applications and enhance existing ones using Sippo wac SDK's, leveraging of all the Sippo services like authentication, contacts, etc... There are three SDKS available as of today:

- SippoSDK for Javascript (also referred to as "SippoJS")
- SippoSDK for Android operating systems
- SippoSDK for iOS operanting systems

These SDKs can be also customized for service providers that want to expose their own set of SDKs to sell their services to small enterprises and developers to compete with existing SaaS providers of WebRTC APIs.


HTML
JS
CSS
HTML5 Application
↦ Provides use case implementation
↦ Needs to include RTC features
Sippo SDK
JS
Sippo SDK
↦ Based on standard telco API projects
↦ Authentication and feature management
↦ Abstracts signaling and GW interoperability
↦ Abstracts browser and add non-webrtc browsers
↦ Extra features
WebRTC Browsers
↦ Provides WebRTC technology

Every Sippo application runs connected to a Sippo Application Server, as some of the features are not implemented on the browser but on the Sippo backend. Both, the applications and the SippoSDK libraries can be hosted at the Sippo wac or at external servers.

Communication between the WebRTC applications running on the browser and the Sippo wac is done by using the WAPI interface, which dispatches the incoming messages to the corresponding services, as shown in the picture below:

The documentation and details of the SippoSDKs are explained online at http://doc.quobis.com, with a set of tutorials and guides to help you getting started and develop your first application.

2.1.9 GDPR and personal user data gathered

Sippo solutions doesn't store or hold any personal customer data and any data it does collect is not correlated with any particular customer. It's up to you, therefore, to determine what data you gather from your customers when you're defining the context variables you want to use in each engagement.

You're responsible for determining the amount of data required to define your engagement scenarios, and handle engagement routing, as well as determining what data should be relayed to your associates.

Note: Again, you have complete control over the data you choose to gather using Sippo solutions, how it is presented to associates, and how and if it is stored.

2.1.10 Frequently Asked Questions

Can I use Sippo wac as a WebRTC gateway?

Yes, you can. Sippo wac as its own WebRTC media server so you don't need an additional media server. Depending on the use case, you can also integrate a third party vendor. Quobis has a wide experience with many of the supported vendors and can provide the professional services required to deploy and configure all the elements involved in the system.

What is the maximum amount of concurrent sessions that Sippo wac supports?

Sippo wac has a micro-services architecture designed for high load scenarios. It can scale to thousands of concurrent calls and millions of subscribers when deployed correctly. However the exact number of sessions and simultaneous calls which can be supported has to be determined depending on the type and number of features which are being used, as there are ones that have much more impact than others.

2.2 Architecture and components

2.2.1 Internal architecture description

Sippo has a service-based architecture instead of the traditional monolithic architectures. These services are loosely coupled, independently deployable and organized around business capabilities. In order to deliver the business requirements, these services are containerized and inter-connected in a highly efficient and scalable way.

The services are deployed into software containers using Docker technology. A container is a standard unit of software that packages up code and all its dependencies, including everything needed to run an application: code, runtime, system tools, system libraries and settings.

Services can be scaled in/out individually, which means that we can have as many instances of those services as needed in order to meet the performance requirements, thus providing high scalability and high availability.

This chapter explains how these services are grouped into logical entities and the main functionality that they cover, so the sysadmin can understand the overall architecture that is independent of the actual container deployment. The detailed configuration of these components and its deployment and mapping into containers is explained in subsequent chapters in this guide.

First of all it is important to clarify terms and concepts before start. Sippo solutions are based on two main areas or sections:

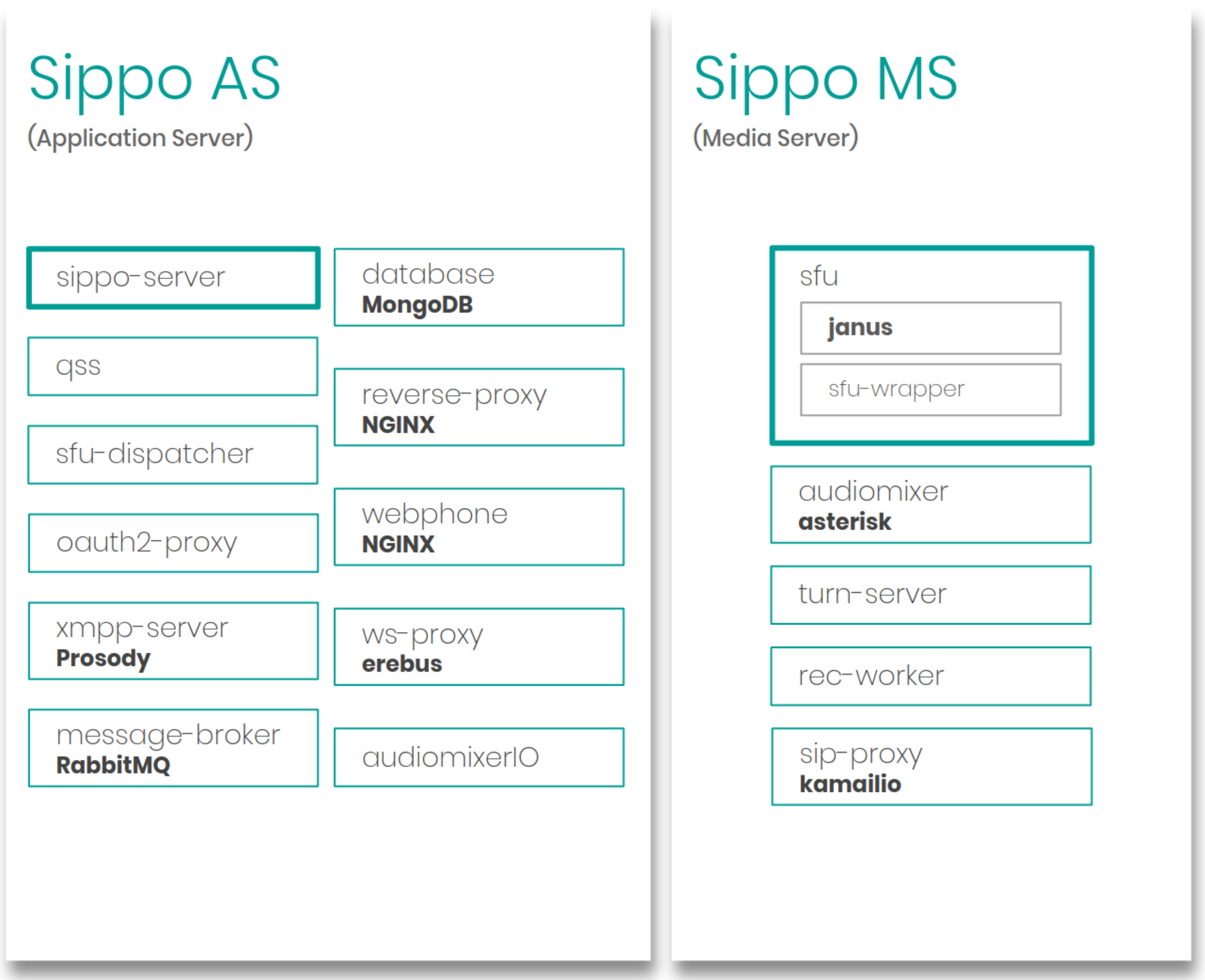
- *Backend architecture components*
- *User device application*

Let's define each element involved and its main role:

2.2.2 Backend architecture components

Sippo services are logically grouped into two main entities: the Sippo Application Server (SippoAS, in short) and the Sippo Media Server (SippoMS, in short). This is a logical separation that does not imply that the actual services are running at different machines or software containers.

Please note that not all the components are used on every deployment. Depending on the functionalities that are being used to fulfill the requested uses cases, some services might not be needed at all.



2.2.2.1 Sippo application server (Sippo AS)

The Sippo AS hosts the services that are responsible for the application logic, such as the basic real time services (audio calling, video calling, messaging, etc…), call management services (transfer, holding, call groups, routing, etc…) and also related services such as for example user management, authentication, contact management, file transfer, notifications, etc… The REST API and SDKs are also managed from the Sippo AS.

The table below shows all the components of the Sippo AS, details of its configuration can be found on subsequent chapters:

Sippo-server (SippoAS.sippo-server) This service is mandatory as it handles the user login and management, call management, contact management, authentication, etc…. It is also responsible for the communication with the SDK and the REST API.This service has been present since the initial version of Sippo and plays a central role as it handles features such as authentication, authorization, contact and CDR management. It also exposes and implements the logic of the Sippo REST API. In order to increase the performance and availability of a Sippo deployment, there can be several instances of SippoAS.ss running at the same time.

QSS (SippoAS.qss) This service name stands for “Quobis Signaling Server”. It is responsible for internal signaling management between the Sippo applications, the SippoAS and the SIP networks. The signaling exchanged between the Sippo applications and the QSS is a Quobis proprietary protocol and is capable of supporting complex scenarios such as multi-ringing, multi-device, multi-party conferences, etc… This component is not required when there is a third-party WebRTC gateway that has its own signaling stack. This component is composed of several subcomponents as well that are not listed here for simplicity but are addressed in subsequent chapters.

OAuth2-proxy (SippoAS.oauth2-proxy) This service handles the access delegation based on the open standard OAuth2, which is one of the most common authentication mechanisms. It allows SippoSDK applications to authenticate against the SippoAS and also plays the role of identity provider (e.g. Microsoft ADFS). It also allows the connection of non-standard OAuth2 access delegation systems.

XMPP server (SippoAS.xmpp-server) This service implements a XMPP server to handle asynchronous messages exchange between SippoSDK clients. It supports one to one and group chats, and communicates with a specific service within Sippo AS (SippoAS.sippo-server.xmpp) in order to synchronize groups, contacts, push notifications, etc…

Message-broker (SippoAS.message-broker) This service is responsible for the internal messaging between the SippoAS components (and, if enabled, SippoMS components) via a event-based queue. This communication is asynchronous, decoupling services by separating sending and receiving data and thus allowing to reduce loads and delivery times. It provides also reliability, persistence and delivery acknowledgements so no messages are lost in case of a disruption.

Database (SippoAS.database) This service implements a NoSQL distributed database. It provides horizontal scalability so it is well suited for web and transactional applications such as the ones developed with Sippo.

Reverse-proxy (SippoAS.reverse-proxy) This service, sitting on the edge of the network, implements a reverse-proxy and also a HTTP server. It serves as an entry point to every client application request in order to provide load balancing, protection from DoS attacks, caching and encryption.

Storage (SippoAS.storage) This service acts as a static storage for application assets and storage for user generated data like images, attachments, audio and video clips, etc… in a secure and distributed way.

SFU-dispacher (SippoAS.sfu-dispatcher) This service provides load balancing between several SFU services running in the SippoMS, allowing the system to scale correctly and to provide redundancy in case of system failure.

AudiomixerIO (SippoAS.audiomixerio) This service provides a input/output I/O interface for the for the Audiomixer. To grant the cluster architecture, a multi-host wrapper is required to grant movement of resources from one instance of the component to another in case of failure.

ws-proxy (SippoAS.ws-proxy) Internally knowns as `erebus` , this back-end websocket proxy acts as a API compatible versions gateway, allowing Sippo to achieve the API contract and help developers to migrate progressively their applications with the evolution of the API.

2.2.2.2 Sippo media server (Sippo MS)

The Sippo Media Server gathers all the services related with WebRTC and SIP media management, such as:

- Media termination (ICE/STUN/DTLS protocols)
- NAT traversal and TURN management
- Audio transcoding
- Audio and video recording
- Media quality management
- Media mixing
- Multi-party video (SFU architecture)

In addition, the Sippo MS also handles the interconnection with SIP networks providing a number of configurations to connect Sippo applications to existing SIP platforms, from a basic SIP trunk to complex registration scenarios.

Please note that SippoMS is an optional entity as the SippoAS can also work in a standalone way with third-party media servers without the need of a SippoMS. Check section Supported WebRTC gateways for more information on the supported vendors and how to properly configure SippoAS in each case.

The list below shows all the components of the Sippo MS, details of its configuration can be found on subsequent chapters:

SFU (SippoMS.sfu) This service creates multi-party conference rooms in a Single Forwarding Unit approach (SFU). Basically, it receives audio and/or video streams from each conference participant and forwards them to the rest of participants according to a set of rules managed in the SippoAS configuration. It also provides WebRTC media termination and WebRTC to SIP interworking

SFU wrapper (SippoMS.sfu-wrapper) This services provides a load-balancing of multiple SippoMS.sfu services for redundancy and scalability.

Audiomixer (SippoMS.am) This service provides the mixing of the audio streams and the signaling and RTP media interconnection with external SIP networks. It also provides transcoding capabilities.

TURN server (SippoMS.turn-server) This service provides connectivity tools in order to ensure real-time traffic exchange between peers independently from the network topology and intermediate firewalls by implementing the TURN protocol.

Recordings (SippoMS.rec-worker) This service does a post-processing of the raw video recordings generated by SippoMS.sfu so that they can be stored and consumed securely by third-party applications.

SIP-proxy (SippoMS.sip-proxy) Placed on the edge of the SIP interconnection, the SIP proxy provides a SIP engine able to grant security, optimal header adaptation and full integration with the SIP core of external services. It is the base element for SIP integrations, SIP header manipulations and SIP entry point to the system.

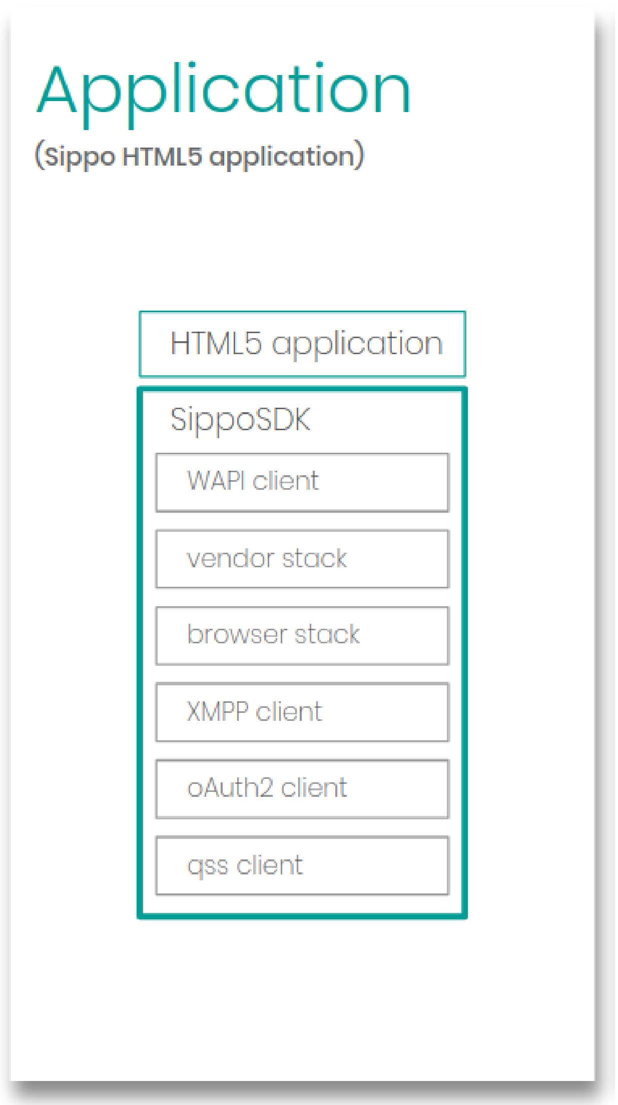
The diagram below shows a high-level description of the SippoAS and SippoMS components, its relationships and how they are usually connected to external networks. Please note that we have also included the REST API interface, SIP interfaces, SippoSDKs and external push notification systems.

Note:

Sippo products provide their own implementation for the Media Server role, but Sippo solutions support an extra set of leader manufacturers for this role in order to bring options for an optimal solution for your use case.

Check the section *Supported WebRTC gateways* for a full list of supported WebRTC gateways.

2.2.3 User device application



Let's consider any SippoSDK application, in this case called: *webphone*

HTML5 application The business logic application, the one that covers the interface between the user and the functionalities and cover the part that is not technologically related. It is the core of the *use case*.

SippoSDK The SippoSDK acts as an application abstract layer. It provides the ability to implement the same application with different WebRTC gateway vendors. This avoids the vendor lock and provides extra functionalities in order to have freedom to choose the better technical solution for each use case.

vendor-stack SDK provided by the WebRTC gateway. It provides connection between the application and the service infrastructure. Usually it handles the signaling part of the RTC protocols involved. Most of the times it is a private protocol.

browser-stack SDK provided by the browser to handle the media devices. It provides the functionalities required to obtain the media that is transferred to the other side/party in order to achieve a real time communication. It varies from different browsers and devices and require some kind of normalization to grant multi-device support.

2.3 Supported WebRTC gateways

Sippo is a vendor and signaling agnostic platform. This means that it can work in collaboration with different open-source and proprietary WebRTC gateways. Thanks to its abstraction layer, the SippoSDK contains a subset of signaling stacks to support several signaling protocols. Therefore is it possible to use the same Web and Mobile application with different gateway vendors by just using the version of the SDK compatible with each gateway.

On the section *Architecture and components*, this role is called Media Server (SippoMS). Check the corresponding section for a complete solution's components description.

Sippo solutions had been tested with a different WebRTC gateways. The gateways supported are included in the list below:

- **Open source solutions:**
  - Kamailio SIP router (version 5.0)
  - FreeSwitch (version > 1.6)
  - Asterisk (version > 15)
  - Janus Gateway (version > 0.6.3)[1]:
- **Vendor proprietary gateways[2]:**
  - Oracle Communications WebRTC Session Controller (version 7.2)
  - Nokia Enterprise Session Border Controller
  - Dialogic Powermedia XMS (version > 3.4)
  - Ericsson WebRTC Controller Gateway (WCG)

Note: Sippo solutions are compatible with **SIPoWS signaling** between WebRTC clients and SippoMS role. Any solution based on this protocol is expected to work properly with Sippo.

Specific configuration of both Sippo AS and the gateways can be found as dedicated application notes issued by Quobis and the corresponding vendor. In this manual the main configuration points common to all the vendors will be covered. For a complete list of interoperability tests, specific software version and capabilities of each vendor, please contact your sales representative at Quobis.

[1] To be able to use Janus as the WebRTC gateway is necessary to use also the QSS service to have some signaling mechanism. Janus gateway only handles the media so it requires an additional element to be able to handle the signaling.

[2] SippoSDK is compatible with SIP over Websocket (RFC7118), so in general it will be compatible with any proprietary gateway supporting SIP over Websocket even if they are not included in the list above. However some functionalities may not be fully supported so we recommend to ask Quobis if specific use case are covered.

CHAPTER

THREE

DEPLOYMENT REQUIREMENTS

3.1 Deployment introduction

Sippo is an on-premises solution, with a hard dependency of the use case for optimized performance. In order to deploy it, it is important to decide the correct deployment for your needs. Your Quobis representative will guide you through this selection process, but here is a list to understand all the Sippo options.

3.1.1 Type of deployment

Follow your Quobis representative to choose the correct deployment for your needs.

- **Based on docker containers**: Basic or mid-environments, HA, proof of concepts and small deployments. Isolated services running on containers, easy to manually switch and test between different versions and functionality schemes.
- **Based on Kubernetes**: Designed for cluster and high load deployments. Production ready with higher hardware requirements.

3.1.2 Architecture used

By default Sippo is deployed based on a *SFU architecture*, but your Quobis representative will guide you through this decision. Each vendor provides a different architecture.

- **SFU based**: Default deployment based on Janus SFU. Provides balanced performance for all use cases.
- **SIPoWS based**: Deployment based on Kamailio for high load and contact center deployments optimized for single 1-to-1 calls and recording.

Based on the previous decision the following scenarios will be offered:

1. Deploy on docker, SFU based
2. Deploy on docker, SIPoWS based
3. Deploy on Kubernetes, SFU based
4. Deploy on Kubernetes, SIPoWS based

3.1.3 Chapter contents

Following sections will guide you through the requirements for each deploy. Prepare them before go to next chapter and launch the installation procedure.

We will cover the following requirements prior to start with the installation procedure

- *Hardware requirements*: Virtual machine requirements, number of hosts and specs required

- *Network considerations*: A table to check with your networking team
- *Firewall and proxy considerations*: Some tips and definitions when solving network issues
- *HA and cluster network configuration*: Roles and architecture for HA environments

3.2 Hardware requirements

Sippo solutions are “software-only” deployments designed to run over any OS based on containers architecture. The different and required roles of the Sippo solutions could be deployed into one or several machines. Check *Deployment introduction* for extra information.

Despite of the solution is designed to be cross platform, Sippo has a series of limitations that recommend some specific hardware resources. On this section we will review the main hardware requirements to deploy the backend infrastructure: Sippo AS and Sippo MS.

Virtual environment tested:

- VMware vSphere ESX 5.1 or superior
- Amazon Web Service cloud
- Azure cloud

Base operative system tested:

- Debian 9

The following specifications refer to the hardware requirements of virtual machines:

3.2.1 Laboratory setup

For **lab setups (up to 25 concurrent calls)**, the strict VM minimum hardware requirements are:

Architecture	Intel x86 processors, 64 bits
Number of CPU cores	2x CPU core
Processor speed	2.0 GHz or higher
RAM	At least 8 GB memory
HDD	20 GB hard drive or higher
Network Interfaces	Single Ethernet 1000base-TX NICs
IP Addressing	One (1) IP address

Note: A single laboratory host merging all roles into the same machine (Sippo AS + Sippo MS) was tested to handle up to 25 concurrent calls

On this case we will use the **docker compose** deployment with the following considerations:

- It will be deployed based on docker using the repo: sippo-compose
- Installation will be done based on remote Ansible execution or OVA delivery based on less-effort basis
- Hardware (or virtual hardware) required:
  - At least 1 machine (all in one)

3.2.2 Production setup

For a **production environment**, please contact the Quobis sales team in order to review hardware specifications according to your estimated traffic usage.

The following architectures are available:

- **Laboratory**: single host (1 VM)
- **Production cluster**: cluster architecture (6 VMs)

Based on your defined architecture you will require one of more of the following hosts:

Architecture	Intel x86 processors, 64 bits
Number of CPU cores	4x CPU core
Processor speed	2.0 GHz or higher
RAM	At least 8 GB memory
HDD	100 GB hard drive or higher
Network Interfaces	3x Ethernet 1000base-TX NICs
IP Addressing	Three (3) IP address

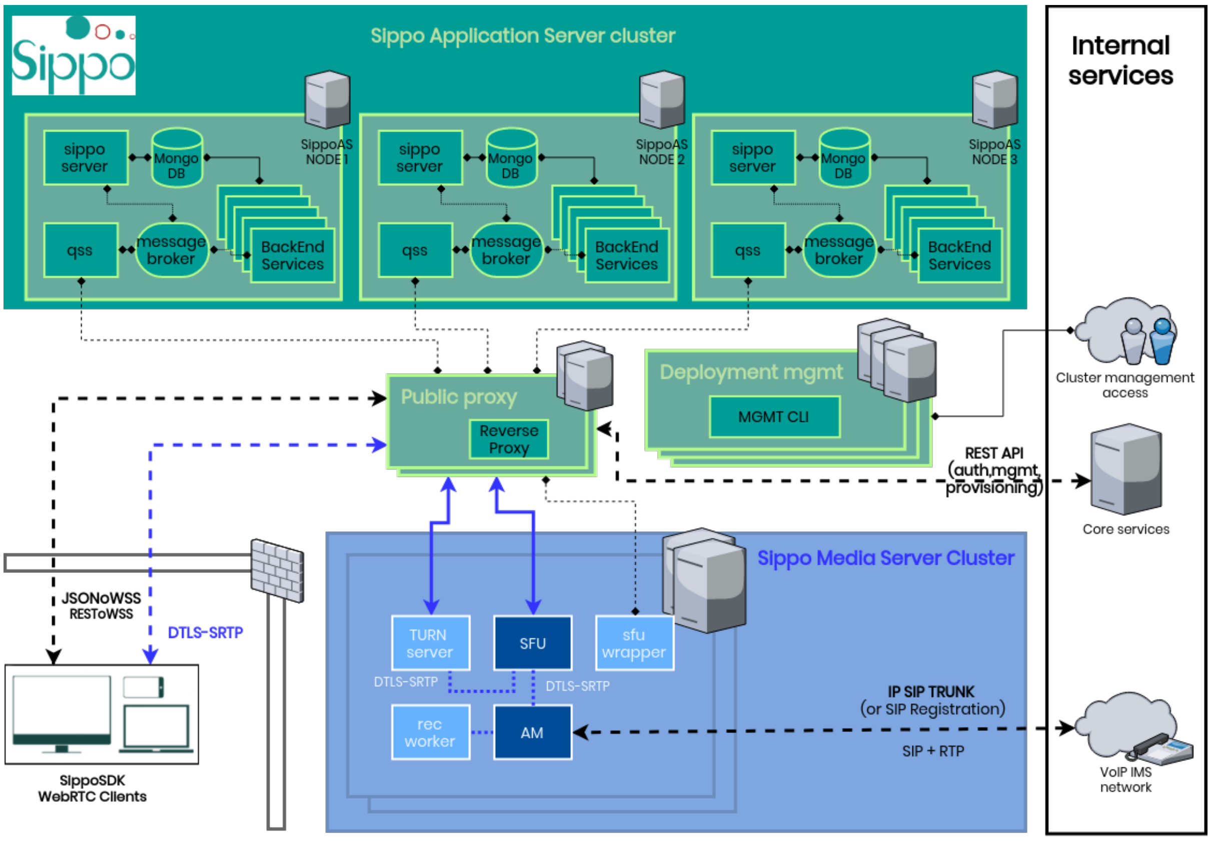
Next chapters will handle how to deploy and configure these machines and their corresponding roles.

On this case we will use the **Kubernetes cluster** deployment with the following considerations:

- It will be deployed based on Kubernetes using the repo: sippo-k8s
- It will be used both, for HA and stand-alone
- The customer is aware that any environment not in Kubernetes won't be official production, even in low-load scenarios
- The production installation will be always based on remote access install (Ansible), we won't send OVAs for production
- Hardware (or virtual hardware) required:
  - At least 3 machines (1 K8s master + Sippo AS + Sippo MS)
  - At least 6 machines for HA in AWS cloud environment
  - At least 8 machines for HA in VMware (hosted) environment

3.3 Network considerations

This is a cluster diagram example.



- Check extra information about HA and cluster details at *HA and cluster network configuration*.
- Check extra information about specific impact on firewall configuration at *Firewall and proxy considerations*
- Check extra information about network connections and cluster security considerations at *Cluster security management*

3.4 Firewall and proxy considerations

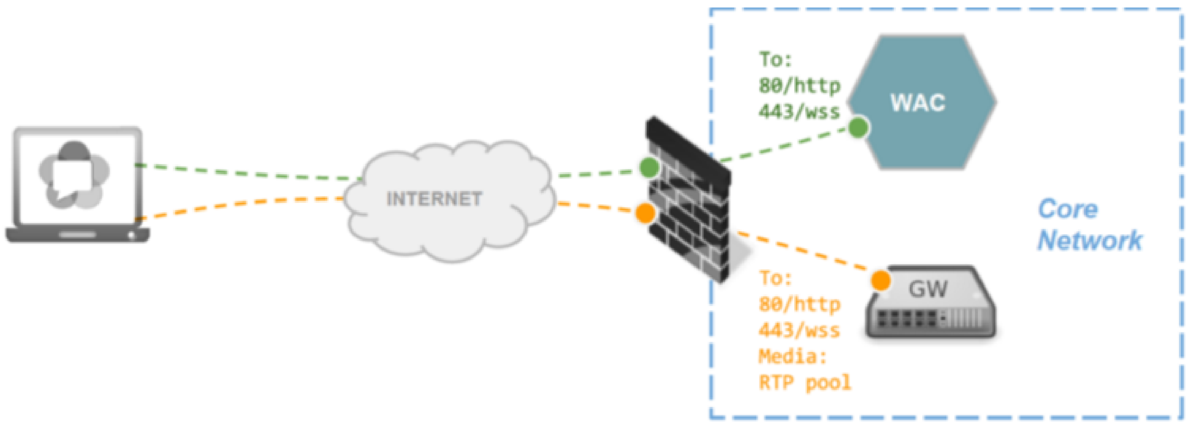
There are two common scenarios where a firewall is involved on deployment scenarios:

- A local firewall in front of the Sippo deployment
- A remote firewall in front of the WebRTC clients and applications

Let's check how they affect, and what is required to configure on our side.

3.4.1 Dealing with local firewalls (Server side)

It is a quite common situation to find a firewall in front of the Sippo deployment, both in service provider and enterprise scenarios. The following picture represents a typical use case:



The system administration will need to configure a specific port forwarding rules to make both Sippo AS and Sippo MS public ports accessible from Internet.

This deafult table will help to clarify which ports to open and why. Required information for network security management.

Table 1: Default network requirements

ORIGIN	ORIGIN PORT	DEST SERVICE	DEST PORT	PROTOCOL	DESCRIPTION	REQUIRED
ucclient	n/a	sippo-server	80	TCP	Application control traffic	Yes
ucclient	n/a	sippo-server	443	TCP	Application control traffic	Yes
ucclient	n/a	turn-server	443	TCP	Media traffic	Yes
ucclient	n/a	sfu	10000-18000	UDP	Media traffic	Optional
ucclient	n/a	audiomixer	10000-10999	UDP	Media traffic	Optional
turn-server	n/a	sfu	10000-18000	UDP	Internal media traffic	Yes
sippo-server	n/a	Apple APN	443	TCP	Push notifications traffic control for Apple devices	Yes
sippo-server	n/a	Android APN	443	TCP	Push notifications traffic control for Android devices	Yes
sippo-as	9021	sippo-ms	9030		Control traffic from SippoAS and SippoMS	Yes
sippo-as	9021	sippo-ms	9031		Control traffic from SippoAS and SippoMS	Yes
sippo-as	9021	sippo-ms	9038		Control traffic from SippoAS and SippoMS	Yes
audiomixer	5060	<customer SIP trunk>	<customer SIP port>	UDP/TCP	SIP signaling	Yes
<customer SIP trunk>	<customer SIP port>	<customer SIP trunk>	5060	UDP/TCP	SIP signaling	Yes
audiomixer	n/a	<customer SIP trunk>	<customer RTP ports>	UDP	SIP outoging media traffic	Yes
<customer SIP trunk>	n/a	audiomixer	20000-20999	UDP	SIP incoming media traffic	Yes

Some notes:

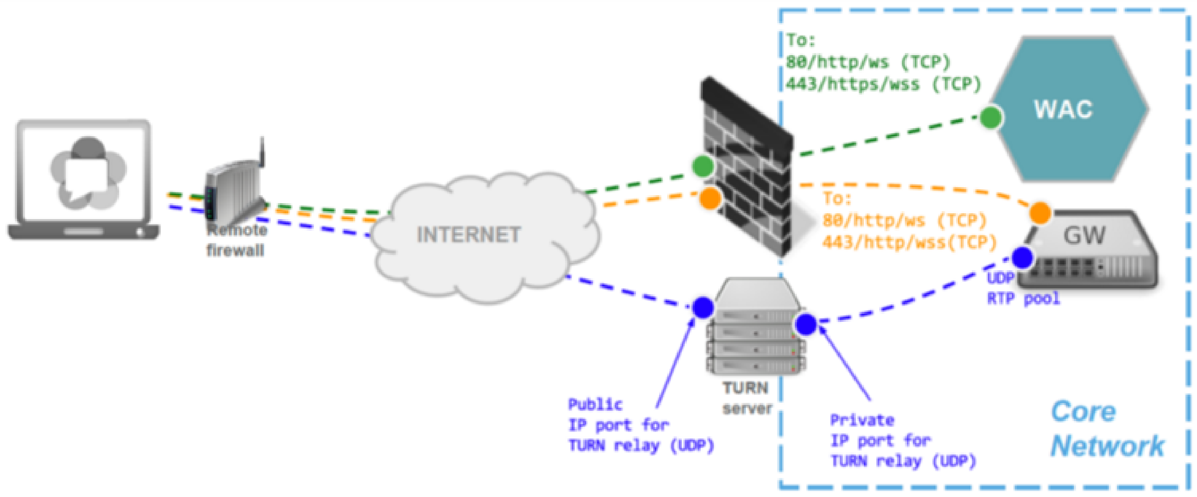
- Some WebRTC gateway vendors may require other ports to be open.[1]
- The UDP traffic could be configured on the WebRTC gateway. The traffic is secured due the DTLS-SRTP[2] and SRTP protocols.
- For STUN[3] and network discovery protocols, extra ports could be required. WebRTC gateways normally implement a simplification of ICE for NAT traversal called Lite ICE which requires to be listening on a public interface.

Note: The GW part of the diagram should be taken as a general reference as each vendor can present some differences in the recommended deployment.

3.4.2 Dealing with remote firewalls

It is also quite common to find remote firewalls blocking RTP or Websocket traffic, both in residential and business customers. In such scenario, where there is no possibility to change configuration on the remote firewall, a TURN server can mitigate some of those problems and specially the RTP media relay.

As the TURN server needs to relay media between two WebRTC endpoints (or the WebRTC endpoint and the PSTN), it needs a public IP address on its untrusted interface, as shown on the picture below:



From version 4.0 in above, Sippo solutions include TURN server features, it is included as part of the Sippo Media Server and you can find it defined at *Architecture and components* section.

A TURN server in the architecture can be helpful in the following scenarios:

1. We need to receive the media in our network and the WebRTC Gateway chosen only handles signaling.
2. We want to support scenarios where the the clients will be behind Proxies. Some browsers supports the method HTTP CONNECT to establish a TCP connection with the TURN server to exchange DTLS-SRTP traffic.[4]

[1] By using port 80 and 443 (for HTTPS and WSS) will minimize connectivity issues with users which are behind restrictive firewalls and proxies.

[2] DTLS-SRTP (RFC5764) is the protocol used to send the media traffic in WebRTC. It encrypts all the payloads using SRTP and the symmetric key is negotiated with RFC6347 Datagram TLS, which is a modification of TLS designed to be used over UDP.

[3] STUN (Session Traversal Utilities for NAT, RFC5389) is a protocol that serves as a tool for other protocols in dealing with Network Address Translator (NAT) traversal. It is mandatory in WebRTC and it allows to discover the public IP from which an endpoint is reaching a server in Internet. When using the Sippo AS, each subscriber will use a STUN server provisioned in the credential sections.

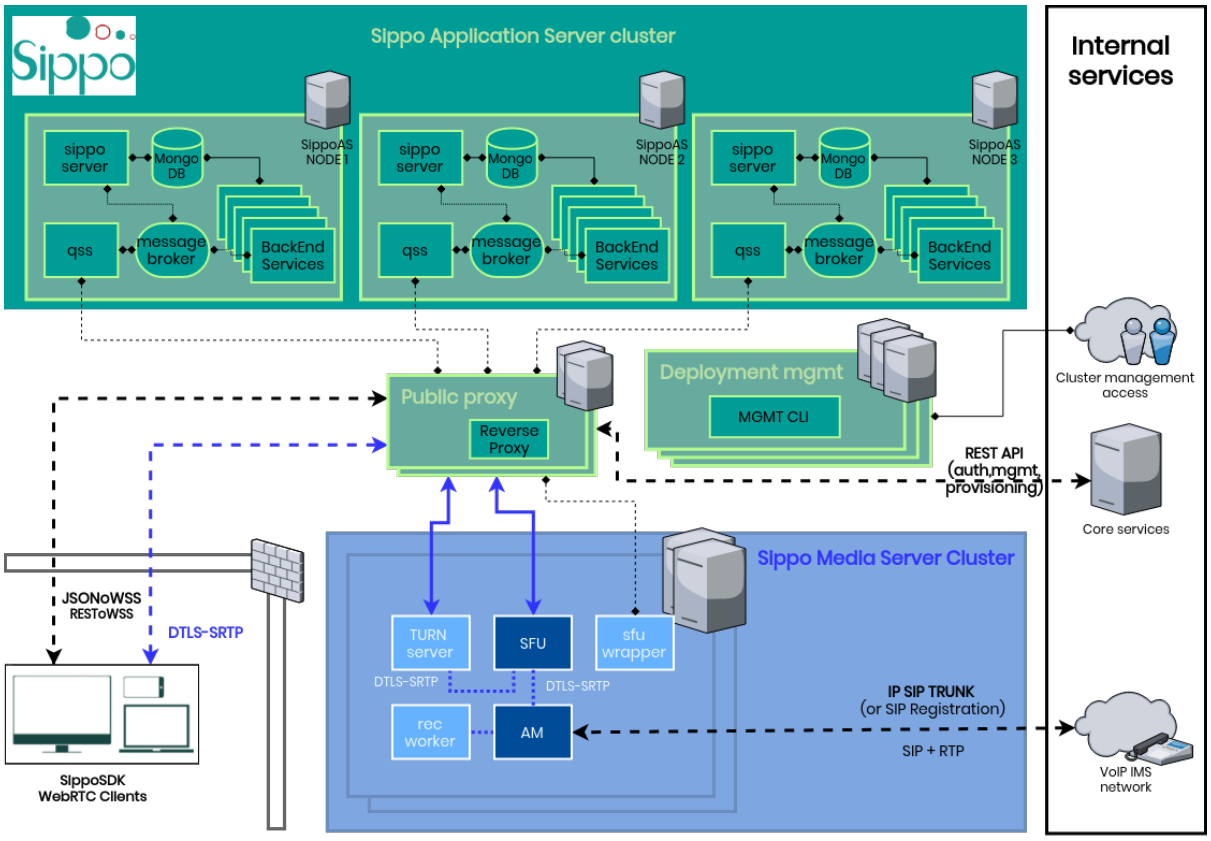
[4] This mechanism is defined in the The ALPN HTTP Header Field IETF draft. This mechanism is used and currently implemented by Google Chrome and Mozilla Firefox

3. We need to support scenarios where the clients are behind restrictive scenarios and they can only exchange DTLS-SRTP traffic through ports 80 or 443. Please note that this ports must be configured in the TURN servers to receive connections requests from the browsers.

3.5 HA and cluster network configuration

For several reasons, a cluster network should be necessary. One reason is capacity, to distribute load and expand system capacity. The second reason is to avoid the *single point of failure*. A typical high availability architecture is designed for active and stand-alone roles. Current state of the art discourages this kind of architecture since it wastes 50% of resources during 99% of time.

So Sippo AS is available with HA features under a cluster architecture form.



On this architecture, the network requirements are the same, but the roles are spread through different machines.

On the previous figure we can distinguish the following elements of the Sippo cluster environment:

Public proxy A web proxy that receives all the HTTPS requests from external systems and grant connection with the public side (Internet) and internal side (Enterprise or IMS core).

Deployment manager Hearth of the orchestration system, provides ability to modify the deployment adding or removing more nodes (computing power). It also handles the HA between the different working nodes and should be on HA itself (three machines at least).

Database engine Required for persistent data. This includes its own synchronization platform. This role could be played by a third party vendor if required.

Working nodes The Sippo AS role is played by these working nodes that work stateless, allowing two growing factors: scalability (add more nodes to add more load) and availability (on the event of a node failure, another one will replace it).

Media manager The media manager role is played by a dedicated cluster, it can grow on different patterns than the Aplication Server since each scenario will demand more or less media traffic. (For example, think on a chat only scenario).

Note: In order to understand different roles and names, visit *Architecture and components*

The main ideas under the cluster architecture are:

- Reliability: The **Deployment manager** is aware of the roles that turn down and can reply replicating them with new pods.
- Scallability: The architecture is flexible so new phisical hosts could be added as **working nodes** to the cluster and the Deployment manager could handle this increase of computing power replicating roles through the new hosts.
- All the outbound traffic is introuduced on the cluster through the **Public proxy**, this can control and distribute all the traffic between the different roles on the cluster.
- All roles are stateless and the persistent data is stored on the **Database engine** providing a high-availability environment
- The **media manager** is isolated from the cluster allowing third party ha mechanisms and allowing to control the traffic on their own ways.

3.6 Cluster security management

3.6.1 Anti-DoS measures

A Denial of Service (DoS) or a Distributed Denial of Service (DDoS) attack is an attempt to make a machine or network resource unavailable to users, and is intended to temporarily or indefinitely interrupt or suspend services to a host connected to the network resource.

These attacks typically involve saturating the target machine with external communication requests (flooding), so that the machine can no longer respond to legitimate traffic or does it so slow it can be treated as unavailable.

The defensive strategy implemented in Sippo involves applying a specific configuration at the cluster's entry point (nginx ingress) to ban IPs if request frequency is detected as abnormal.

Sippo allows to activate this feature and configure what will be considered as an abnormal request frequency for the same IP through the Ansible's variable definition file.

See *Variables definition* to configure this feature.

Note: This feature is only available through the Kubernetes based deployment.

Note: When requests exceed this limit a 429 code is returned.

3.6.2 CORS configuration

Cross-Origin Resource Sharing (CORS) is a mechanism witch allows a web application running at one origin to access selected resources from a different origin.

For the current version this configuration has migrated from the Sippo server service directly to the nginx controller and allows for two different scenarios:

- Restrict all traffic to a single domain name, intended for webphone deployed under a specific domain.
- Allow traffic for several domains (whitelist), required when crossed domains are used for different components.

See *Variables definition* to configure this feature.

Note: This feature is only available through the Kubernetes based deployment.

3.6.3 Password management

Sippo deployment procedure includes an encrypted password vault. This file contains the list of passwords required for the deployment.

Before executing the deployment the passwords must be encrypted and pasted into the file:

- `dockerPassword`. Used for accessing the docker repository.
- `tokenSms`. Token used by the *SendSMS service*.
- `wrapperPassphrase`. SFU wrapper password.
- `apnkeyId`. Used by Apple push notifications
- `apnTeamId`. Used by Apple push notifications
- `apnTopic`. Used by Apple push notifications
- `accesKeyId`. Used by the Kubernetes cert manager
- `secretKeyId`. Used by the Kubernetes cert manager
- `hostedZoneId`. Used by the Kubernetes cert manager
- `grafanaPassword`. Grafana password
- `smanPassword`. Sippo manager password
- `firebaseKey`. Firebase password.

Using this vaults ensures passwords are not stored in plain text format. The remaining passwords can be automatically generated and pasted into the vault using ansible command line:

- `xmpp_ps`
- `janus_admin_ps`
- `audiomixer_admin_ps`
- `sapi_ps`
- `postgress_ps`

For configuration reference see *Password definition*

CHAPTER

FOUR

SYSTEM INSTALLATION

4.1 Introduction

On this section we will guide you through the installation process. To achieve it properly you should:

- Understand the Sippo roles and elements. Check *Architecture and components* for extra information.
- Fulfill the hardware requirements. Check *Hardware requirements* for extra information.
- A defined deployment based on your solution requirements. Check *Deployment introduction* for extra information.

Based on that you will receive:

- A pre-installed and ready to run Sippo OVA.
  - Install the OVA following the instructions at *Sippo OVA Deployment*
  - Jump to *Initial steps after installation* to check your environment.
- A Sippo OVA among with an Ansible installer
  - Deploy the OVA to work as destination host or Ansible master host
  - Run the Ansible installer based on *Ansible orchestrator*
  - Jump to *Initial steps after installation* to check your environment.
- A standalone Ansible installer
  - Prepare your computer to play the Ansible playbook
  - Run the Ansible installer based on *Ansible orchestrator*
  - Jump to *Initial steps after installation* to check your environment.

Note: The standalone Ansible installer requires to prepare the whole cluster for the installation. Check with your Quobis representative the system requirements.

4.2 Sippo OVA Deployment

This is the fastest way to deploy a laboratory environment. The main goal is to deploy the Sippo AS and Sippo MS roles on a single virtual machine (VM). The idea is to simplify the networking and platform issues and bring the bare minimum elements online in order to test the software or any other feature under a Proof of Concept basis.

The deployment of a Sippo OVA has several phases:

1. Install or prepare your VMware virtualization platform.

2. Deploy the corresponding Sippo OVA.
   - Phases 1 and 2 are covered at *Appendix A. VMware OVA deploy*. It is a standard VMware OVA deployment. At that section a complete from bare-metal installation is covered.
3. Turn on machines, configure network and execute the Ansible script.
   - Section *Ansible orchestrator* will cover the setup of components if you require an extra update or a components change.
4. Ensure your installation is ready.
   - Section *Initial steps after installation* cover first powerup, initial network setup and configuration wizard execution.

4.2.1 Sippo OVA information

The OVA is a prefixed system image, your first task once deployed is to ensure that it is properly configured into the network. Please check information here in order to adapt it (network, hostnames, etc) to your needs.

Operative system

- Sippo OVA operative system is **Debian 9**

Network configuration

- **IP management address**: 192.168.10.2
- **Netmask**: 255.255.255.0
- **Default Gateway**: 192.168.10.1

Remote network access

Your Sippo instance will be managed from the system command line interface. Default port number for SSH service in Sippo OVA is 22/TCP.

In order to login into the Sippo instance, you can open the terminal console of the VM (remember that the layout is ES) or open a SSH connection to the default IP address of the system using the credentials shown above:

Note: If you are using the terminal console (directly on your VMware client), the provided default layout is es_ES.

System terminal user access

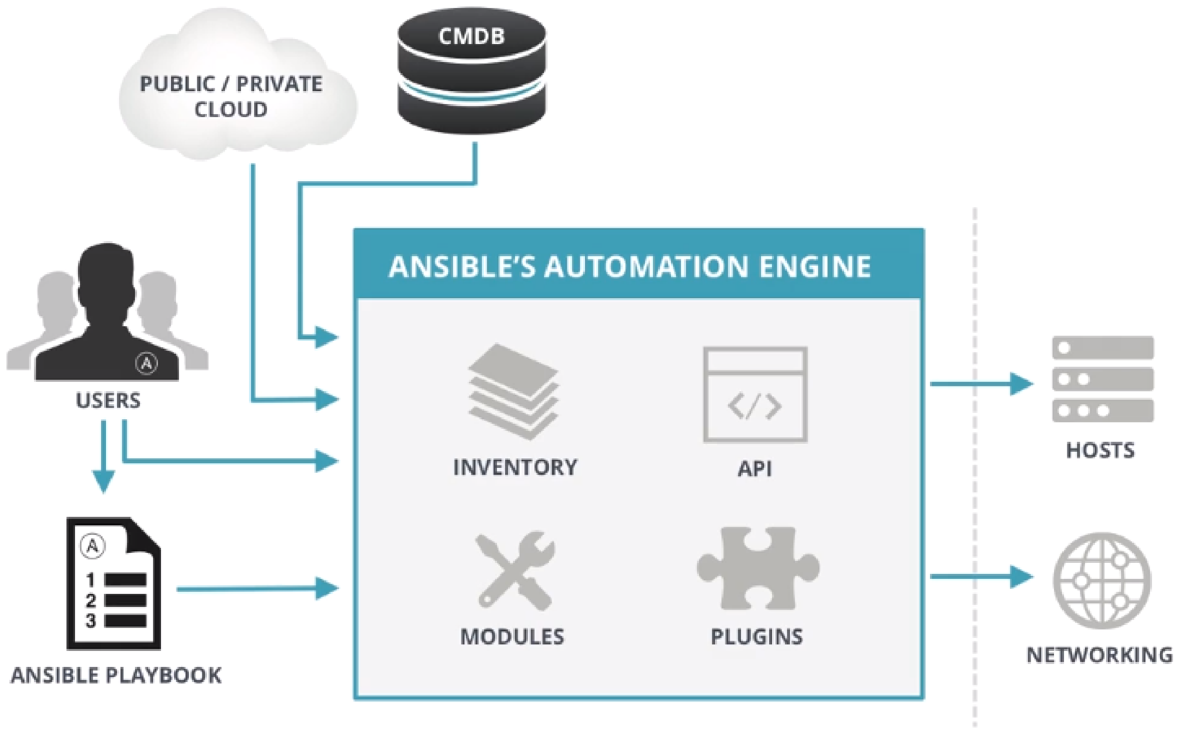
- **login**: admin
- **pass**: quobis

4.3 Ansible orchestrator

Contents

4.3.1 Basic concepts

Ansible is a task automation tool for system deployments. It will act as a wizard to setup the environment. Consider it as the *installer* of the Sippo solutions. The installer will be run only and once from a single host. That host will communicate and execute instructions over all the nodes involved on the Sippo network.



The main flow of the installation process is as follows:

1. Deployment of all hosts, manually, or using the Sippo OVA provided. Check *Introduction*.
2. Configuration of the installer (components and versions selection). There are two main installation methods:
   - Based on Docker for laboratory setups: *Docker based deployment*
   - Based on Kubernetes for production setups: *Kubernetes based deployment*
3. Execution of the Ansible script Check *Ansible execution*
4. Initial steps after Ansible script execution. These steps are required to validate your installation. Check *Initial steps after installation*.

4.3.2 Ansible installer requirements

Check the following requirements on the machine where you are going to run the Ansible installer:

- Supported OS: Debian 9
- At least 2GB of RAM
- Ansible > 2.3.1
- Ensure that all the target machines
  - **Have network visibility with the Ansible script installer and are reachable via a SSH connection.**
    * Includes `sudo` and an accessible user included on `sudoers`

Also you can use the Official Sippo OVA to use as master host.

Note: Ansible installation requires access to a container repository: https://registry.quobis.com. Contact Quobis representative to get your access token.

Depending on your deployment type, jump to *Docker based deployment* for laboratory setups or to *Kubernetes based deployment* for production setups.

4.4 Docker based deployment

Contents

Used for laboratory and all-in-one-machine deployments.

4.4.1 Ansible installer acquisition

You will receive the Ansible installer on a `tgz` package. Decompress the "tgz" package on the machine that is going to execute the Ansible installer (it doesn't need to be the production one).

4.4.2 Certificates installation

On next steps you must provide instructions to locate the certificates used to expose within your application. Prepare them before start. There is an specific folder named `/ssl/` on the Ansible installer, it will help you to have all ready before start since all configuration default values are pointing to this folder.

Save in `/ssl/`:

- *sippo-host.crt* , *sippo-host.key*: Public and private certificates to be used on the NGINX public interface. You main app entry point.
- *sippo-sapi-cert* , *sippo-sapi-key*: Public and private certificates to be used on the NGINX **internal** interface. Any self signed certificates will do the work. The `sippo-server` component uses it on `wiface1`.
- *APNsAuthKey_keyID.p8*. Apple Push notifications certificate required for `sippo-server` notifications.

Save in `/ssl/prosody/`:

- *apn.cer*, *apn.crt.pem*, *apn.key*, *apn.key.pem*. Valid certificate for Apple push services used by chat features.
- *voip_services.cer*, *voip_services.cer.pem*. Required certificates for Apple VoIP services.

4.4.3 Hosts definition

We are going to focus on the first main entry point of installation, the `hosts` file. It will help us to define the architecture and the node access/distribution.

```
$ vim hosts
```

You will find a YAML file with all entries enclosed on brackets `[]` that are referred as *hosts*. You may consider them *roles* as defined on *Architecture and components*.

This will customize your deploy on a very flexible way. For instance, if you have another database that is going to be used on this deployment, you should comment the `[mongodb]` host, or if you have you own entry point for the cluster, you can skip the Sippo load balancer. These kind of modifications must be always confirmed by a Quobis representative.

On each *host* you must indicate:

- The node's IP address for SSH connection (reachable from Ansible execution machine).
- `ansible_user` user name used by Ansible to log into the remote machine
- `ansible_ssh_pass` user pass used by Ansible to log into the remote machine
- `ansible_sudo_pass` sudo pass needed by the `ansible_user` to log as administrator on the remote machine

Note: Use `' '` to enclose variables that includes spaces, like passwords or file locations.

Here is an example of generic host definition and the fields that you must fill

```
[qss]
192.168.0.10 ansible_user=<remote_user> ansible_ssh_pass=<remote_pass> ansible_sudo_pass=
↪<remote_sudo_pass> ssh_key_file='/path/to/your/.ssh/id_rsa'
```

The `[nginxLB]` role is required only if you have several `sippo-server` instances on your deploy and need to balance the load between them.

```
[nginxLB]
192.168.0.12 ansible_user='<remote_user>' ansible_ssh_private_key_file=sandbox.pem
```

4.4.4 Variables definition

Let's go now for the variables definition. This file will allow to customize and modify all the internal parameters of the installation.

`group_vars/all.yml`

These are the mandatory fields that need to be modified:

- Container registry: `docker__registries` Indicate the username, mail and token previously mentioned for container repository access. *Ansible installer requirements*
- Certificates: Paths and filenames for exposed certificates. More info on the certificates section: *Certificates installation*.
- SIP trunk: Configuration of the SIP trunk interconnection. Check PSTN configuration section.
- **IP and network: There are several IPs fixed on this file, required for configuration, please add based on yo**
  - URIs: If you do not have a public URL quobis could provide you one <subdomain>.quobis.com and the corresponding certificates.

- Component versions: If you require some specific version of a component or to comment it, modify the value or comment the files to avoid installation.

Other fields could remain as they are. The file contains ad-hoc information about each parameter.

4.5 Kubernetes based deployment

Contents

Used for production environments, Kubernetes deployment is intended for cloud and production environments. It will allow us to scale and to handle high availability properly. There are different host requirements and extra considerations than the laboratory environment.

4.5.1 Ansible installer acquisition

You will receive the Ansible installer on a `tgz` package. Decompress it on the machine that is going to execute the Ansible installer. Not necessarily the destination production machine.

4.5.2 Certificates installation

On next steps you must provide instructions to locate the certificates used to expose within your application. Prepare them before start. We created a folder on the Ansible installer that will help you to have all ready before start.

Save in `/ssl/`:

- *sippo-host.crt* , *sippo-host.key* files: Public and private certificates to be used on the NGINX public interface. You main app entry point.
- *sippo-sapi-cert* , *sippo-sapi-key* files: Public and private certificates to be used on the NGINX **internal** interface. Any self signed certificates will do the work. The `sippo-server` component uses it on `wiface1`.

Save in `/ssl/prosody/`:

- *apn.cer*, *apn.crt.pem*, *apn.key*, *apn.key.pem*. Public valid certificate for Apple push services.
- *voip_services.cer*, *voip_services.cer.pem*. Public valid certificate for Apple VoIP services.

4.5.3 Inventory definition

In an scenario where you could have many different deployments the inventory feature allows to have configurations saved for each one of them.

Each inventory or configuration is stored under a folder under the `/inventory` directory. Within that folder should be:

- `host`
- `group_vars/all`
  - `main.yml`
  - `vault.yml`

4.5.4 Hosts definition

We are going to focus on the first main entry point of installation, the `hosts` file. It will help us to define the architecture and the node access/distribution.

```
$ vim hosts
```

You will find a YAML file with all entries enclosed on brackets `[]` that are referred as *hosts*. You may consider them *roles* as defined on *Architecture and components*.

On the Kubernetes deploy you will find only four types of hosts:

- `[master]` The machines where the `kubectl` will run. All the management is done from these machines. They are the *control panel* of the deployment. Three instances are required when deploying on VMware.
- `[node]` Working nodes, the ones that make the job. The handle traffic, run services, etc. From a minimum of 3 to your traffic needs.
- `[nginx]` If your system requires an external entry point, this is it.
- `[nfs]` Required only on AWS deployments, it will handle a distributed storage system.

On each *host* you must indicate:

- The node's IP address for SSH connection (reachable from Ansible execution machine).
- `ansible_user` user name used by Ansible to log into the remote machine
- `ansible_ssh_pass` user pass used by Ansible to log into the remote machine
- `ansible_sudo_pass` sudo pass needed by the `ansible_user` to log as administrator on the remote machine

Note: Use `' '` to enclose variables that includes spaces, like passwords or file locations.

Here is an example of a host definition file:

```
[master]
192.168.1.10 ansible_user=<remote_user> ansible_ssh_pass=<remote_pass> ansible_sudo_pass=
↪<remote_sudo_pass> ssh_key_file='path/to/your/.ssh/id_rsa'

[node]
192.168.1.20 ansible_user=<remote_user> ansible_ssh_pass=<remote_pass> ansible_sudo_pass=
↪<remote_sudo_pass>
192.168.1.21 ansible_user=<remote_user> ansible_ssh_pass=<remote_pass> ansible_sudo_pass=
↪<remote_sudo_pass>
192.168.1.22 ansible_user=<remote_user> ansible_ssh_pass=<remote_pass> ansible_sudo_pass=
↪<remote_sudo_pass>

[nginx]
192.168.1.30 ansible_user=<remote_user> ansible_ssh_pass=<remote_pass> ansible_sudo_pass=
↪<remote_sudo_pass>

[nfs]
192.168.1.40 ansible_user=<remote_user> ansible_ssh_pass=<remote_pass> ansible_sudo_pass=
↪<remote_sudo_pass>
```

(continues on next page)

(continued from previous page)

```
```

4.5.5 Password definition

Passwords are stored, encrypted, in `group_vars/all/vault.yml` file. Some of them must be introduced in this file manually. See *Password management* for the list of human input passwords.

Additionally, the remaining passwords can be included in this file using the following command to generate further commands to paste the generated keys into the vault:

```
ansible-playbook -i inventory/<inventory name> --vault-id <inventory name>@prompt main.yml --
↪tags "passwords-deploy"
```

Which should output something similar to:

```
Ansible command to encrypt password: ansible-vault encrypt_string --vault-id <inventory name>
↪@prompt 'HMdM2wUb2otnyuE5XagpStLgoTxJCMXgdZs25Tu4S8xW0IKIkps9oRhdNmlZbxkx' --name 'postgres_
↪ps' >> <inventory name>/group_vars/all/vault.yml"
```

Paste all the generated commands into the terminal to copy the passwords into your vault.

Password format example from the `vault.yml` file:

```
dockerPassword: !vault |
        $ANSIBLE_VAULT;1.1;AES256
        38346332323133396563643534643664653731626163646233323131386566303730316634643935
        6266326230386662393938346664313763396366313734640a393532613962363965646362653733
        32346334646536663763623839343064363534323031303538336263366165336333623831333337
        3438663762326532610a38313530353431393732306137313333396534643032333333462383633
        3935
```

4.5.6 Variables definition

Let's go now for the variables definition. This file will allow to customize and modify all the internal parameters of the installation.

`group_vars/all/main.yml`

The file will present a set of variables grouped by functions. You should expect the following:

4.5.6.1 Deploy variables

Indicate the username, mail and token previously mentioned at *Ansible installer requirements* for container repository access.

```
#
# Deploy variables
#
registryServer: registry.quobis.com
dockerUsername: <registry_user>
dockerEmail: <registry_user_mail>
```

4.5.6.2 Component versions

If you require some specific version of a component or get rid of them. Modify the default value, or comment the variables to avoid installation of that specific component on the node hosts.

```
#
# Version variables for docker Images
#
sippoServerVersion: 20.0.0
erebusVersion: 1.2.1
qssVersion: 4.6.0
sfuDispatcherVersion: 1.1.0
webphoneVersion: 5.4.0
webphoneStack: -janus
webphoneVariant: ''
webphone_base_href: '/' #If you add some path for example webphone, you need to put /webphone/
sfuWrapperVersion: 1.6.0
o2pVersion: 1.11.0
namespace: default
xmppVersion: 1.1.0
messageBrokerVersion: 3.7-management
pathInMaster: /home/quobis/ #Select the path where the kubernetes folder is
#update_machine: True
```

4.5.6.3 Certificates

Paths and filenames for exposed certificates. More info on the certificates section: *Certificates installation*.

```
#
# NGINX
#
serverName: sippo.quobis.com
nginx_key: sippo-host.key
nginx_crt: sippo-host.crt
nginx_sapi_key: sippo-host.key # -- NEW --
nginx_sapi_crt: sippo-host.crt # -- NEW --
nginx_official_repo_mainline: False
```

4.5.6.4 Monitoring and Sippo manager

Sippo deployments could include (under license extension) a monitoring and management tool called **Sippo manager**. This extension consist in a set of software monitoring tools to help monitor and administer the system. They will gather information from the system and alert the administrator about any possible problem. You can check extra information about these components at *Architecture and components*

```
#
# Sippo Manager
#
sippoManagerVersion: latest
# It's advisable to create a different user (admin role) for Sippo Manager (sman) to avoid the␣
↪sharing if sippo-server admin password
smanUsername: admin@quobis

#
# Grafana configuration
#
grafanaHost: smtp.gmail.com:465
grafanaUser: XXX@XXX
grafanaVersion: 6.4.3

#
```

(continues on next page)

(continued from previous page)

```
# Loki configuration
#
lokiVersion: 1.4.0

#
# Prometheus configuration
#
prometheusVersion: v2.13.0

#
# Sippo maintainer
#
maintainerVersion: 1.0-kubernetes1.15.3
deleteConferences: True
#Delete conferences with state finished and older than CALL_HISTORY_DAYS days. To enable this␣
↪feature, "deleteConferences" must be True
callHistoryDays: 7
```

4.5.6.5 SFU-Wrapper

On this section you only need to modify the SFU cluster configuration. If you have several hosts with this role, define a label and IP for each instance. Also select the preferred codec to use on the SFU. This will affect to the mobile platform support and recordings. Check with you Quobis representative if you need to change this field. Here is configuration example of this element:

```
#
# SFU Wrapper
#
codec: "vp8"
sfuWrapper:
  - labelWrapper: sfu1
    ipJanus: 192.168.1.60
  - labelWrapper: sfu2
    ipJanus: 192.168.1.61
  - labelWrapper: sfu3
    ipJanus: 192.168.1.62
  - labelWrapper: sfu4
    ipJanus: 192.168.1.63
```

4.5.6.6 File storage System

Based on your type of deployment, select if you need NFS or EFS. Comment the elements that you don't need as they wont be used on installation. Here is a configuration example of both options.

```
#
# NFS Server
#
nfsServer: 192.168.1.40
nfsPath: "/home/nfs"
```

```
#
# EFS Server
#
efsProvisionerVersion: v2.4.0
efsFileSystemId: fs-67dd9daf
awsRegion: eu-west-1
efsPath: "/home/nfs"
```

4.5.6.7 Database

For multiple instances of the database deployment set `replica:true` and use a comma-separated array on the `databaseUrl` parameter

```
#Single host DB deployment
replica: false
databaseUrl: "database.host.local"
```

```
#Cluster DB deployment
replica: true
databaseUrl: "database-node-1.{{ namespace }}.svc.cluster.local,database-node-2.{{ namespace }}
↪.svc.cluster.local,database-node-3.{{ namespace }}.svc.cluster.local"
```

4.5.6.8 XMPP server

This section holds information related to the XMPP server running within the Sippo cluster.

```
#
# xmpp-server
#
maxHistoryMessages: 50
mucLogExpiresAfter: never
xmppDatabaseVersion: 12.1-alpine
xmppDatabase: true
```

The server can either use internal storage or an external database. If `xmppDatabase` is set to *true* PostgresSQL will be used and a valid version must also be entered.

4.5.6.9 Sippo Media Server configuration

Required to specify the media server elements and its configuration. The Sippo Media Server components are deployed using docker as main element, so all elements SFU, SIP-proxy and Audiomixer are exposed to the network.

On the first step we must cover the WebRTC media engine configuration, version of the different components and the IP of the RTC exposed on the cluster.

```
#
# WebRTC - Asterisk and Janus
#
##############################################################################
# asteriskVersion:         Asterisk and registry image version              #
# janusVersion:            Janus and registry image version                 #
# webRtcIp:                Exposed Janus public IP to media stream          #
# webRtcStartPort:         Exposed Janus start port to media stream         #
# webRtcEndPort:           Exposed Janus end port to media stream           #
##############################################################################
asteriskVersion: 16.7.0
janusVersion: 0.9.2
webRtcIp: 10.1.21.88
webRtcStartPort: 20000
webRtcEndPort: 20499
```

On a second step, you must configure the SIP-proxy and the Audiomixer elements.

```
#
# SIP integration - Asterisk (media) and Kamailio (signaling)
#
```

(continues on next page)

(continued from previous page)

```
################################################################################
# announceOnlyUser:             Announcement if the user is alone in the      #
#                               conference room                               #
# rtpIp:                        Asterisk external media IP                    #
# rtpStartPort:                 Asterisk external start port for RTP          #
# rtpEndPort:                   Asterisk external end port for RTP            #
# sipIp:                        Kamailio external SIP IP                      #
# sipPort:                      Kamailio external SIP port                    #
# pbxIp:                        Customer endpoint SIP IP                      #
# pbxPort:                      Customer endpoint SIP port                    #
# pbxDomain:                    R-URI, FROM and TO domain to requests         #
################################################################################
announceOnlyUser: False
rtpIp: 10.1.21.93
rtpStartPort: 10000
rtpEndPort: 10249
sipIp: 10.1.21.93
sipPort: 5060
pbxIp: 10.1.3.2
pbxPort: 5060
pbxDomain: 10.1.3.2
```

4.5.6.10 Nginx ingress

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Includes two network protection related features: Anti DoS protection and CORS configuration.

DoS

- `ddosProtection`: Set to `true`, activates the functionality. Default is `false`.
- `rps`: In case the feature is activated, sets the limit to the maximum number of request per second from a given IP. Defaults to `5`;
- `rpsHigh`: Sets the maximum number of requests per second, but only to the web application endpoints. Defaults to `20`.

```
ddosProtection: false
rps: 5
rpsHigh: 20
```

CORS

- `corsDomain`: If not set it will only allow requests from `serverName` (from certificates configuration). Otherwise, you can define a list of domains separated by `|` or a regular expression.

```
corsDomains: "domain1.com|domain2.net"
```

Note: There shouldn't be any whitespace or the regular expression won't match.

Now you are ready to run the ansible installer. Jump to *Ansible execution*

4.6 Ansible execution

Contents

Once acquired your ansible installer and correctly configured every file corresponding to your deployment (*Kubernetes based deployment* or *Docker based deployment*), the next step is to execute it. To do this all machines involved in the deployment must have network visibility with the machine running the script.

Ansible works by executing a list of tasks on the remote machines. These lists of tasks, or plays, are contained in the **playbook**, and the network location of the machines involved is stored in the **inventory**.

For this installer the playbook is stored in `main.yml` and the inventory inside the `hosts` file, so the basic syntax to execute the installer is:

```
$ ansible-playbook -i hosts main.yml
```

4.6.1 Docker based deployment

This type of deployment is the most simple of the two. Just open a terminal inside the folder with the two files mentioned above and run the the command.

```
$ ansible-playbook -i hosts main.yml
```

4.6.2 Kubernetes based deployment

Kubernetes based deployment involves the use of tags in the ansible-playbook. Tags are a way of telling ansible which tasks it should perform and which not. By default, if no tag is specified, ansible will run all of them. The installer has the following tags:

Tags	Role
reset-cluster	Restart the cluster
deploy-cluster	Deploy the cluster
flannel-deploy	Deploy flannel instance as CNI
nginx-ingress-deploy	Deploy nginx as ingress controller
nfs-install	Install an nfs server in machine indicated in the hosts file
sippo-deploy	Deploy sippo-server, qss, sfu-dispatcher, xmpp-server, sfu-wrapper
sippo-destroy	Destroy sippo-server, qss, sfu-dispatcher, xmpp-server, sfu-wrapper
sippo-server-deploy	Deploy sippo-server
qss-deploy	Deploy qss
sfu-dispatcher-deploy	Deploy sfu-dispatcher
sfu-wrapper-deploy	Deploy sfu-wrapper
webphone-deploy	Deploy webphone
sfu-deploy	Deploy janus
audiomixer-deploy	Deploy asterisk
node	Join node
monitoring	Deploy MetricBeat
logging	Deploy EFK
create-registry-secret	Create secret to save docker registry credential
reset-cluster	To restart the cluster

Table 1 – c

Tags	Role
deploy-cluster	To deploy the cluster
flannel-deploy	Deploy flannel instance as CNI
nginx-ingress-deploy	Deploy nginx as ingress controller
nfs-install	Install an nfs server in machine indicated in the hosts file
sippo-deploy	Deploy sippo-server, qss, sfu-dispatcher, xmpp-server, sfu-wrapper
sippo-destroy	Destroy sippo-server, qss, sfu-dispatcher, xmpp-server, sfu-wrapper
erebus-deploy	Deploy erebus
erebus-destroy	Destroy erebus
sippo-server-deploy	Deploy sippo-server
sippo-server-redeploy	Re-deploy sippo-server
sippo-server-restart	Restart sippo-server applying changes in configuration files
sippo-server-start	Start sippo-server applying changes in configuration files
sippo-server-stop	Stop sippo-server applying changes in configuration files
sippo-server-update	Update sippo-server if the version in group_vars/all.yml was changed
sippo-server-check-config	diff config between repository and templates
qss-deploy	Deploy qss
qss-redeploy	Re-deploy qss
qss-restart	Restart qss applying changes in configuration files
qss-start	Start qss applying changes in configuration files
qss-stop	Stop qss applying changes in configuration files
qss-update	Update qss if the version in group_vars/all.yml was changed
qss-check-config	diff config between repository and templates
sfu-dispatcher-deploy	Deploy sfu-dispatcher
sfu-dispatcher-redeploy	Re-deploy sfu-dispatcher
sfu-dispatcher-restart	Restart sfu-dispatcher applying changes in configuration files
sfu-dispatcher-start	Start sfu-dispatcher applying changes in configuration files
sfu-dispatcher-stop	Stop sfu-dispatcher applying changes in configuration files
sfu-dispatcher-update	Update sfu-dispatcher if the version in group_vars/all.yml was changed
sfu-dispatcher-check-config	diff config between repository and templates
sfu-wrapper-deploy	Deploy sfu-wrapper
sfu-wrapper-redeploy	Re-deploy sfu-wrapper
sfu-wrapper-restart	Restart sfu-wrapper applying changes in configuration files
sfu-wrapper-start	Start sfu-wrapper applying changes in configuration files
sfu-wrapper-stop	Stop sfu-wrapper applying changes in configuration files
sfu-wrapper-update	Update sfu-wrapper if the version in group_vars/all.yml was changed
webphone-deploy	Deploy webphone
webphone-redeploy	Re-deploy webphone
webphone-restart	Restart webphone applying changes in configuration files
webphone-start	Start webphone applying changes in configuration files
webphone-stop	Stop webphone applying changes in configuration files
webphone-update	Update webphone if the version in group_vars/all.yml was changed
webphone-check-config	diff config between repository and templates
sippo-maintainer	Use sippo-maintainer to purge database and take actions over a k8s cluster
sfu-deploy	Deploy janus
audiomixer-deploy	Deploy asterisk
node	Join node
monitoring	Deploy Grafana, Prometheus, Loki, promtail, node-exporter, mongodb-exporter, kube-s
logging	Deploy EFK
create-registry-secret	Create secret to save docker registry credential
limit-range-deploy	Create a default limit range in configured namespace. Cpu max: 1, memory max: 1Gi,
routing-deploy	Create nginx routing to the minimum cluster stuff
oauth2-proxy-routing-deploy	Create nginx routing to oauth2-proxy
oauth2-proxy-check-config	diff config between repository and templates

Table 1 – c

Tags	Role
passwords-deploy	Create passwords to xmpp api, janus api, asterisk ami and sippo-server sapi
xmpp-server-check-config	diff config between repository and templates
check-config	diff config (oauth2-proxy, qss, sfu-dipatcher, sippo-server, webphone, xmpp-server) bet

As with the docker deployment, open a terminal inside the folder with both files(`main.yml` and `hosts`) to execute ansible. Use the following syntax to apply tags.

```
$ ansible-playbook -i inventory/<inventory name>  main.yml --tags "tag1, tag2, tag3"
```

```
$ ansible-playbook -i inventory/<inventory name> main.yml --skip-tags "tag1, tag2, tag3"
```

Note: It is mandatory to specify tags in kubernetes deployment. The installation will not work otherwise.

4.6.2.1 Clean installation

Lets suppose that it is the first time deploying Sippo with Kubernetes on the machines that you are using. The following steps will make a clean installation of Sippo.

1. Launch the cluster, only needed in virtual-machine not in cloud.

   ```
   $ ansible-playbook -i inventory/<inventory name> main.yml --tags "deploy-cluster"
   ```

2. Choose you file storage option between NFS in a baremetal deployment or EFS in an Amazon deployment.

   ```
   $ ansible-playbook -i inventory/<inventory name> main.yml --tags "nfs-install,nfs-deploy"
   ```

   or

   ```
   $ ansible-playbook -i inventory/<inventory name> main.yml --tags --tags "efs-deploy"
   ```

3. Deploy basic services and upload files.

   ```
   $ ansible-playbook -i inventory/<inventory name> main.yml --tags --tags "upload-files,
   ↪message-broker-deploy,database-deploy"
   ```

4. Deploy sippo-server, qss, sfu-dispatcher, xmpp-server, sfu-wrapper and and webphone.

   ```
   $ ansible-playbook -i inventory/<inventory name> main.yml --tags --tags "sippo-deploy"
   ```

4.6.2.2 Install sfu and audiomixer

Run the following command:

```
$ ansible-playbook -i inventory/<inventory name> main.yml --tags "pre-installation-docker, sfu-
↪deploy, audiomixer-deploy"
```

4.6.2.3 Add a new node to the cluster

The hosts file must be edited appropriately to identify the new node. Then run the following command:

```
$ ansible-playbook -i hosts main.yml --tags "pre-installation,node"
```

4.6.2.4 Add or manage only one node or set of nodes

Modify the hosts file so that the master is left intact, but only the nodes to be managed (added/reset) are kept.

```
$ ansible-playbook -i inventory/<inventory name> main.yml --tags "node"
```

4.6.3 Updating Configuration

On a deployed instance of Sippo WAC the administrator may want to change certain configuration of a particular service. This can also be accomplished through the Ansible orchestrator which allows modifications on a single container without redeploying the whole cluster again. To do so, specific tags should be used following a certain sequence.

1. Execute the `*-update` tag of the service you want modify. This will restore the container version if it was changed in `group_vars/all/main.yml`.
2. Apply the modifications to the configuration and templates of the service.
3. Execute the `*-restart` tag to restart the service and apply the changes made.

4.7 Other deployments

Sippo was installed successfuly on different platforms and environments for PoC, R&D projects and customer testing.

Please contact with Quobis representative to have instructions in order to have Sippo products deployed on the following environments:

- MS Azure
- HyperX virtualization platform
- Bare metal installation

4.8 Initial steps after installation

4.8.1 Status verification

Once all previous steps have been completed and all components have been installed, to check the deployment's status it is necessary to make sure each individual process taking part in the deployment is running.

On a kubernetes based deployment, these processes correspond with *pods* running inside the kubernetes cluster.

To get the list of deployed pods, on a terminal, run the following command:

```
$ kubectl get pods
```

This should output something similar to this:

```
NAME                                                          READY   STATUS    RESTARTS   AGE
database-node-1-rc-8zcxp                                      1/1     Running   3          45d
database-node-2-rc-62ncz                                      1/1     Running   5          17d
database-node-3-rc-qmzvj                                      1/1     Running   2          45d
efs-provisioner-database-node-1-poc-688bb4786d-p6s2g          1/1     Running   9          5d
efs-provisioner-database-node-2-poc-6bd58d655-wmllz           1/1     Running   3          45d
```

(continues on next page)

(continued from previous page)

```
efs-provisioner-database-node-3-poc-879d459db-sw25h         1/1     Running   18         12d
efs-provisioner-filesharing-poc-7bcfd6d848-bzwms            1/1     Running   1          45d
efs-provisioner-filestorage-poc-568899d484-bt2tz            1/1     Running   11         5d
efs-provisioner-filestorage-xmpp-server-poc-6fb5b75b9f-pnmkg 1/1    Running   0          8h
efs-provisioner-hosts-xmpp-server-poc-cfb496747-ddspb       1/1     Running   11         6d
message-broker-79944c564-6wkcr                              1/1     Running   0          8h
qss-auth-http-5d697d678f-n67tz                              1/1     Running   0          8h
qss-calltransfer-basic-865c4bc966-6kjd4                     1/1     Running   2          5d
qss-conference-state-95f4d9c78-mwlqx                        1/1     Running   0          8h
qss-invites-rooms-c68576695-f4hxf                           1/1     Running   1          5d
qss-io-websockets-547777bcfc-8lcsd                          1/1     Running   0          8h
qss-log-conference-76d74c9f75-5f2q5                         1/1     Running   1          5d
qss-meeting-basic-8569bc6f8-9flw7                           1/1     Running   0          8h
qss-peer-jt-597fdf9849-zd4hg                                1/1     Running   1          5d
qss-registry-authenticated-56b6c68fc9-c84mp                 1/1     Running   0          8h
qss-resolver-wac-6f74dd4864-zsb8w                           1/1     Running   3          5d
qss-rooms-basic-d68994894-qgszk                             1/1     Running   0          8h
qss-trunk-asterisk-746bf4f6c7-brbwx                         1/1     Running   1          5d
qss-watchdog-invites-6c7784478-8bx26                        1/1     Running   0          8h
qss-watchdog-registry-7b8bcdb8f7-xnf5d                      1/1     Running   1          5d
sfu-dispatcher-69875b884f-5rkkd                             1/1     Running   0          12d
sfu-wrapper-sfu1-d95b8b8f4-4djv8                            1/1     Running   0          8h
sippo-server-6b7c6f7684-lsxp5                               1/1     Running   3          5d
sippo-storage-65b9fb755b-8jjsl                              1/1     Running   0          18d
sippo-storage-65b9fb755b-gs2xb                              1/1     Running   0          8h
webphone-angular-587d965c9c-mv5p9                           1/1     Running   0          8h
xmpp-server-797f47dc67-cv5qf                                1/1     Running   0          6d
```

On this list you should be able to see at least one pod per role configured in the ansible script and all pods should have the **STATUS** column set to *Running*.

If some element is missing you should consider restarting the installation process making sure all steps have been followed correctly.

If one or more pods have a status different from *Running* this means those services have encountered an error. To to show details of a specific pod and related resources, on a terminal, run:

```
$ kubectl describe pod <name-of-the-pod>
```

The following is an example output for a describe of `sippo-server` pod:

```
Name:           sippo-server-6b7c6f7684-lsxp5
Namespace:      poc
Priority:       0
Node:           ip-172-32-33-233.eu-west-1.compute.internal/172.32.33.233
Start Time:     Thu, 26 Dec 2019 11:13:21 +0100
Labels:         app=sippo-server
                pod-template-hash=2637293240
Annotations:    <none>
Status:         Running
IP:             100.96.4.103
IPs:            <none>
Controlled By:  ReplicaSet/sippo-server-6b7c6f7684
Containers:
sippo-server:
    Container ID:   docker://dddc63acfc156b252af7958601f6245bfd27f43f744ab003d792d5794b2ef015
    Image:          registry.quobis.com/quobis/sippo-server:19.2.0
    Image ID:       docker-pullable://registry.quobis.com/quobis/sippo-
↪server@sha256:db316db5c79033fb7709f022b6c0d3dbe3c1ae6e938861c5c3cfe664c372f4bd
    Ports:          8000/TCP, 5678/TCP
    Host Ports:     0/TCP, 0/TCP
```

(continues on next page)

(continued from previous page)

```
    State:          Running
    Started:      Thu, 26 Dec 2019 11:14:34 +0100
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
    /home/nfs/filesharing from kube-nfs-pvc-filesharing (rw)
    [...]

Conditions:
Type                  Status
Initialized           True
Ready                 True
ContainersReady       True
PodScheduled          True
Volumes:
wac-config:
    Type:       ConfigMap (a volume populated by a ConfigMap)
    Name:       wac-config
    Optional:   false
[...]

QoS Class:        BestEffort
Node-Selectors:   <none>
Tolerations:      node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:           <none>
```

Also the following command can be executed to show the logs for the selected pod.

```
$ kubectl logs <name-of-the-pod> --tail 50 -f
```

Next is an example output for this command:

```
Thu Jan 02 2020 23:57:02 GMT+0000 (UTC) [761cf75f-4ddc-40dc-beea-ffc8bc23c08e~
↪hr4VcVoYQIEToJSBAAIf~5e0e625ecc8296cb78101f79~5dc5775c5b8a2934b2e39704#node@sippo-server-
↪6b7c6f7684-xxqwh<1>:/wac/lib/core/io/wapi/Wapi.js] debug: onWAPIMessage 877551, /sessions/
↪5e0e625ecc8296cb78101f79, PUT
Thu Jan 02 2020 23:57:02 GMT+0000 (UTC) [node@sippo-server-6b7c6f7684-lsxp5<1>:/wac/lib/core/
↪Sessions.js] debug: 100.96.4.22 5dc5775c5b8a2934b2e39704,  update session␣
↪5e0e625ecc8296cb78101f79
Thu Jan 02 2020 23:57:02 GMT+0000 (UTC) [761cf75f-4ddc-40dc-beea-ffc8bc23c08e~
↪hr4VcVoYQIEToJSBAAIf~5e0e625ecc8296cb78101f79~5dc5775c5b8a2934b2e39704#node@sippo-server-
↪6b7c6f7684-xxqwh<1>:/wac/lib/core/io/wapi/Wapi.js] debug: onWAPIMessageResponse 877551, /
↪sessions/5e0e625ecc8296cb78101f79, PUT
Thu Jan 02 2020 23:57:13 GMT+0000 (UTC) [node@sippo-server-6b7c6f7684-lsxp5<1>:/wac/lib/core/
↪io/wapi/Wapi.js] debug: client disconnection with session id 5e0e625ecc8296cb78101f79
Fri Jan 03 2020 00:00:49 GMT+0000 (UTC) [node@sippo-server-6b7c6f7684-lsxp5<1>:/wac/lib/core/
↪Sessions.js] debug: Remove session due expiration, [ '5e0e625ecc8296cb78101f79' ]
Fri Jan 03 2020 00:00:49 GMT+0000 (UTC) [node@sippo-server-6b7c6f7684-lsxp5<1>:/wac/lib/
↪services/Meetings/index.js] debug: onSessionDown: , { id: '5e0e625ecc8296cb78101f79',
token: 'Greup9ahrrswzNyb7j5KDXiyqLcdEUmB;',
user: '5dc5775c5b8a2934b2e39704',
userObj:
{ id: '5dc5775c5b8a2934b2e39704',
    domain: 'acme.com',
    username: 'user4',
    email: '',
    created: 1573222236772,
    lastLogin: 1577978596271,
    role: 'user',
```

(continues on next page)

(continued from previous page)

```
    capabilities: [ '+whiteboard', '+call-history' ],
    mobilePhone: [],
    landLineNumber: '',
    alias: 'b14edcef-406b-4c76-ac92-25d1j39d6f34' },
domain: 'anzen.com',
from: 1578000990073,
to: -1,
expires: 1578009602649,
registered: '5dc5775c5b1232934b2e39704@static',
context:
{ src: '100.96.4.22',
    useragent: 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML,␣
↪like Gecko) Chrome/79.0.3945.88 Safari/537.36',
    language: 'es-419',
    location: 'https://poc.quobis.com/webphone/index',
    platform: 'MacIntel',
    vendor: 'Google Inc.',
    deviceId: '' },
pushToken: '' }
Fri Jan 03 2020 00:00:49 GMT+0000 (UTC) [node@sippo-server-6b7c6f7684-lsxp5<1>:/wac/lib/
↪services/UserSettings.js] debug: user4@acme.com disconnected
```

CHAPTER

FIVE

SYSTEM ADMINISTRATION

5.1 Sippo user management

Sippo Application Server (Sippo AS) handles the management of subscribers, its capabilities and its classification into domains, supporting multiple tenants into the same deployment.

On this section we will cover the administration of the users and how to apply different permissions, domains, etc.

5.1.1 User management concepts

On this section, we will cover the following points:

Section contents

5.1.1.1 Sippo domains

Users on the system are organized on domains. The domains allow to spread capabilities and provide services on a limited way to a selected group of users.

Different use cases could take advantage of this concept, like different companies sharing the same Sippo instance, or maybe different offices of the same company.

Every resource on the Sippo system belongs to a domain, so there is a default domain configured on the system. There is a base domain called `quobis` but extra domains can be created based on your use case requirements.

- Check *Sippo domains* to expand information about domains, how to create, edit and remove them.

5.1.1.2 Unique userID (UUID)

Sippo users have a unique identifier, the *UUID*.

The *UUID* is defined as an unique identifier of a working element on the Sippo environment. It can refer to a user, a group or to any other entity that can be referred on the system. This is used to resolve usernames, alias and any other element user-friendly to refer to a functional element.

The *UUID* is composed by a hash number with a prefix `wac-user`. This will be used as argument on some REST API commands.

```
wac-user:5d8222a9aa655a69a10cb5c1
```

The *UUID* will correspond with a unique username and a domain. It uses a well-known URI representation:

```
<username>@<domain>
```

The Sippo userID is unique in the system, any request to create the same `username` on the same `domain` will be rejected.

- Check *Create and edit users* to expand information about users, how to create, edit and remove them.

5.1.1.3 User capabilities

Capabilities are used to identify features on the system that are granted or not to some user or group. These features could be conditioned by the backend platform, the device used by the user or just based on administration settings.

- Check *Users capabilities* to expand information about this topic.

5.1.1.4 How to use the administration commands?

The administration of the Sippo WAC is based on the REST API, all commands to handle the system must be done based on a rest API, you can use CURL or any other tool that allows you to consume the Sippo REST API (SAPI).

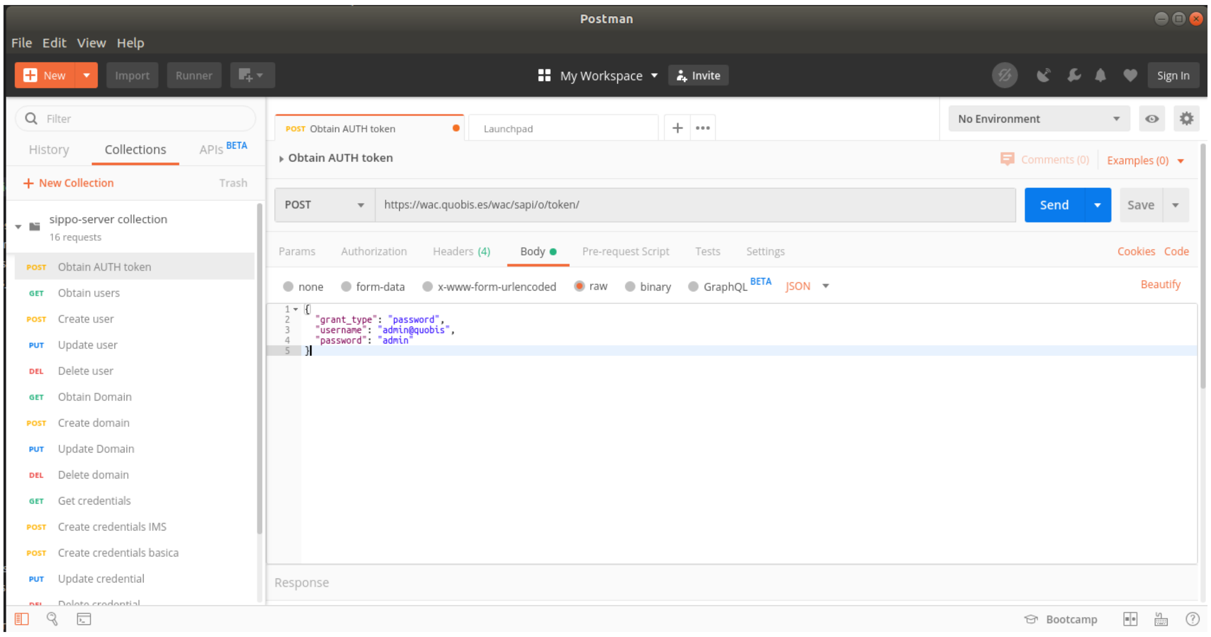
We present here two alternatives:

- **CURL**: to use it on bash commands
- **Postman**: to use it from a visual user interface

To run a query with **CURL** on a Bash script here is a sample:

```
curl -X POST https://wac.quobis.es/wac/sapi/o/token/ \
  -H 'Content-Type: application/json' \
  -H 'Accept: application/json' \
  -d $'{
      "grant_type": "password",
      "username": "username@domain",
      "password": "YoUr_Pa$sWoRd"
    }'
```

In case that you decide to use **Postman** or any other visual tool, check with your Quobis representative if he can provide a library to import all the available commands and have it ready for reutilization.



POST Obtain AUTH token

POST https://wac.quobis.es/wac/sapi/o/token/

```
{
  "grant_type": "password",
  "username": "admin@quobis",
  "password": "admin"
}
```

REST API authentication

All REST API requests must be authenticated using a Authorization header. Use the `/o/token` endpoint with a POST command to obtain it:

```
POST https://wac.quobis.es/wac/sapi/o/token/ \
    -H 'Content-Type: application/json' \
    -H 'Accept: application/json' \
    -d $'{
      "grant_type": "password",
      "username": "username@domain",
      "password": "YoUr_Pa$sWoRd"
      }'
```

Build your authentication header with the `token_type` and `access_token`.

```
Authorization: <token_type> <access_token>
```

Both fields are provided by the WAC as response if the user is valid:

```
{
  "access_token": "ffda919214b7d8cfafcb2c67fc4bf27a68f6120a4dd7de2ac2ce61009",
  "refresh_token": "2947926b49285a734a577a0343d6a824c0e77d52824cfd460df89042",
  "expires_in": 3599,
  "token_type": "Bearer"
}
```

Note: You can expand information about authentication on the Sippo WAC on the following section: *OAuth2-proxy - Federating identities*

Note: All this process can be simplified with the **Sippo manager** tool. Ask your Quobis representative about it.

5.1.1.5 User roles

We have three types of users (roles):

admin Used for reporting roles and management. These users will have access to all the content on the system and will not be limited by permissions.

user For standard use, calls, agents, etc. They are limited by user/domain capabilities.

anonymous It is the default type for any user created on the system. It is intended for external users that uses the platform for a limited scope of time or a single session. They are limited by the domain capabilities and other limitations imposed by the domain.

5.1.2 Sippo domains

On this section, we will cover the following points:

Section contents

5.1.2.1 Domains, routing and user visibility

The Sippo domain is the main framework or container of the Sippo users. Any user will be assigned to a domain. If is not specified, a default domain will be assigned.

By default (or based on contact configuration), the Sippo users of a domain have all the other users of the domain accessible to reach for presence and other information. Users from other domains are still reachable (you can dial and call them, using the `username@domain` unique identifier), but not directly accessible as other users from your own domain.

Any request on your application that involves an username without specify the domain will result on the resolution along your own domain.

Consider the following example. In the case that there are two users on the system: `bob@quobis` and `bob@acme`, if you as user of the `quobis` domain (`alice@quobis`) call to `bob`, it will be resolved as `bob@quobis`. You always have the option to call “the other Bob” if you dial directly to `bob@acme`.

5.1.2.2 Listing domains

An administrator user can retrieve the list of the Sippo domains on the system and the properties of it.

```
GET https://wac:8001/sapi/domains HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1ee
Content-Type: application/json

[
    {
        "id": "57877673e8f15d3472817349",
        "parent": null,
        "domain": "quobis",
        "origins": [
            null
        ],
        "enableAnonymous": false,
```

(continues on next page)

(continued from previous page)

```
        "anonymousExpiration": 0,
        "anonymousCapabilities": [],
        "services": {
            "callalarms": [],
            "callcontrol": [],
            "chat": [],
            "contacts": [
                "wac",
                "static",
                "google"
            ],
            "datapipe": [],
            "filetransfer": [],
            "forms": [],
            "im": [],
            "ldap": [],
            "login": [],
            "meetings": [],
            "oauth2client": [],
            "presence": [],
            "remotelog": [],
            "usersettings": [
                "wac"
            ]
        }
    }
]
```

Where the following fields could be found:

id Sippo UUID of the domain. To be used for other API operations.

parent In case that there is a hierarchy of domains, this field will indicate the parent domain. Parent domain limitations will be extended to the the son domain.

domain Name of the domain.

origins Used to map the domain specified on the `username` login body request with the anonymous credential assigned. Check *Anonymous users* to extend information about this topic.

enableAnonymous Anonymous login enabling option. Check *Anonymous users* to extend information about this topic.

anonymousExpiration Time in milliseconds anonymous users are allowed to perform operations on the system before they will be requested to refresh the authentication.

anonymousCapabilities List of capabilities of the anonymous users for this domain on the system. Check *Users capabilities* to expand information about this topic.

services List of services available on the system. Some ones will include extra details in order to define the backends included on it.

Getting details about an specific domain

Once you get the UUID of the domain, you can retrieve the details of your domain. The READ access to the user's domain details is granted for all users of the domain.

```
GET https://wac:8001/sapi/domains/57877673e8f15d3472817349 HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1ee
Content-Type: application/json
```

5.1.2.3 Creating a domain

On multi-tenant environments or based on the specific business needs, it could be convenient to have separated domains for different sets of users. This can be easily managed using the REST API to create the desired domains for the system.

```
POST https://wac:8001/sapi/domains HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1ee
Content-Type: application/json

{
    "domain": "demo.quobis.com",
    "parent": "quobis",
    "origins": ["demo.quobis.*"],
    "enableAnonymous": true,
    "services": {
        "login": [],
        "presence": [],
        "callcontrol": [],
        "contacts": ["wac", "static"]
    }
}
```

The previous example will create a new domain on the system that enables anonymous access and limit the services available to these users to the specified ones.

As you can understand it is specially interesting on use cases where anonymous users are involved. But also allows to create several domains where the users are not visible between them.

5.1.2.4 Update a domain

Typical use cases to update a domain is to grant a new service, remove it or to assign a new domain for the anonymous REFERRER header.

On this case, just use the PUT command with the UUID of the domain.

```
PUT https://wac:8001/sapi/domains/578895afe8f15d347281734f HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1ee
Content-Type: application/json

{
    "domain": "demo.quobis.com",
    "parent": "quobis",
    "origins": ["demo.quobis.com", "showcase.quobis.com"],
    "enableAnonymous": false,
    "services": {
        "login": [],
        "presence": [],
        "callcontrol": [],
        "contacts": ["wac", "static"]
    }
}
```

5.1.3 Create and edit users

We are going the cover this section the basic actions to handle the Sippo users: create, list, modify and delete. All these actions here can be complemented with other actions available on the REST API. Check the Sippo REST reference for a full list of available commands, parameters and answers.

Chapter contents:

5.1.3.1 Adding users

Create a user with a `POST` command using the `users` endpoint:

Note: Remember that you will need a valid authorization token to do any API REST action. Check *How to use the administration commands?* to expand information.

```
POST https://wac:8001/sapi/users HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1eef
Content-Type: application/json

{
   "domain": "quobis",
   "username": "alice",
   "email": "alice@demo.quobis.com",
   "role": "user",
   "capabilities": [],
   "mobilePhone": ["987654321"]
}
```

The following fields are available:

domain As explained on previous section, in case it is left empty, the default domain will be included.

username The username on the domain to be assigned to the user. If the user already exists the SippoAS will reject the user creation request.

role Administrator, user or anonymous. Check *User roles* for extra information.

email Context information of the user. An external email used to notify about meetings and other platform events that require the user attention.

mobilePhone [] Context information of the user. Array of external phone numbers to notify about meetings through SMS and other platform events that require the user attention.

capabilities [] Forced or pruned capabilities of the user. To limit or expand the functionalities available to this user. Check *Users capabilities* for extra information.

You should create a user for each user that access to the system. Maybe to use the Quobis Sippo collaborator, or for a custom WebRTC application, or maybe to consume the REST API, all users on the system should be identified.

Check users created at database

You can check the users created or obtain a complete list of them using the `GET` command with the same endpoint:

```
GET https://wac:8001/sapi/users HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1eef
Content-Type: application/json
```

Also, it is possible to get a single user to get full details over it. The user must be identified using the UUID as parameter for the HTTP request:

```
GET https://wac:8001/sapi/users/57877673e8f15d347281734a HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1eef
Content-Type: application/json
```

5.1.3.2 Editing users

As administrator, you need to update users, change its domain, or its personal data (mail, phone, etc).

To do it, simply execute a PUT command over the corresponding UUID of the user that you need to update:

```
PUT https://wac:8001/sapi/users/5788b1a8e8f15d3472817350 HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1eef
Content-Type: application/json

{
   "domain": "quobis",
   "username": "alice",
   "email": "imalice@demo.quobis.com",
   "role": "user",
   "capabilities": [],
   "mobilePhone": ['1111111'],
   "alias": "masterOfTheUniverse"
}
```

5.1.3.3 Delete users

To delete a user, simply use the `DELETE` command along with the `UUID`:

```
DELETE https://wac:8001/sapi/users/5788b1a8e8f15d3472817350 HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1eef
Content-Type: application/json
```

This action will simply remove the user entry on the system.

Warning: All other elements linked to this user, like credentials or any other element on the database will not be removed with this command.

5.1.3.4 Adding a password to an user - Basic credential

To log in the system. Any user will need an authorization token to use. On Sippo solution, the authorization tokens are defined as `credentials`. Also is common to refer to it as a “password” the information that confirms that we have a legitimate user.

A `credential` is simply a authorization password. There are three kind of credentials on the system:

- `basic`: Required for user authentication.
- `sip-mapping`: Presents the equivalence between a SIP identity and a system resource. Required to identify the user at the SIP gateway when the user reaches the external SIP network. Refer to *SipMappings* for a detailed description.

- `ims`: Required to identify the user on a third-party SIPoWS gateway.

Note: The `ims` credentials are used only when a third party SIPoWS MS is included instead the default SFU provided by Sippo.

The classic old school password is the `basic` credential, used by the SippoSDK application to obtain a token. Be sure to create a basic credential for each user that uses the system.

On this case, you will need to specify on the body of the request the UUID of the user to link with the credential:

```
POST https://wac:8001/sapi/credentials HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1eef
Content-Type: application/json

{
   "source": "wac",
   "type": "basic",
   "context": {},
   "domain": "quobis",
   "user": "5c4345a61f59a0026fa2929f",
   "data": {
      "password": "qa2"
   },
   "lease": 0
}
```

5.1.3.5 Adding SIP network access - Sip-mapping

In order to grant a user or *usergroup* to reach the SIP network through the SIP gateway on the Sippo MS, it is required to configure properly the SIP trunk and to add a `sip-mapping` to the resource.

To do it, add the username of the resource on the SIP network. This will be used to identify the user or *usergroup* on the SIP network.

```
POST https://wac:8001/sapi/sipMappings/ HTTP/1.1
     Authorization: Bearer ec3fe973ea04739e1...12ec5adc8b323c83c0
     Content-Type: application/json

     {
     "ownerUri": "wac-user:aliceId",
        "credential": {
           "username": "205"
           "password": ""
        }
     }
```

The previous example will create a `sip-mapping` that will identify the user as “205” on the SIP network. Also, any call received on the SIP side with 205 as destination will be routed to the corresponding user (identified by its UUID `wac-user:aliceId`).

5.1.3.6 Adding SIPoWS credentials - IMS credentials

When the deployment includes a third party SIPoWS or vendor SIP gateway playing the Sippo MS role (check extra info at *Architecture and components*), it is required to add extra authentication information to the system users.

For example, if you include a *Kamailio* playing the SippoMS role, and use the SippoSDK stack SIPoWS.

Consider it the login that must be done on the third-party element. Your user could be identified with a e-mail like username and a password, but these SIP gateways will need their own authentication framework. For example: you can be `alice@quobis` with pass `MyB1rdDay`, but you can have linked a SIP identity like `alice@quobis.com` with SIP authentication username `123@10.2.3.4` with a different password `23t6iyb3r5yliu`.

This link is made using the IMS credentials.

As commented, an authentication element on Sippo is a credential, so we will create the same request as we did for the basic password, but specifying the `ims` credential type:

```
POST https://wac:8001/sapi/credentials HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1eef
Content-Type: application/json
{
   "source": "wac",
   "type": "ims",
   "context": {},
   "domain": "quobis",
   "user": "586b6602ffc539dc7fc3261d",
   "data": {
      "username": "alice@quobiscom",
      "userauthname": "123@10.2.3.4",
      "password": "23t6iyb3r5yliu",
      "authserver": "wss://sipowsgw.quobis.com",
      "iceserver": [
         {
            "urls": "turn:158.61.103.18",
            "credential": "secret",
            "username": "alice"
         }
      ]
   },
   "lease": 0
}
```

user Required to link with the UUID of the Sippo user created previously.

data.username / **userauthname** / **password** Authentication information for SIPoWS related authentication.

data.authserver Address of the third party SIPoWS gateway.

data.iceserver Address and authentication information of the third party TURN server in place to avoid NAT problems.

lease Duration of the register. `0` will let the third party SIP gateway to define the expiration time.

5.1.4 Users capabilities

On this section, we will cover the following points:

Section contents

5.1.4.1 Introduction

The list of available features or behaviors of a SippoSDK application is determined by the resulting capabilities of the user session. Right after the user makes its first successful login, a set of **capabilities** are established.

There are 3 types of capabilities:

- **Service capabilities**: defined by the `sippo-server` configuration and enabled services.
- **Stack capabilities**: defined by the vendor of the signaling stack.
- **Session capabilities: defined by the wac-user or wac-domain configuration.** Where:
  - **Pruned capabilities**: defined by configuration to remove specific features per user or domain.
  - **Forced capabilities**: defined by configuration to add specific features per user or domain.

The SippoSDK application will receive a set of capabilities following next conditions:

- All capabilities are provided by active services (system functionalities).
- All capabilities are limited by the stack (client browser or app limitations).
- Administrators can add capabilities for extra features on the client side.
- Administrators can limit any present capability.

Every user is able to retrieve their platform capabilities. The SippoSDK application must adapt the user interface to the resulting status of the abilities granted. So, as a result, removing a capability, results on limit access to a feature.

```
GET https://wac:8001/sapi/capabilities/ HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1eef8

{
  "domain": [
    "stats",
    "alarms",
    "callcontrol",
    "chat",
    "contacts",
    "datapipe",
    "filetransfer",
    "login",
    "meetings",
    "login-token",
    "presence",
    "log",
    "settings",
    "recording"
  ],
  "user": [
    "[+c2cagent, +caller-extra-info]"
  ]
}
```

5.1.4.2 Service and stack capabilities

These capabilities define the available features for a user on an application:

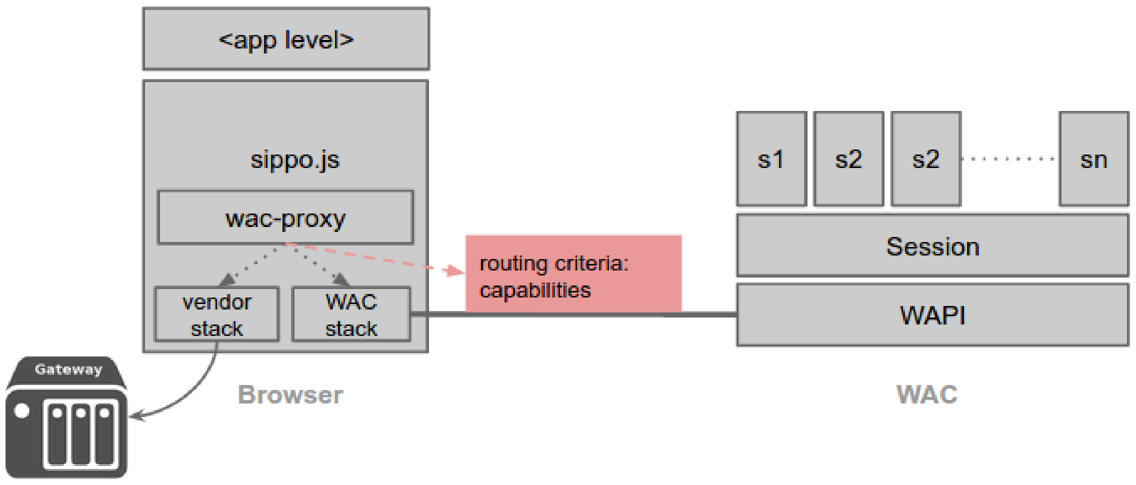
- **Service capabilities**: defined by the `sippo-server` configuration and enabled services.
- **Stack capabilities**: defined by the vendor of the signaling stack.

Depending on the configuration of the `sippo-server` and the stack used in each deployment, the set of available features in a Sippo environment will be different. The running services at the `sippo-server` determinate the functionalities available in the server side and provided by the `REST API`.

In the same way, some vendor stacks provide functionalities that are not available in others (like data channel or call transfer), maybe due to what the manufacturer uses as Media Server or due to platform limitations.

In the end, the set of available features is exposed to the applications on top of SippoSDK. The final application can use this to adapt the user interface (for example, do not show a call transfer button is the underlying stack does not support it), or for any other purpose (configuration or monitoring).

Some functionalities can be provided by the Sippo AS, others by the stack and others are provided by both elements. In the latter case the Sippo AS has a higher priority. The next picture represents how is the routing process. Whenever an application invokes a method on the SippoSDK, it is routed to the vendor stack or to the `sippo-server`. This decision is made in the `wac-proxy`, one of the internals modules of SippoSDK.



For configuration purposes. To generate the `session capabilities` the Sippo AS needs to know what services are running and create an array with the corresponding capability of each one. This is accomplished by the service `Capabilities`:

```
(wac.ini)
...
[capabilities]
...
```

5.1.4.3 Configurable session capabilities

In addition to the **Service capabilities** and the **Stack capabilities**, there are capabilities defined by configuration:

- **Pruned capabilities**: defined by configuration to remove specific features per user or domain.
- **Forced capabilities**: defined by configuration to add specific features per user or domain.

These are very useful to customize the behavior of some users or domains without affecting the other ones.

Forced capabilities are just strings for the `sippo-server`, that will be exposed to the applications without changes. An administrator is free to add extra configuration capabilities needed by the application. This way, any application can define new capabilities.

These capabilities are configured during the user provisioning process, by adding an array of capabilities to the user.

- Include a `+` as prefix to force (add) a capability to a user.
- Include a `-` as prefix to prune (remove) a capability to a user.

The following example will edit a user to include an extra capability (`caller-extra-info`) and to remove another one (`chat`).

```
PUT https://wac:8001/sapi/users/5788b1a8e8f15d3472817350 HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1eef
Content-Type: application/json

{
   "domain": "quobis",
   "username": "alice",
   "email": "imalice@demo.quobis.com",
   "role": "user",
   "capabilities": [+caller-extra-info, -chat],
   "mobilePhone": ['1111111'],
   "alias": "masterOfTheUniverse"
}
```

Note: The update command must include all the fields that you require to add to the user, not just the ones to update, so `username`, `domain`, etc. must be included on each `update` operation.

Note: Every application defines a set of specific capabilities that could be used by the application. Check the specific application documentation for a complete list of available forced capabilities.

5.1.4.4 List of session capabilities

Complete list of Sippo wac capabilities. Each SippoSDK application will use them in order to provide more or less actions to their users.

5.1.4.4.1 Configuration options

Capabilities pre-defined at SippoSDK level. Not related with the stack itself, neither with the services active at the sippo-server side. They are interesting to develop your own app with SippoSDK or when using the Sippo collaborator application.

call-from-chat Ability to call directly from the CHAT.

call-rehydration Ability to recover an existing call when refreshing page.

caller-extra-info Ability to access to extra context information about an incoming call.

change-camera Ability to change the camera during call.

create-contacts Ability to create contacts.

hide-incoming-domain Ability to display or hide the domain of the caller substituting it for the configured one.

make-calls Ability to make calls.

settings Ability to show a form in the client applications with some configurable parameters of the WAC or the CORE network.

view-call-history Ability to display call history.

view-contacts Ability to display contacts.

view-meetings Ability to display meetings.

whiteboard Ability to create a whiteboard session.

5.1.4.4.2 Service capabilities

Capabilities implemented based on services running on the "sippo-server". These will be active if the service is correctly configured and running in the back-end system.

call-alarms Ability to process received WebRTC stats data from the stack (stats-reporting) check defined alarms and trigger it if they match.

call-history Ability to access to the call history endpoint.

callcontrol CORE - Call routing and creation of one DB entry per call.

chat Ability to create a chat (proprietary protocol) - If service present at sippo-server it prevails.

contacts CORE - Ability to access to contacts endpoint.

credentials CORE - The WAC acts as a storage service for credentials.

datapipe Ability to create RAW data channels (based on Websocket, not WebRTC DataChannels) using Sippo AS as proxy.

filetransfer Ability to send files using WAC as proxy. Chat capability is required.

forms Ability to create user/agent forms to be filled by both.

log CORE - The SippoSDK sends local logs to the WAC for debugging purposes. Disabled by default.

login Ability to login in the WAC with basic authentication (username, password).

login-token Ability to login into the WAC by using a token-based mechanism, like OAuth2, which allows to use a third party authentication server.

meetings Ability to view / create / remove meetings.

presence Ability to receive presence updates about my contacts.

recording Ability to record a call (infrastructure mode).

sessions CORE - Sippo wac-session service is active on the Sippo AS.

stats Ability to process received WebRTC stats data from the stack (stats-reporting) and send it to the Sippo AS for processing.

5.1.4.4.3 Stack relying capabilities

Capabilities implemented at SippoSDK level on the specific media integration. We can face limitations on this section based on the vendor running on your environment.

attended-transfer Ability to transfer a call to a third user from the on-hold panel.

audio-call Ability to create an audio only call.

audio-video-call Ability to create a combined audio & video channel.

blind-transfer Ability to transfer a call directly to a third party user.

collaboration-call Ability to create a call without audio or video.

conference-hold Ability to hold a call when using a conference.

conferences Ability to create multi-endpoint calls.

datachannel Ability to create ciphered RAW data channels end to end.

geturi Ability to obtain a URI assigned by the gateway (`ims` credentials).

hold Ability to put a call on-hold.

im Ability to create a chat using the IM caps of the stack.

local-recording Ability to record a call directly on the user's browser.

media-update Ability to manipulate and change the current audio / video streams.

multicall Ability to have multiple 1-to-1 simultaneous calls.

network-reporting Ability to receive periodic network reports checking the call quality on real-time.

screen-sharing Ability to share screen and use it as video stream.

stats-reporting The underlying stacks reports statistics about the current call.

video-call Ability to create a video channel.

5.1.5 Anonymous users

On this section, we will cover the following points:

Section contents

5.1.5.1 Introduction

Sippo grants the ability to access to the system as an anonymous user. This is done to provide communication capabilities to applications or users that do not need to have persistency on the system. Typical use cases involves the sporadic use of a meeting platform or a customer care application from a call center company.

To activate this functionality there are some settings that are involved;

- The **Sippo domain** should provide a frame to include this users and route their call requests to the corresponding destination.
- If a third party SIPoWS or vendor Media Server is involved on the system, we need SIP credentials to register on the external SIP WebRTC gateway.

To obtain an authentication login for these users, an anonymous login request must be done.

```
POST https://wac:8001/sapi/o/token HTTP/1.1
Content-Type: application/json

{
    "grant_type": "anonymous",
    "username": "@quobis",
}
```

5.1.5.2 Domain options

Every anonymous request to the REST API, **must include** the `username` field with the format `@domain` including the desired domain to use by the application:

```
{
    "grant_type": "anonymous",
    "username": "@quobis",
}
```

That will specify the requested domain, but not the username. This info will be crossed with the existing domains and finally a domain will be assigned or the login request will be rejected.

So, to grant these users access to the system, at least, one domain should include the domain specified on the requested.

Note: To expand information about how to create and edit domains, check: *Sippo domains*

When you create or edit a domain, you should have these options in mind in case that you are required to grant anonymous access to the system.

origins Used to map the domain specified on the `username` login body request with the anonymous credential assigned. It could be an array. Any request done that includes the domain will try to select the corresponding domain, if not, the origins regular expression will be applied in order to fit the anonymous user on a valid domain.

enableAnonymous Anonymous login enabling option. In order to grant anonymous access, this option must be enabled. Any anonymous access will be granted to a domain that does not enabled this option.

anonymousExpiration Time in milliseconds anonymous users are allowed to perform operations on the system before they will be requested to refresh the authentication.

anonymousCapabilities List of capabilities of the anonymous users for this domain on the system. Check *Users capabilities* to expand information about this topic.

Of course you can limit the access the different services or functionalities to the different anonymous domains. Just create as many domains as your use case demands and assign different permissions schemas.

On the following example, we expose a domain that accept anonymous access for users that request access to the domains `"*.quobis.com"` or `quobis`:

```
{
    "domain": "demo.quobis.com",
    "parent": "quobis",
    "origins": ["*.quobis.com", "quobis"],
    "enableAnonymous": true,
    "anonymousExpiration": 600000,
    "anonymousCapabilities": ["-audio-video-call", "+chat"],
    "services": {
        "login": [],
        "presence": [],
        "callcontrol": [],
        "contacts": ["wac", "static"]
    }
}
```

The anonymous users accepted on this domain will be limited to the services listed, and also could have limited capabilities. In this case, the `audio-video-call` was limited but the `chat` enforced.

Note: Check extra info about capabilities at *Users capabilities*.

5.1.5.3 Anonymous credentials on SIPoWS environments

As explained before on the user creation section. It is needed to grant a pool of anonymous users when a third party SIPoWS Media Server is involved. The system will automatically link an anonymous user with a free IMS credential. The Sippo system could allow a user to access the system based on a limited domain, but not all WebRTC SIP gateways allow this kind of access.

On that cases, an anonymous IMS credential is required.

An **anonymous IMS credential** is defined as a SIP identity to be used by an anonymous user to be identified on a third party WebRTC SIP gateway.

So you need to previously provision a pool of anonymous IMS credentials. You can do it exactly as you do for IMS credentials (Check here: *Adding SIPoWS credentials - IMS credentials*) but skipping the link with the Sippo user.

```
POST https://wac:8001/sapi/credentials HTTP/1.1
Authorization: Bearer ec3fe973ea047b6fd38228eb9f9b9661cbb5b0fdeac01ff13af1eef
Content-Type: application/json
{
   "source": "wac",
   "type": "ims",
   "context": {},
   "domain": "quobis",
   "data": {
      "username": "alice@quobiscom",
      "userauthname": "123@10.2.3.4",
      "password": "23t6iyb3r5yliu",
      "authserver": "wss://sipowsgw.quobis.com",
      "iceserver": [
         {
            "urls": "turn:158.61.103.18",
            "credential": "secret",
            "username": "alice"
         }
      ]
   },
   "lease": 0
}
```

Any IMS credential that **does not have** a user linked, it is considered an anonymous IMS credential and will be linked to a valid anonymous sessions to be used if required by the SippoSDK stack.

Be sure to provision enough anonymous IMS credentials for your use case.

5.1.6 Login Access Control

Sippo includes several configuration options for changing how logging-in to the system works. This allows for a system administrator to change the way users create sessions after validating some provided credentials.

Section contents

5.1.6.1 Introduction

To allow users to create sessions, the *login service* must be enabled in the general configuration (enabled by default). This is done through the `[login]` entry within the `wac.ini` configuration file.

```
[login]
```

All configuration options relative to user login reside within the `login.toml` file, located in the `/wac` folder relative to the root directory. The following features will be configured in this file.

5.1.6.2 Login Alarms

A failed login always generate a 'Failed login' stat with service field set to 'Login'.

Login service can be also configured to generate an alarm when a configurable number of failed logins occurs in less than configured seconds.

Next example shows how to configure WAC to generate an alarm if an user fails to login 5 times in less than 10 seconds:

```
[login]
limitRateUnits = 5
limitRateSeconds = 10
```

5.1.6.3 Exclusive Login

Exclusive login is a feature that allows to configure the system to disable multiple simultaneous active sessions of the same user.

This means that a user which is already logged from a specific endpoint, **CAN** log in from another device but first, the currently active session will be dumped and a new session will be created through this last device.

Following, is an example of how this feature is enabled:

```
[login]
exclusive = true
```

Note: By default this feature is disabled.

5.1.6.4 OAuth2 Login

The following three parameters only make sense when an external OAuth2 provider is used for logging-in to the system. Chapter *SippoAS Service: OAuth2 client* explains how to configure the OAuth2 client to access external OAuth2 providers.

When a user tries to log in through the OAuth2 provider:

- `createOAuthUser`: Boolean which tells whether a user should be created on the fly (`true`) or not (`false`) with the information gathered from the OAuth2 provider. If set to `false` the system will try to find an existing user and, if not found, will respond with a failed login attempt.

- `OAuthUserForcedDomain`: Domain name which, if configured and `createOAuthUser` set to `true`, will be assigned unconditionally to all users created on the fly instead of using the email or any other field used by default to set the domain. Default value is `null`.
- `newUserCaps`: Comma separated list of capabilities to be set to automatically created users. Default value is an empty string.

```
[login]
createOAuthUser = true
OAuthUserForcedDomain = "example.org"
newUserCaps = "+call-history,+change-camera,+create-contacts"
```

Note: Sippo is configured to use its own OAuth2 provider by default, thus these features are disabled.

5.1.6.5 Other configuration options

5.1.6.5.1 sendStatic

Indicates if the login for any given user should return a static credential (`true`) or not (`false`). This should be true for scenarios with Janus gateway. With other gateways, such as SIPoWS, user are usually provisioned in the database as they are in the gateway.

```
[login]
sendStatic = true
```

5.1.6.5.2 requestHeader

String which indicates the name of the request header where to get the the real public IP of a client. If left as an empty string (`''`) then `'X-Forwarded-For'` will be used as the default header.

This parameter should be used in scenarios where a firewall or network element between the public internet and the server informs of the client's IP on a different header from the standard.

```
[login]
requestHeader = "Client-IP-Header"
```

5.2 OAuth2-proxy - Federating identities

This SippoAS feature allows the system to play the OAuth2 provider or client role.

OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 supersedes the work done on the original OAuth protocol created in 2006. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and

living room devices. This specification and its extensions are being developed within the IETF OAuth Working Group..

5.2.1 OAuth2 Protocol description and introduciton

We are not going to explain in detail the OAuth2 protocol, for a quick introduction into OAuth2 please refer to this document, which is nicer to read than the full OAuth2 specification.

The SippoAS is able to play as a provider or as a client:

OAuth2 client role

- it can relay the authentication process into an external authority (Google, LinkedIN, Twitter, etc).

OAuth2 provider role

- it can authenticate and validate trusted applications and users (Sippo manager, Sippo collaborator, admin profiles, etc).

5.2.1.1 Protocol flow

```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

This figure depicts the interaction between the different roles.

The process of obtaining the grant is defined in four variants:

- authorization code
- implicit grant
- resource owner password credentials
- client credentials

We do support all them four:

5.2.1.1.1 Authorization Code Grant

```
+----------+
| Resource |
|   Owner  |
|          |
+----------+
     ^
     |
```

(continues on next page)

(continued from previous page)

```
    (B)
+----|-----+          Client Identifier      +---------------+
|         -+----(A)-- & Redirection URI ---->|               |
|  User-   |                                 | Authorization |
|  Agent  -+----(B)-- User authenticates --->|     Server    |
|          |                                 |               |
|         -+----(C)-- Authorization Code ---<|               |
+-|----|---+                                 +---------------+
  |    |                                         ^      v
 (A)  (C)                                        |      |
  |    |                                         |      |
  ^    v                                         |      |
+---------+                                      |      |
|         |>---(D)-- Authorization Code ---------'      |
|  Client |          & Redirection URI                  |
|         |                                             |
|         |<---(E)----- Access Token -------------------'
+---------+       (w/ Optional Refresh Token)
```

5.2.1.1.2 Implicit Grant

```
+----------+
| Resource |
|  Owner   |
|          |
+----------+
     ^
     |
    (B)
+----|-----+          Client Identifier     +---------------+
|         -+----(A)-- & Redirection URI --->|               |
|  User-   |                                | Authorization |
|  Agent  -|----(B)-- User authenticates -->|     Server    |
|          |                                |               |
|          |<---(C)--- Redirection URI ----<|               |
|          |          with Access Token     +---------------+
|          |            in Fragment
|          |                                +---------------+
|          |----(D)--- Redirection URI ---->|   Web-Hosted  |
|          |          without Fragment      |     Client    |
|          |                                |    Resource   |
|     (F)  |<---(E)------- Script ---------<|               |
|          |                                +---------------+
+-|--------+
  |    |
 (A)  (G) Access Token
  |    |
  ^    v
+---------+
|         |
|  Client |
|         |
+---------+
```

5.2.1.1.3 Resource Owner Password Credentials

```
+----------+
| Resource |
|  Owner   |
|          |
+----------+
     v
     |    Resource Owner
    (A) Password Credentials
     |
     v
+---------+                                  +---------------+
|         |>--(B)---- Resource Owner ------->|               |
|         |         Password Credentials     | Authorization |
| Client  |                                  |     Server    |
|         |<--(C)---- Access Token ---------<|               |
|         |    (w/ Optional Refresh Token)   |               |
+---------+                                  +---------------+
```

5.2.1.1.4 Client Credentials

```
+---------+                                  +---------------+
|         |                                  |               |
|         |>--(A)- Client Authentication --->| Authorization |
| Client  |                                  |     Server    |
|         |<--(B)---- Access Token ---------<|               |
|         |                                  |               |
+---------+                                  +---------------+
```

5.2.2 SippoAS Service: OAuth2 provider

Since the need of securing the accesses to our REST APIs, we set up an OAuth2 provider in the SippoAS to control those accesses.

The goal of using OAuth2 is to provide these key features:

- **User Authentincation**: authenticate the users who use the API
- **Resource profiles**: restrict access to specific resources depending on user's profile

For a quick introduction check *OAuth2 Protocol description and introduciton*. Or check the full OAuth2 specification.

5.2.2.1 What SippoAS OAuth2 provider does?

This SippoAS service plays three roles of the OAuth2 authorization framework:

- **Resource Owner**: the SippoAS itself will grant or not access to the resources
- **Resource Server**: we also are who host the resources
- **Authorization Server**: the SippoAS will also be responsible of issuing the tokens

5.2.2.2 Implementation

The SippoAS OAuth2 provider `[oauth2provider]` exposes a REST API that uses PassportJS to enforce the access restrictions through a wrapper added in the `[wiface]` service. So the services provided are limited by a security profile.

In addition, the oauth2orize library provides the framework for deploying an OAuth2 server.

5.2.2.3 Configuration

The configuration of a service involves initializing it in the `wac.ini` file and some new sections to be added to the `"servicename".toml`.

As the other services, `OAuth2Provider` has its section in the `wac.ini` file:

```
[oauth2provider]
```

and his own `oauth2provider.toml` file describing its configuration:

```
[oauth2provider]
wiface = public
maxTokens = 0
accessTokenExpiration = 3600
refreshTokenExpiration = 1209600
```

wiface - Default: `public` The interface to add the endpoints to.

maxTokens - Default: `0` **(unlimited)** The maximum simultaneous tokens issued per user. Could be used to establish the max number of endpoints for a single user.

accessTokenExpiration - Default: `3600` **(1h)** The seconds an access token is valid for. An access token is needed for the client to access the service.

refreshTokenExpiration - Default: `1209600` **(2w)** The seconds a refresh token is valid for. A refresh token is used to obtain an access token without client credentials input.

Note: While in **maxTokens** `0` stands for unlimited, in **accessTokenExpiration** and **refreshTokenExpiration** `0` means instant expiration, which would make the service useless

Note: Also make sure the selected `wiface` has defined a **secure port**. This service will refuse to work on plain HTTP.

5.2.3 SippoAS Service: OAuth2 client

5.2.3.1 What SippoAS OAuth2 client does?

This service provides other services the ability to register OAuth2 clients so they can make use of third party services that require the OAuth2 protocol to be used for authentication and authorization.

Used by *OAuth2-proxy - Federating identities*

5.2.3.2 How it works

For a quick introduction check *OAuth2 Protocol description and introduciton*. Or check the full OAuth2 specification.

1. **Service start**

   When the SippoAS OAuth2 client service starts, it first registers the callback URL (named internally `redirect_URI`) into the `wiface` given in the configuration file.

2. **OAuth2 provider registration**

   That callback will be sent to the OAuth2 provider so it knows where we are listening for their requests. In addition, the callback URL (`redirect_uri`) will probably need to be registered in advance into the provider (this is the case for Google, where no random `redirect_uri` are allowed).

3. **Ready to go**

   Once the callback URL is set up, and the OAuth2 provider is aware of our callback URL, the service is ready to receive new client registration requests.

5.2.3.2.1 Token validation `validateAccessToken()`

Triggered by a login process, it validates an access token against the given provider service. It receives two arguments, both of them of type `string`:

1. The provider name to validate the access token against to.
2. The access token to validate.

The return value is of type `promise` which resolves with the token information or rejects with an error.

The first thing done is to check for the providers name . If the provider is not configured validation is rejected, otherwise it extracts the necessary information from the provider for validation. The token is then validated against the provider and if the response succeeds and has no errors the function returns correctly and login continues.

5.2.3.2.2 Authorization token storage: `_saveToken()`

This method is the responsible of storing the authorized tokens and associated information into the credentials storage of the SippoAS for later usage. We keep the tokens in our database so subsequent requests to the provider's services doesn't need the user to authenticate each time. Only when the token expires we will show again the authentication process.

When the authorization information is stored, an event is fired so the services that where waiting for the OAuth2 process can react and start using the third party services.

5.2.3.3 Configuration

The service requires two configuration steps: it needs to know the SippoAS interface it has to be attached to and the path for the providers JSON files. It is configured through the `wac.ini` file. A sample configuration may be:

```
[oauth2client]
wiface = public
secure = true
providers = config/oauth2/providers
```

- **wiface**: the interface to add the endpoints to. Required.
- **secure**: listen for secure connections at specified wiface. Defaults to false.
- **providers**: path to a directory with JSON files for each available OAuth2 provider.

5.2.4 OAuth2 provider: Obtaining an OAuth2 token

Some times, for testing, development or administration purposes, we may need to obtain an OAuth2 token from the SippoAS (OAuth2 provider) in order to access its REST APIs.

The easiest way to do so is to execute an special HTTP request as we were an authorized client asking for such a token.

5.2.4.1 Requirements

1. A valid user credentials (from `authdb`)
2. A registered OAuth2 client
3. Any HTTP client (we'll be using the generally available `curl`)

5.2.4.2 Process

This is a one step process (given the requirements are satisfied). We have to execute this HTTP request:

`POST /sapi/o/token`

```
POST http://wac:8001/sapi/o/token HTTP/1.1
Content-Type: application/json; charset=utf-8

{
  "grant_type": "password",
  "username": "our_valid_username",
  "password": "our_valid_password"
}
```

To make it easier to read, we have used the form urlencoded format as you can see in the `cURL` command:

```
% curl -d "grant_type=password&username=xxxxxxx&password=xxxxxx" -X POST https://wac:8001/sapi/
↪o/token
{
  "access_token":
↪"abf3073ab081731f542edc2d221d26fc979dc2038052f3730f05f84cf269f995b5d403bca654d79aa6e7e3d6e363c171c2c89d5cd92d
↪",
  "refresh_token":
↪"6774659d347164f315c2d772baef1475c0ac508eebc5431b626cb4c2df04f66177145a0743d56ae35e24e66f1635da35d3f2736fc74c
↪",
  "expires_in":3599,
  "token_type":"Bearer"
}
```

And we can see the response with the token we want.

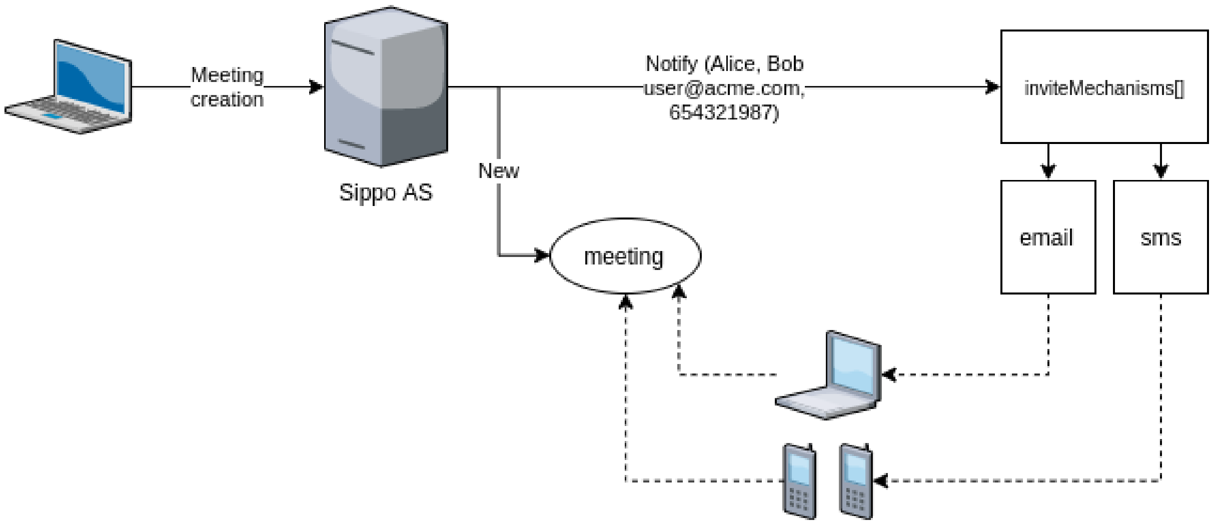
5.3 Meetings service

Section contents

The meetings service allows users to schedule conferences to be run on the Sippo platform. The service manages the creation of the invites to all the participants and exposes the webpage that will serve as conference place for anonymous users.

The invitation to the different participants are based on the available notification methods.

Mainly, the system solves the creation of the conference and uses the available notification backends to send the corresponding invitations to the meeting participants.



Mainly the notification methods use templates for send the messages, these templates could be configured on all the notification backends. Please, expand information about them at:

- *SendSMS service*
- *Email service (mailer)*

Note: This service is used by *Sippo collaborator* and exposed to third-party applications through the REST API. Check *Reference documentation about REST API* to expand information about "how to use".

5.3.1 Configuring the service

The service, as usual is configured via the `wac.ini` file. It contains a top-level entry named `[meetings]` that holds the service's details.

Mandatory fields

linkHost The host that will be used to build invitation's links. If not given, the public IP of the SippoAS will be used.

Optional fields

inviteMechanisms[] The mechanisms the SippoAS will handle to send invites. The ones not defined here, will need to be handled by application. For example, if `sms` is missing here, no invites will be sent **from the SippoAS** via SMS.

Currently available values are:

- `email` –> Dependent service required: `[mailer]`
- `sms` –> Dependent service required: `[sendsms]`

linkFormat The format to use for the meeting URL. If `unique` the meeting ID is used. If `user` the creator user's alias is used.

templatePath Where to find the sms and email template folders, relative to `config/`. Defaults to `meetings/tpl`. Please read docs about its structure.

shortenerToken Needed token to do shortened links using a Bitly account associated to the token. If this field isn't available, this service won't be used.

languagesTemplates To specify which are the languages of the templates that will be used to create the message sent by sms. Currently available values are: `es`, `en` and `ca`. By default, all will be used.

remindBefore If specified, SippoAS will send a reminder to every participant in the meeting `N` minutes before (`N` is the value of this parameter).

pstnExtension If specified, the meeting invitations will contain information in order to join the meeting through pstn.

5.3.2 Email invites

When creating a meeting and specifying a valid email in the request, email invites are sent to participants.

- Those invites are sent using the optional `mailer` service. To define how the invites look like, email templates are used. You can refer to the details reading the *Email service (mailer)* documentation.

5.3.3 SMS invites

When creating a meeting and specifying a valid phone (MSISDN format) in the request, SMS invites are sent to participants.

- Those invites are sent using the optional `sendsms` service. To define how the invites look like, sms templates are used. You can refer to the details reading the *SendSMS service* documentation.

5.3.4 Available parameters for meeting templates

To customize the notification templates, you can use the following available variables:

meeting.name	Name of the meeting.
meeting.user.id	Id of the user creating the meeting.
meeting.user.domain	Domain of the user creating the meeting.
meeting.user.username	Username of the user creating the meeting.
meeting.user.email	Email address of the user creating the meeting.
meeting.user.created	Timestamp of when the user creating the meeting was created.
meeting.user.lastLogin	Timestamp of when the user creating the meeting last logged in.
meeting.user.role	Role of the user creating the meeting.
meeting.user.capabilities[]	Capabilities of the user creating the meeting.
meeting.user.mobilePhone	Mobile phone number of the user creating the meeting.
meeting.user.landLineNumber	Number of the user creating the meeting.
meeting.user.alias	Alias of the user creating the meeting.
meeting.users[]	Ids of the users participating in the meeting.
meeting.language	Default language for the meeting.
meeting.participants[]	List of the participants in the meeting.
meeting.validSince	Timestamp of the date since which the meeting is available.
meeting.validUntil	Timestamp of the date until which the meeting URL is available.
meeting.url	URL of the meeting.
meeting.ddi.extension	Meeting DDI extension.

5.4 Email service (mailer)

Section contents

The SippoAS provides a service for sending emails. That service may be used by other services that need to reach users via email.

The emails to be sent can be created by the service's user or may use already defined templates. This allows the developer or the administrator to not need to define similar or equal emails more than once. Localized versions for such templates are also an option.

All the features provided by the email service are supported by the underlying library nodemailer. Check it out if you need something that's not explicitly supported by this service.

Note: This service is used by `Meetings`, a sippo-server specific feature service. Check more about this feature and how to configure it at *Meetings service*

5.4.1 Configuring the service

The service is configured via the `wac.ini` file as usual. It contains a top-level entry named `[mailer]` that holds the service's details.

There is one mandatory option to be provided: `host`, which identifies the SMTP host.

Here we present the full list of available options:

host The host (name or IP address) of the SMTP server.

username The username for authenticating against the SMTP server.

password The password to use for the given username.

port The SMTP port to use (defaults to 587)

secure Whether to use TLS or not when talking with the SMTP. `false` still allow STARTTLS (defaults to false).

pooled Whether to use a connection pool or not (defaults to false)

from A forced From: address, ignoring each email's one. (defaults to the one given in the email object)

Note: Currently, only SMTP/S is supported

5.4.2 Creating new templates

Templates use the handlebars syntax. It's an easy and intuitive replacement syntax. Just wrap your items to be replaced with `{{ }}` and you're ready to go. Please take a look at provided template and website to learn more about it.

To have a template ready to use, please drop it in a new folder inside `templatePath` (variable under `[meetings]` entry) . For instance, if you want to add a new template for sending emails for meetings, you can create a new folder called `email` inside `templatePath` and copy there your template files.

5.4.3 Template naming rules

Template must adhere to strict but simple naming rules. As the email service allows to send both plaintext and HTML versions of an email, there has to be a template for each of those versions. The plaintext one is mandatory and the HTML is optional.

For the plaintext version, the file must be named `text.hbs`. The HTML template, is named `html.hbs`. If both templates are found, the email will have one alternate part for each of the versions.

For example, this might be the real contents of `templatePath`:

```
templatePath/
`- email/
   `- html.hbs
   `- text.hbs
```

5.4.4 Localization of templates

There is also support for localization of templates. From the administrator point of view it is as easy as creating a subdirectory in the template directory named after the locale to be used. Inside this folder we can also include two more files `from.hbs` and `subject.hbs` which are copied to the corresponding fields of the email. That is, for our example `email` template we can provide a Spanish translation by creating a folder `es` inside it with the corresponding localized templates.

```
templatePath/
`- email/
   `- html.hbs
   `- text.hbs
   `- es/
      `- html.hbs
      `- text.hbs
      `- from.hbs
      `- subject.hbs
```

If no translation is requested, the `email` contents will be used.

Note: It's up to the service using the email service to ask for the localized versions of the email.

5.5 SendSMS service

Section contents

The `sendsms` service provides the ability to send SMS to mobile devices. This service may be used by other services that need to reach users via SMS.

The SMS will be created by already defined templates. These templates will include key words to be replaced by variable values.

In addition there is the possibility to create different SMS texts based on the language selected.

All the features provided by the sms service are supported by the underlying external service Beepsend. This external service has an API with which we can send SMS from the SippoAS.

To properly get this service working, it is necessary to create a profile on the provider platform. Check specific configurations on the corresponding backend configuration section.

Note: This service is used by `Meetings`, a sippo-server specific feature service. Check more about this feature and how to configure it at *Meetings service*

5.5.1 Configuring the service

The service is configured via the `wac.ini` file as usual. It contains a top-level entry named `[sendsms]` that holds the service's details.

```
[sendsms]
backend[] = sendsms
; backend[] = bezeqsms
; backend[] = linkmobility
```

Optional paramters

- `templatePath`: This key will define the path where to look for templates (defaults to `<rootdirectory>/config/smstemplates`).

It can be used as an absolute path or as a relative path. E.g.

```
templatePath = ./otherfolder
```

In order to find the templates in `<rootproject>/config/otherfolder`.

5.5.2 Available backends

- **sendsms** send SMS using Beepsend API. The backend will connect using a connection token with the API and will send one or several messages. When a message has more than 70 characters, Beepsend will create two related messages.

```
[backend.sendsms]
module = sippo-sms-bs
urlService = https://api.beepsend.com
token = xxxxxxxxxxxxxxxxxxxxxxxxxx
from = Quobis
```

Backend parameters:

There are three mandatory options to be provided: `urlSevice`, `token` and `from`.

urlService: The BeepSend API URL.

token: Connection Token for authentication.

from: The sender id. It must be a string with one of the following formats:

- Alphanumeric. Maximum allowed characters are 11.
- MSISDN numbers or short numbers / codes. Between 9 and 17 chars.
- National format. Maximum length is 7 chars.

encoding: Allows you to specify the message encoding . Available options are UTF-8, ISO-8859-15 or Unicode. Defaults to UTF-8.

- **sippo-linkmobility** send SMS using the LinkMobility provider. To use it, enable it in the `wac.ini` file and set up the required parameters as follows:

```
[backend.linkmobility]
module = sippo-sms-linkmobility
login = username
password = <your_md5password>
from = QUOBIS
```

Backend parameters:

login Authentication parameter, as provided by LinkMobility

password Authentication parameter, as provided by LinkMobility. It MUST be the already MD5-hashed value.

from The string shown as the SMS originator, defaults to QUOBIS.

5.5.3 Creating new templates

Templates use the handlebars syntax. It's an easy and intuitive replacement syntax. Just wrap your items to be replaced with `{{ }}` and you're ready to go. Please take a look at provided template and website to learn more about it.

To have a template ready to use, please drop it in a new folder inside `templatePath`. For instance, if you want to add a new template for sending SMS for meetings, you can create a new folder called `myMeetingsSMS` inside `templatePath` and copy there your template files.

5.5.4 Template naming rules

Template must adhere to a strict but simple naming rules. As the sms service sends only plaintext, the plaintext template is mandatory. The file must be named `text.hbs`.

For example, this might be the real contents of `templatePath`:

```
templatePath/
`- myMeetingsSMS/
   `- text.hbs
```

5.5.5 Localization of templates

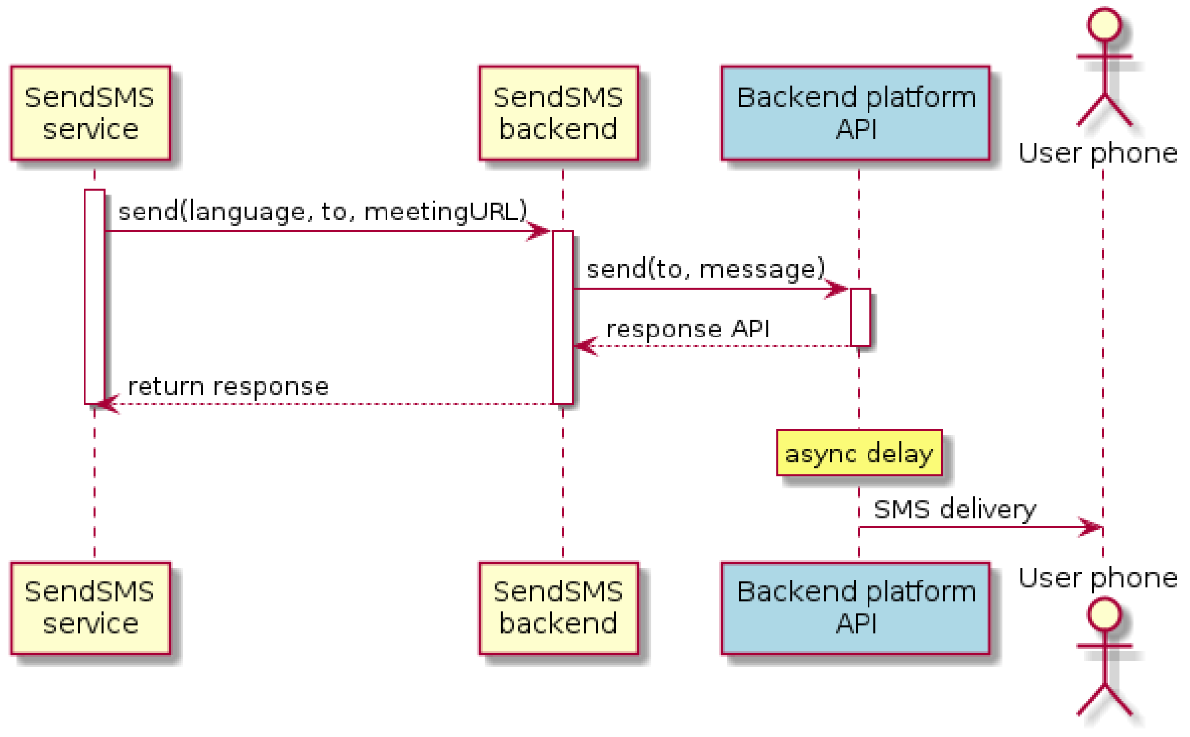
There is also support for localization of templates. From the administrator point of view is as easy as creating a subdirectory of the template directory named after the locale to be used.

That is, for our example `myMeetingsSMS` template we can provide a Spanish translation by creating a folder `es` inside it with the corresponding localized templates:

```
templatePath/
`- myMeetingsSMS/
   `- text.hbs
   `- es/
      `- text.hbs
```

If no translation is requested, the `myMeetingsSMS` contents will be used.

5.5.6 Activity diagram



5.6 Certificate administration

Chapter contents

5.6.1 Introduction

In a production environment all the connection must be established over TLS to encrypt all the communications and to validate the identity of the Sippo AS and the Sippo MS servers when it is accessed by the WebRTC clients. Additionally, several WebRTC features (e.g. screen sharing) supported by the browsers can not be executed in web pages downloaded over non-TLS connections. It is also important to load the rest of contents of the Web over TLS to avoid **cross-domain** issues (CORS) when mixing content loaded over TLS and non-TLS connections.

5.6.2 How to configure Sippo roles to enable secure connections

The Sippo server installed with the official Ansible script comes with the `openssl` software installed. So the generation of Certificate Signing Request (CSR) to be signed by a CA is a straightforward process.

5.6.2.1 Generate certificates

Those are the recommended steps to generate the CSR and the private Key which will be used in production. Please, make sure the key file is not accessible by unauthorized users:

```
admin@wac: cd /opt/sippo/certs
admin@wac:/opt/sippo/cert$ openssl req -new -newkey rsa:2048 -nodes -out webrtcappname_
↪yourdomain_com.csr -keyout webrtcappname_yourdomain_com.key -subj "/C=ES/ST=ProvinceNameo/
↪L=CityName/O=BBVA/OU=DepartmentName/CN=webrtcappname.yourdomain.com"
```

The output below will be shown in the terminal:

```
Generating a 2048 bit RSA private key
.......+++
.............................................................+++
writing new private key to 'webrtcappname_yourdomain_com.key '
-----
```

After finishing this process both the CSR file and the Key file will be stored in the folder. **The CSR file must be sent to the CA to generate a certificate by a trustable CA** which will be stored in the same folder.

Once the Certificate file is received from the CA, it should be stored on the sippo-server configuration folder for convenience, but a custom folder is also supported.

Note: Refer to Let's encrypt and CERTBOT to obtain community free certificates.

5.6.2.2 Configuration on Nginx

All Sippo network connections are governed by an NGINX proxy engine. The certificates must be configured there as follows.

Edit `/etc/nginx/sites-enabled/wac-nginx.conf`

First at public server certificate

```
server {
   listen 443 ssl;    # Change public port if required
   server_name localhost;

   ssl_certificate_key   /opt/sippo/certs/sippo-host-key.pem;
   ssl_certificate       /opt/sippo/certs/sippo-host-cert.pem;
```

Also at the `sapi/` endpoint redirection

```
location /sapi {
  proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header    Host $host;
  proxy_http_version  1.1;
  proxy_pass          https://localhost:8080;  # SAPI interface

  proxy_ssl_certificate     /opt/wac/config/cert/sippo-sapi-cert.pem;
  proxy_ssl_certificate_key /opt/wac/config/cert/sippo-sapi-key.pem;
  proxy_ssl_verify          off;
```

Generally, on that file, ensure the filenames and redirection endpoints.

After changes reload the NGINX and the process is done.

```
$ sudo systemctl reload nginx
```

5.6.2.3 Configuration on the Sippo AS

Note: From Sippo 3.1 in advance, public connections are managed by Nginx, so the configuration must be done there. This section is informative for custom cases where Sippo AS and Sippo MS are directly exposed to the network.

The `sippo-server` configuration (`wac.ini`) must be modified to instruct the Sippo AS to use this new certificate.

Locate the `sippo-server` interfaces where you want to enable secure connections and modify them to serve the keys. Also modify the public and public_secure fields to match your domain definitions.

```
[wiface0]
id = public
public = "http://app.yourdomain.com"
public_secure = "https://app.yourdomain.com"
iface = x.x.x.x
port = 8000
port_secure = 8001
key  = /opt/sippo/cert/sippo-public-key.pem
cert = /opt/sippo/cert/sippo-public-cert.pem

[wiface1]
id = internal
iface = 127.0.0.1
port_secure = 8080
key  = /opt/sippo/cert/sippo-sapi-key.pem
cert = /opt/sippo/cert/sippo-sapi-cert.pem
```

From this point on, all the configured `sippo-server` interfaces will provide a secure TLS connections that uses our designed certificate.

5.6.3 Configure Sippo user credentials

The Sippo users must connect also securely to the WebRTC gateway. To ensure this check these two points:

1. Check your vendor documentation to deploy valid certificates on the WebRTC Gateway
2. Ensure your IMS credentials points to a SSL endpoint.

   ```
   "authserver": "wss://gw.deploy.lab.com",
   ```

   This information is important to get it stored into your credentials template.

5.7 WetRTC Media Codecs

To communicate in real time, using audio and/or video, two users need to be able to agree upon mutually-understood codecs for each track. This section reviews the WebTRC codecs supported by Sippo WAC.

These match, according to RFC 7742 and RFC 7847, the required codecs in any fully WebRTC-compliant application and are compatible with most browsers.

5.7.1 Video

Below is a list of Sippo WAC supported video codecs.

Codec	Features
VP8	Preferred because of its quality and low hardware requirements
AVC/H.264	Should be used when there are strong requirements. (C++SDK)

Video transcoding is not supported, all the participants in a room must use the same codec, either VP8 or H264.

Video codec configuration is done through Ansible variable definition file, SFU-wrapper section. See *Variables definition*

5.7.2 Audio

Below is a list of Sippo WAC supported audio codecs.

Codec	Features
Opus	Default and preferred for of its quality in less bandwidth
G.711 (μ/law ,A/law)	Should be used when the C++SDK is involved

In scenarios where G.711 is involved, participants can keep using Opus and the Audiomixer will do the transcoding.

Note: Please note that Opus requires more CPU so, if there are CPU constraints in the server, please consider the use of G.711 which requires more bandwidth but consumes much less CPU.

CHAPTER

SIX

SERVICE REFERENCE DESCRIPTION

6.1 Main roles reference for Sippo AS

6.1.1 Sippo Server

Contents

6.1.1.1 Service name

- Formal name: Sippo Server
- System name: SippoAS.sippo-server

6.1.1.2 Main feature covered

Development that handles all the application logic and orchestration of different internal services.

6.1.1.3 Configuration

Configuration of sippo-server is done through `config/wac.ini`

File is organized in [section]s. There are four different kind of sections:

- **global**: for options affecting the whole WAC.
- **core**: WAC basic services configuration.
- **service sections**: free name sections that define services.
- **backend sections**: sections with names like "backend.name" that define the backend options for the service that points to this backend.

6.1.1.4 Sippo Server sub-services

- *Agent assigner*
- *Email service*
- *Meetings*
- *Reachability*
- *SendSMS service*
- *UsersGroup*
- *XMPP*

6.1.2 QSS

Contents

6.1.2.1 Service name

- Formal name: Signaling Server
- System name: SippoAS.qss

6.1.2.2 Main feature covered

Development that handles signaling for the SFU architecture. Signaling entails call related features such as call setup, call notification and call transfer.

6.1.2.3 Quobis Signaling Server sub-services

- *AudioMixersIO*
- *Invites*
- *IO-websockets*
- *Meeting signaling*
- *Quick Conference*
- *Rooms*
- *Trunk*

6.1.3 OAuth2 proxy

Contents

6.1.3.1 Service name

- Formal name: OAuth2 proxy
- System name: SippoAS.oauth2-proxy

6.1.3.2 Main feature covered

Service developed by Quobis to handle access delegation based on open standard OAuth2. It enables the authentication of SippoSDK applications and also plays the role of identifying the authentication provider based on its configuration. It also allows the Sippo solutions to handle non-standard OAuth2 access delegation systems by the implementation of ad-hoc custom modifications.

Check section *OAuth2-proxy - Federating identities* for expanded information about this mechanism on the Sippo solutions.

6.1.3.3 Configuration example

The OAuth2 service is divided on two elements, a client (to federate authentications) and a provider (to act as an identity provider). So two level3 services must be initialized on the `sippo-server` configuration file.

Sample `config/wac.ini` file with the parameters completed with the default values:

Client configuration

```
;
; Service/OAuth2 Client
;
; This service allows other services to connect to OAuth2-enabled providers.
;
; Required parameters
;  - wiface: the interface to add the callback to
;
; Optional parameters:
;  - secure: listen for secure connections at specified wiface. Default is false
;  - providers: path to a directory with JSON files for each available OAuth2 provider
;
[oauth2client]
wiface = public
secure = true
providers = config/oauth2/providers
```

Provider initialization

```
[oauth2provider]
```

6.1.4 XMPP Server

Contents

6.1.4.1 Service name

- Formal name: XMPP server
- System name: SippoAS.xmpp-server

6.1.4.2 Main feature covered

Messaging XMPP server to handle asynchronous message exchange between SippoSDK clients.

- Implemented using Prosody IM.

6.1.4.3 Configuration example

In addition to the XMPP server itself that must be started inside the cluster, some complimentary services are required to initialize in order to make this service fully functional.

Initialize xmpp-helper at `config/wac.ini`.

```
; XMPP service
; This service enables communication with the XMPP server
;
; Configure it in `config/xmpp.toml`
[xmpp]
```

Configure xmmp-helper at `config/xmpp.toml`. See *XMPP*.

```
# Mandatory parameters:
#  - xmppUrl: the URL to the WebSocket where the XMPP server is listening
#  - hostsAdminUrl: the URL to the websocket where XMPP server will receive virtualhost updates
#  - adminToken: the Prosody's admin login token
#
[xmpp]
xmppUrl = "wss://web.sippo/xmpp-websocket"
hostsAdminUrl = "http://prosody:5280/quobis_virtualhosts/virtualhost"
adminToken = "PLEASE CHANGE ME"
```

Extra deatils could be configured at xmpp server through the Prosody configuration.

6.1.5 Message broker

Contents

6.1.5.1 Service name

- Formal name: Message Broker
- System name: SippoAS.message-broker

6.1.5.2 Main feature covered

Messaging broker that handles communication of messages and events between independent internal Sippo services.

- Implemented using RabbitMQ.

6.1.5.3 Configuration example

Sample `config/default.toml` file with the parameters completed with the default values:

```
[rabbitmq]
address = "amqp://rabbitmq"

[rabbitmq.exchanges]
default = "sippo"
events = "sippo-events"
rpc = "sippo-rpc"
```

6.1.6 Database

Contents

6.1.6.1 Service name

- Formal name: Database
- System name: SippoAS.database

6.1.6.2 Main feature covered

A database is an organized collection of long-lasting data used by the different service to store the information which must be shared between service instances and be persistent across restarts.

- Implemented using MongoDB.

MongoDB is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (BSON), making the integration of data in certain types of applications easier and faster.

MongoDB is widely used at Sippo solutions for persistent running application storage. Some examples of the information contained into the DB are:

- Authentication entries for Sippo users.
- Credentials relationships to the Sippo users.
- Presence data.
- Call statistics.
- Call log stored as CDRs.

At the current version, the DB information is only accessible through a SippoAS interface (like SAPI), waccli (admin interface), or acceding directly to the MongoDB.

Staring on 4.0, **Sippo Manager** could bring information about the database status and its performance.

Contact with your technical support for any action that you require on the MongoDB.

Note: CDRs are currently stored on the database and could be retrieved using the Sippo REST API. Periodic backups of the DB are recommended.

6.1.6.3 Configuration

The database runs inside a dedicated container or isolated into a dedicated cluster. You can configure the exposed MongoDB endpoint using the configuration file `config/default.toml`.

Included here a sample `config/default.toml` file with the parameters completed with the default values:

```
# Quobis DB library configuraion
#
# Parameters:
# - {string} type - allow select different storage implementations:
#    * in memory (default)
#    * mongodb
#    * couchbase
[db]
type = "mongodb"

# MongoDB storage
#
# It supports up to MongoDB Server 3.4.0
#
# Required parameters:
# - {string} dsn - The DSN to use for database connection.  Authentication is
#         supported.  authSource query parameter might be needed!
#
[mongodb]
dsn = "mongodb://mongodb/wacDev?auto_reconnect=true"

# Couchbase storage
#
# It has only been tested with Couchbase EE 5.1.
#
# Required parameters:
```

(continues on next page)

(continued from previous page)

```
#   - {string} dsn - The DSN to use.  Specify bucket name as path
#
[couchbase]
dsn = "couchbase://quobis:asterisk@couchbase/default"
```

6.1.7 Reverse proxy

Contents

6.1.7.1 Service name

- Formal name: Reverse proxy
- System name: SIppoAS.reverse-proxy

6.1.7.2 Main feature covered

A type of proxy server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client, appearing as if they originated from the proxy server itself.

- Implemented using NGINX when deploying with Docker compose.
- Implemented using Traefik when deploying in Kubernetes cluster.

6.1.8 Static storage

Contents

6.1.8.1 Service name

- Formal name: Static storage
- System name: SippoAS.storage

6.1.8.2 Main feature covered

Used to store and deliver static files making them accessible to clients and services.

6.1.9 SFU dispatcher

Contents

6.1.9.1 Service name

- Formal name: SFU dispatcher
- System name: SippoAS.sfu-dispatcher

6.1.9.2 Main feature covered

Used to balance between different *SFU - Selective Forwarding Unit* units.

It plays a role of Cloud registrar for the different SFUs. This elements provides independence of the SFU instances, allowing the system to scale correctly and also providing redundancy in case of system failure.

It also limits the bandwidth of the video and provides a configuration option for better optimization on different scenarios without integrator intervention.

Note: WebRTC will consider the packet overhead in the BandWidth Estimation (BWE) in next releases. Currently it does not include neither IP, UDP or DTLS-SRTP headers. With the new changes (not before Chrome82) this overhead will be taken into consideration to calculate the used BandWidth.

6.1.9.3 Configuration

Configuration parameter on dispatcher configuration file `quobis-dispatcher-config.js` that allows to change the bandwidth of video calls.

```
//Maximum bandwidth that a video will have during a call
bitrate: 300,
```

Note: This is the maximum bandwidth that the video will have during a call. If it detects packet loss then the bandwidth is reduced by 1/3 until 1/3 of the initial amount is reached.

6.2 Main roles reference for Sippo MS

6.2.1 SFU - Selective Forwarding Unit

Contents

6.2.1.1 Service name

- Formal name: SFU
- System name: SippoMS.sfu

6.2.1.2 Main feature covered

Network element to handle WebRTC media streams. It receives streams from each conference participant and **forwards** them to the rest of participants.

It does not do media processing, it does not change neither the codec nor the bit rate and resolution of the video flows. But in adition to the forward features, it is able to play the WebRTC gateway role providing **media interworking**, WebRTC to SIP media profiles, DTLS-SRTP to RTP and timestamps modifications. When used with codecs which support Scalable Video Codec (SVC) it enables the delivery of flows different bit rates and resolutions to different endpoints based on defined rules.

Other extra characteristics:

- Flexibility to subscribe/publish specific streams.
- Easiness to compose the final application as the streams are received separately.
- Designed for multiparty conferencing.
- Compared with the MCU schema, it provides much lower CPU consumption but higher bandwidth consumption specially from the SFU to the clients.

SFU schema is being used widely on the WebRTC industry due its backend performance and increasing resources on the user devices (network and hardware).

- Implemented using Meetecho Janus.

6.2.2 SFU wrapper

Contents

6.2.2.1 Service name

- Formal name: SFU wrapper
- System name: SippoMS.sfu-wrapper

6.2.2.2 Main feature covered

Linked directly to the *SFU - Selective Forwarding Unit*, the sfu-wrapper grants connection between the sfu unit and the *SFU dispatcher* to handle redundancy and scaling. It also receives direct connections from the applications in order to join to rooms and negotiate the characteristics of the media flows published and received.

6.2.3 Audiomixer

Contents

6.2.3.1 Service name

- Formal name: Audiomixer
- System name: SippoMS.am

6.2.3.2 Main feature covered

SIP back to back user agent used for SIP trunking interconnection. It is also mixes the audio of the conferences and calls the SIP trunking. It enables the adaptation to the particular needs of the SIP trunk.

- Implemented using Asterisk.

6.2.4 Turn Server

Contents

6.2.4.1 Service name

- Formal name: Turn server
- System name: SippoMS.turn-server

6.2.4.2 Main feature covered

Backend support for STUN/TURN protocol and other network protocols required to handle WebRTC media on non-defined networks (DTLS, ICE, NAT discovery, . . .). This element implements connectivity protocols used to achieve real-time connectivity with endpoints which are behind NAT and restrictive firewalls.

- Implemented using coturn.

6.2.5 Recording worker

Contents

6.2.5.1 Service name

- Formal name: Recording worker
- System name: SippoMS.rec-worker

6.2.5.2 Main feature covered

Used to generate media recordings from the media captured in the *SFU - Selective Forwarding Unit* and *Audiomixer*.

6.3 Sippo server internal services reference

6.3.1 Agent assigner

Contents

6.3.1.1 Service name

- formal name: Agent assigner

- service name: AgentAssigner
- Service hierarchy: sippoas/sippo-server/agentassigner

6.3.1.2 Main feature covered

This service attends requests from any source and provides in response a valid agent following some assignation method.

Two different assignation mechanisms are currently supported: system agent assignation and external agent assignation. One, and only one, of these mechanisms, could be used at a given time. The resolution method is specified by the system operator at configuration time.

System agent assignation will assign the first available agent, that is, the first agent connected to the system whose presence is 'unknown' or 'available'.

External agent assignation will delegate the agent assignation to a third-party REST API. A POST request will be made against a previously configured endpoint. The request will have the following body: `{channelId: string, channelType: string, context: any }`. The parameters in the body have the following meaning: `channelId`, will be an unique identifier for the resource being assigned (roomId, conferenceId, etc); `channelType` will be one of the following `chat`, `voice`, `c2c`; `context` will be any arbitrary unknown object intended for usage by the third party assigner that manages the REST API. The response must have the following body: `{agentId: string}`. `agentId` will be the identifier of the assigned agent.

6.3.1.3 Related services and dependencies

Sippo AS

- Message broker
- sippo-server

Sippo framework

- @quobis/ipc (RPCServer)
- @quobis/log
- Users model (core/Users.js)
- PresenceService

Note: It is required to inject the PresenceService since it isn't loaded by default on the Sippo core services.

6.3.1.4 Service interaction description

The service starts a RPC server over RabbitMQ and waits to receive RPC messages on queue `AgentAssignerService`.

When message with `getAgent` method is received, when configured with system agent assignation it will gather all users with `+c2cagent` capability and will return the first with presence `online: true` and `activity: available` or `activity: unknown`. If configured with external agent assignation the service will make an HTTP request to the configured endpoint and return the `agentId` contained in the response.

6.3.1.5 REST API endpoints exposed

None

6.3.1.6 SippoSDK direct implications

None

6.3.1.7 Capability implications

Required to define assignable users with the user capability `c2cagent`

6.3.1.8 Service Configuration

To enable the service the service initialization needs to be added to `wac.ini` file.

```
[agentassigner]
```

To indicate an user must behave as an assignable user, `+c2cagent` capability must be added to the user.

If the block above is not present in the `wac.ini` file, neither external nor internal agent assigners will be enabled. Given the above block is present the operator could choose to use the external or internal agent assigner using the `config/agentassigner.toml` file.

A `config/agentassigner.toml` file with the following content will enable the external agent assigner:

```
[agentassigner]
externalAssignerUrl = "url of the external assigner endpoint"
```

If the `config/agentassigner.toml` does not exist, is empty or contains an empty `agentassigner` section the internal agent assigner will be used.

6.3.1.9 Configuration examples

```
[agentassigner]
```

If `user1@domain` and `user2@domain` includes the capability `+c2cagent`, they will be used to feed the service. When requesting an assignable user, the current service will offer `user1` or `user2` if they are available. Or any one of them if both are not available.

6.3.1.10 Logs: known log messages

Level	Message
Silly	`Agents obtained: ${users}`
Error	`An error ocurred while resolving agent`
Debug	`Message received: ${message}`
Debug	`Trying to obtain agent`
Debug	`Trying to obtain agents`
Debug	`Available agent found: ${agents[0]}`
Debug	`Checking if ${agent.id} is available`
Debug	`Getting presence for user ${userId}`
Info	`Resolved agent ${agent}`
Info	`Agent assigner configured with external assigner at: ${externalAssignerUrl}`
Info	`Agent assigner will use system assigner`

6.3.1.11 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify RabbitMQ server is up and running.
- Verify sippo-server is connected to RabbitMQ.
- Check sippo-server logs looking for the error shown in the previous section.
- Increase log level to `silly` to be able to see the users obtained by the service.
- Check users have `+c2cagent` capability.
- Verify users presence status.

Possible caveats

- It is not possible to ensure that the response is deterministic. The assignable user offered as return depends on a bulk request, results are not ordered, so *"the first user on the array"* do not always correspond with the same user.
- On high load scenarios, it is possible that we can face race conditions since there is a delay between the presence check and the presence update. This service does not update the presence of the assigned users.
- `c2cagent` is not a configurable parameter.

6.3.2 Meetings

Contents

6.3.2.1 Service name

- Formal name: Meetings
- System name: meetings
- Service hierarchy: sippoas/sippo-server/meetings

6.3.2.2 Main feature covered

The meetings service allows users to schedule conferences where participants can be invited. The service will take care of sending those meeting invitations via email or SMS.

For every email meeting invitation, an ICS (calendar) file is sent as an attachment. This calendar file includes the meeting's name, its URL, organizer and start and end dates.

A meeting is accessed via a URL which is only valid for that specific meeting. This is URL is called the "meeting URL" and it can be configured in the service configuration.

6.3.2.3 Related services and dependencies

Sippo AS services

- Database service `database`
- Message queue service `message-broker`
- Sippo QSS - `qss.meeting`
- Specific `sippo-server` services required are:
  - `Resolver` for looking up users and their contact details
- Additionally, the service can take advantage of the following optional dependencies:
  - `Email` for sending meetings' invitations via email
  - `SendSMS` for sending meetings' invitations via SMS
  - `PushNotifications` for notifying of upcoming meetings via push messages

Sippo MS services

- SFU - Selective Forwarding Unit `sfu`
- SFU wrapper `sfu-wrapper`
- AudioMixer `am`
- AudioMixer wrapper `am-wrapper`

6.3.2.4 Service interaction description

Resolver

Using the internal services' API, the Meetings service uses the `Resolver` to fetch the required contact details from users being invited to a meeting. The contact details are then used to properly build the SMS or email meeting invitations.

SendSMS

Meetings makes use of the `SendSMS` service for sending out the meeting invitations to participants' phone numbers using SMS.

Email

Meetings makes use of the `Email` service for sending out the meeting invitations to participants' using emails.

PushNotifications

Meetings makes use of the `PushNotifications` service for sending out a reminder of the meeting using mobile providers' push notifications service.

The push payload will be:

```
{
  "type": "meeting-reminder",
  "info": {
     "id": meeting.id,
     "name": meeting.name
  }
}
```

Please note that above payload is not admin-configurable.

6.3.2.5 REST API endpoints exposed

All of the following endpoints MUST be authenticated with the proper `Authorization` header.

- GET /sapi/meetings
- POST /sapi/meetings
- GET /sapi/meetings/:id
- PUT /sapi/meetings/:id
- DELETE /sapi/meetings/:id

6.3.2.6 SippoSDK direct implications

Meetings service is only supported in SippoSDK-JS. SippoSDK-iOS and SippoSDK-Android do not support it yet.

6.3.2.7 Capability implications

Users that want to create and manage meetings must have the `meetings` capability assigned. If no `meetings` capability is assigned for a user, that user will be unable to schedule new meetings (but he still will be able to join someone else's meetings).

6.3.2.8 Service configuration

The service must be enabled and configured in `config/wac.ini` by making sure the section `[meetings]` is not commented out

The only required parameter is:

- `linkHost`: the host that will be used to build invitation's links. Do not include a trailing slash.

All the parameters described below are optional:

- `inviteMechanism` an array of enabled methods to send meeting invitations, currently only `email` and `sms`. The methods not defined here, will need to be handled by application. For example, if SMS is missing here, no meeting invitations will be sent from the SippoAS via SMS.
- `templatePath` where to find the templates that will be used for the meeting invitations' contents. Defaults to `meetings/tpl`, relative to the config directory (`/config`).
- `shortenerToken` if token for a bit.ly account is provided here, then the meeting URL will be shorten using the external bit.ly service. Please note that the token need to go between quotes. If no token is provided, no shortening will be done.
- `languagesTemplates` a list of languages, whose defined templates will be checked for length (SMS) and variables used in them.

- `remindBefore` amount of minutes before the start of a meeting when a reminder has to be sent via the defined mechanism. Sippo users will also receive the reminder via email if this is provisioned. Additionally, a push notification is also sent, unconditionally. The reminder is a copy of the meeting invitation sent when the meeting was created.
- `pstnExtension` if defined, meeting invitations will be able to include this as the dial-in number to join the meeting via regular audio only phone call. This will populate the meeting's field `ddi` with two values: `extension` (equal to `pstnExtension`) and `pin` (an 8-digit number). They will hold the required information to join the meeting via call to `extension`, which will ask for `pin` on answering.

Please note that previous versions supported a configuration parameter named `linkFormat`, it is not supported anymore.

6.3.2.9 Configuration example

Sample `wac.ini` section with the parameters completed with the default values:

```
[meetings]
linkHost = https://meetings.wac/
inviteMechanisms[] = email
inviteMechanisms[] = sms
; templatePath = meetings/tpl
; shortenerToken = '<ACCESS TOKEN>'
; languagesTemplates[] = es
; languagesTemplates[] = en
; remindBefore = 10
; pstnExtension = +34999999999
```

6.3.2.10 Templates definition

One important aspect of the meetings service is the ability to configure the contents of the meeting invitations sent to participants.

The service allows to configure them depending on the channel used to send them (SMS or email, currently) and the meeting's defined language.

Templates are defined in the given `templatePath` from the configuration (relative to `./config`) and are structured in the following manner (default values used):

```
config/
 +- meetings/
     +- tpl/
         +- email/
         |   +- en/
         |   |   +- {from,html,subject,text}.hbs
         |   +- es/
         |       +- {from,html,subject,text}.hbs
         +- sms/
             +- en/
             |   +- {from,text}.hbs
             +- es/
                 +- {from,text}.hbs
```

As you can see, the file hierarchy is defined by the mechanism used to send the meeting invitation (sms or email) and, underneath, the language of the meeting. This way we have full control on how to build the meeting invitation.

Additionally, for each of the mechanism, each of its parts is described in one single file for increased flexibility. For instance, in the case of the SMS, the sender of the SMS can be defined in the `from.hbs` file while the contents of the SMS itself, are defined in the `text.hbs` file. In the case of the email we have more fields available, so there are more files to be customized. Particularly for email we have both text.hbs and html.hbs. This way the email will be properly rendered in all email readers, no matter if they support HTML or not.

All the templates are defined using hbs files, which are Handlebars files. Handlebars is a third party library that helps users to define templates by separating the JavaScript code from the HTML or text template itself. See the linked website above for further information.

Apart from the standard Handlebars tags and syntax, we provide the formatDate and formatTime helpers to localize the date strings to the meeting invitation's language. For example:

```
Your meeting {{ meeting.name }} will start on {{ formatDate
meeting.validSince }} at {{ formatTime meeting.validSince
hour="numeric" minute="numeric" hour12=false timeZone=''Europe/Madrid'' timeZoneName="short" }}
```

You can see the full list of parameters that can be fed in the Intl documentation, under the options description.

The data passed to the templates is the following:

party The email address or phone number of the recipient

meeting The full meeting object:

- **meeting.validSince** The date the meeting is scheduled to start
- **meeting.validUntil** The date the meeting is expected to end
- **meeting.url** The URL to access the meeting
- **meeting.ddi**
  - **meeting.ddi.extension** The dial-in number for this meeting
  - **meeting.ddi.pin** The required PIN to inform when dialing in the meeting DDI
- **meeting.language** The configured language for the meeting

meeting.user The full user object of who created the meeting:

- **meeting.user.username** The username of the user who created the meeting
- **meeting.user.domain** The domain of the user who created the meeting
- **meeting.user.email** The email of the user who created the meeting

Please have a look at the provided templates in config/meetings/tpl to explore all the possibilities. A sample full template may be:

```
<p>Hi {{ party }},<br/>you have been invited to a meeting via our
SIPPO platform.</p>
<p>To join us, please
<a href="{{ meeting.url}}">this link</a>
{{#if meeting.ddi.extension}} or dial {{meeting.ddi.extension}} and introduce the code{
↪{meeting.ddi.pin}}{{/if}}.</p>
<p>If you cannot make it, please answer "No" to the RSVP.</p>
<p>Thanks!</p>
```

6.3.2.11 Calendar file

When sending a meeting invitation by email, a calendar file is attached. This file is formatted according to RFC-5545.

The main fields to note that are included are: meeting's starting and ending dates and times, meeting's name, URL and organizer.

The organizer is an RFC-defined value of the format Name <email@address.tld>. This service will use the meeting's creator username as Name and its email address. If no email address is defined, a dummy none@example.org will be used.

6.3.2.12 Logs: known log messages

Error messages

- `error: Invalid linkHost`: the configuration parameter `linkHost` must be set to a valid URL. Do not include a trailing slash.
- `error: Unable to send reminder for meeting: meetingId. Cause: msg`: an error occurred when trying to send a meeting reminder using Push Notifications. Please review Push Notifications service's logs.
- `error: Error while processing onSessionRegistered`: an error occurred when a user joined the meeting's URL. The full error must be printed following this line, which will give you details about the precise failure.
- `error: Error while processing onSessionDown`: an error occurred when a user disconnected from the meeting. The full error must be printed following this line, which will give you details about the precise failure.
- `error: Process onDeleteUser failed`: an error occurred when a deleted user was removed from its meetings. The full error must be printed following this line, which will give you details about the precise failure.

Warning messages

- `warn: unable to load resource for sms|email (msg)`: the service was unable to find the meeting invitation generator for the indicated mechanism. Probably the meeting invitation mechanism was misspelled or is not available.
- `warn: Cannot send the meeting invitation, not available`: a meeting invitation was not sent because the meeting created by the application was configured to do not send a meeting invitation from the sippo-server instance. This message is a confirmation that this server MUST NOT send a meeting invitation.
- `warn: Cannot send meeting invitation by email: Email service not available`: the meeting invitation cannot be sent by email because there's no such service enabled.
- `warn: Cannot send meeting invitation by sms: SMS service not available`: the meeting invitation cannot be sent by SMS because there's no such service enabled.
- `warn: Failed to invite participant INVITE to meeting NAME(id)`: the service was unable to send the given meeting invitation.
- `warn: The sms template TPL is too large. It will be sent several SMS by meeting`: the configured template, named `TPL` is too large to fit one single SMS. It will span across several SMS. This message only appears when the `languagesTemplates` variable is set.
- `warn: Field <FIELD> hasn't been used in TPL`: if an optional field `<FIELD>` available for the configured template `TPL` has not been used, this warning is issued. This message only appears when the `languagesTemplates` variable is set.

6.3.3 Reachability

Contents

6.3.3.1 Service name

- Formal name: Reachability
- System name: sippoas.sippo-server.reachability
- Service hierarchy: sippoas/sippo-server/reachability

6.3.3.2 Main feature covered

This service provides a way for obtaining the reachability of a system resource. Supported resources are users and groups.

Reachability is defined in terms of transports and resources. A resource will be reachable if communications are possible using at least one of it's transports. Supported transports are push notifications an direct connection using the *Wapi*.

This service given a resource, which could be a user id or a group id, will provide all the resource's transports. For a user it provides information about it's push tokens and *Wapi* connection status and for a group it will do the same for all of it's participants.

6.3.3.3 Related services and dependencies

The list below represents the services that have some kind of interaction with the `reachability` service.

- Sippo AS
  - `sippoAS.sippo-server.UsersGroup`
  - `sippoAS.sippo-server.PushNotifications`
  - `sippoAS.sippo-server.User`
  - `sippoAS.sippo-server.Session`

6.3.3.4 Service interaction description

The service starts a RPC server over RabbitMQ and waits to receive RPC messages on the `reachability` queue. When a message with `getResourceReachability` method is received it will try to obtain the reachability for the resource.

Reachability information gathering uses the following algorithm:

1. Check if the resource if an user and if so get connection status and push tokens for that user
2. Check if the resource is a group and if so get reachability information from all of it's participants

If the provided resource is not an user nor a group just return no reachability information.

If `sippoAS.sippo-server.PushNotifications` is not enabled this service will not try to resolve pushes.

If `sippoAS.sippo-server.UsersGroup` is not enabled this service will not try to resolve groups.

6.3.3.5 REST API endpoints exposed

None

6.3.3.6 SippoSDK direct implications

None

6.3.3.7 Capability implications

None

6.3.3.8 Service configuration

Enabling the service is a matter of adding the line `[reachability]` to `config/wac.ini` file.

```
[reachability]
```

6.3.3.9 Configuration examples

```
[reachability]
```

6.3.3.10 Logs: known log messages

6.3.3.11 Troubleshooting & caveats

If the service is not working as expected, first make sure the RabbitMQ server is up and running and sippo-server is connected, if it's not and error log indicating that should be displayed.

If the service does not resolve groups or push transports please check if the *UsersGroup* and *PushNotifications* services are enabled.

6.3.4 Service Hooks

Contents

6.3.4.1 Service name

- Formal name: Service Hooks
- System name: sippoas.sippo-server.servicehooks
- Service hierarchy: sippoas/sippo-server/servicehooks

6.3.4.2 Main feature covered

This service allows to implement hooks to core services and their events. The task the service accomplishes is the linking of the these events to the hooks (implemented as backends) listeners.

Note that hooks are non blocking and become no part of the originating call workflow, so they cannot alter the behavior of the original execution, they only define **additional** operations.

6.3.4.3 Hooks

KamasSessionHooks

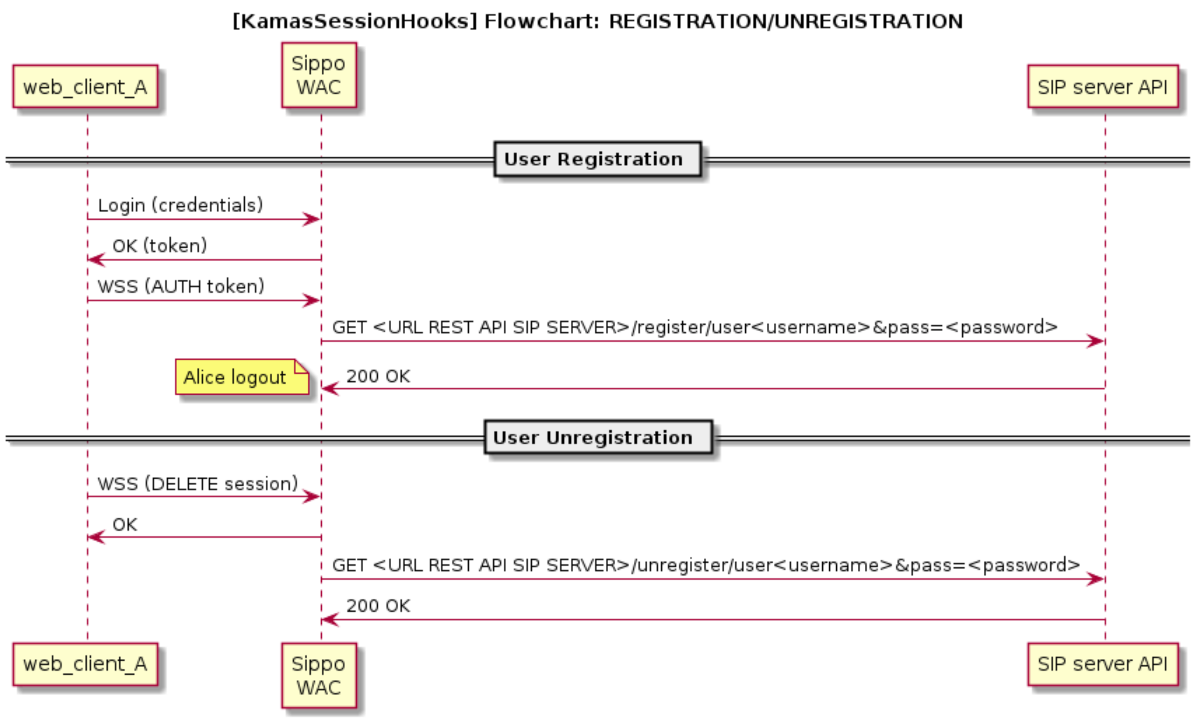
Listens to to client logins and logouts, more specifically to a user's first login or last logout, and based on it registers or un-registers them in the SIP server.

6.3.4.4 Service interaction description

KamasSessionHooks

When a user makes its first login (if there are many instances only the first one is considered) this *hook* makes a request, with the user credentials, to the SIP server API in order to register the new session.

In like manner, when this same user closes its last session a similar request is sent to un-register the client from the SIP server.


[KamasSessionHooks] Flowchart: REGISTRATION/UNREGISTRATION
web_client_A
Sippo WAC
SIP server API
User Registration
Login (credentials)
OK (token)
WSS (AUTH token)
GET <URL REST API SIP SERVER>/register/user<username>&pass=<password>
Alice logout
200 OK
User Unregistration
WSS (DELETE session)
OK
GET <URL REST API SIP SERVER>/unregister/user<username>&pass=<password>
200 OK
web_client_A
Sippo WAC
SIP server API

Note: KamasSessionHooks works by searching for the sip-mapping that associates a system resource to a SIP identity. See *SipMappings* for reference

6.3.4.5 REST API endpoints exposed

The SippoAS.sippo-server.servicehooks service adds the following endpoinst to the Sippo REST API (SAPI).

- `GET /sapi/hooks` Retrieve a list with all enabled hooks.

6.3.4.6 SippoSDK direct implications

None

6.3.4.7 Capability implications

None

6.3.4.8 Service configuration

The service is enabled through the configuration block `[servicehooks]` in the `config/wac.ini` file. A hook is modeled as a `Backend` and is configured with different `backend[]` entries.

Each of the hooks enabled is also self configured through a `[backend.hook-name-here]` configuration block. In each one of this configuration blocks a `module` entry must be included which defines its location relative to `/src/modules`.

KamasSessionHooks

Additionally to the mandatory `module` entry there is one optional parameter:

- `baseUrl`:the URL of the SIP server or proxy to send the register/un-register requests. Defaults to `http://kamas:8080/`.

6.3.4.9 Configuration examples

Service enabled and one hook configured:

```
[servicehooks]
backend[] = KamasSessionHooks
```

KamasSessionHooks configuration:

```
[backend.KamasSessionHooks]
module = service-hooks/KamasSessionHooks
baseUrl = http://kamas:8080/
```

6.3.4.10 Logs: known log messages

KamasSessionHooks

Level	Message
debug	`kamas:onUserUp: register wasnt thrown`
info	`kamas:onUsersRegistered:${response.code}`
debug	`kamas:onUsersRegistered:headers:${response.headers}`
debug	`kamas:onUsersRegistered:body:${response.data}`
debug	`kamas:onUserDown: register wasnt thrown`
info	`kamas:onUsersUnRegistered:${response.code}`
debug	`kamas:onUsersUnRegistered:headers:${response.headers}`
debug	`kamas:onUsersUnRegistered:body:${response.data}`

6.3.5 SipMappings

Contents

6.3.5.1 Service name

- Formal name: SipMappings
- System name: sippoas.sippo-server.sipmappings
- Service hierarchy: sippoas/sippo-server/sipmappings

6.3.5.2 Main feature covered

This service provides a way for defining, obtaining and updating SIP identities (`sip-mapping`) of a system resource. Supported resources are users and groups.

This service plays a key part on the SIP integration. It can identify the originator endpoints on the SIP network when they come from the Sippo system, and can identify resources on the system based on a SIP R-URI.

With this service, currently individual users and groups could be identified on the SIP network, but it opens the ability to establish multiple politics for SIP assignation.

SIP mapping

A `sip-mapping` presents the equivalence between a SIP identity (user@domain) and a system resource. By having this equivalence, we can route incoming SIP INVITE methods to a specific resource and replace the system identity when a system invite is being routed to the SIP network.

SIP mapping owner

It is defined as the owner of a `sip-mapping` the system resource assigned to it.

System resources

Currently the service supports `user` and `usersgroup` as assignable resources. Where a `user` is a single Sippo platform user and a `usersgroup` is a reachable element that involves multiple participants as a calling hunt group.

6.3.5.3 Related services and dependencies

The list below represents the services that have some kind of interaction with the *SipMappings* service.

Sippo AS

- Database
- Message Broker

Sippo framework

- `sippoAS.sippo-server.UsersGroup`
- `sippoAS.sippo-server.User`
- `sippoAS.qss.trunk`

6.3.5.4 Service interaction description

Administrative configuration

An admin can interact with the service through the REST API in order to create, update or delete a `sip-mapping` for a given resource.

When a new `sip-mapping` is created, the service validates if the resource exists before. An error will be thrown if the resource exists before.

Internal services interaction

Other internal services can reach *SipMappings* using the *Message Broker*. For instance, the Trunk service (`sippo-server.trunk`), when requires to identify a system resource, sends a message to *SipMappings* service in order to obtain the given resource.

Note: If `sippoAS.sippo-server.UsersGroup` is not enabled this service will reject any request that try to obtain the `sip-mapping` of a `usersgroup`.

User service also can take advantage of this service. Under request, this server can map **anonymous users** with SIP identities that are dynamically generated under request. A mapping for an anonymous user will follow this format:

```
dynamicMappingPrefix + userId + @ + dynamicMappingDomain
```

Where `dynamicMappingPrefix` and `dynamicMappingDomain` are both configurable parameters

6.3.5.5 REST API endpoints exposed

- create a sip-mapping identifying owner and SIP identity to use. `POST /sipMappings`
- Update a sip-mapping. Changing both ownership or SIP identity. `PUT /sipMappings`
- Obtain the sip-mapping corresponding to the owner specified. `GET /sipMappings`
- Remove a mapping entry. `DELETE /sipMappings`

6.3.5.6 SippoSDK direct implications

None

6.3.5.7 Capability implications

None

6.3.5.8 Service configuration

In order to enable the service, you have to add the configuration block `[sipmappings]` to the `config/wac.ini` file.

This service accepts two parameters, both used to configure the format of the dynamical mappings generated to the anonymous users, as explained in the service description.

- dynamicMappingPrefix: Prefix used for anonymous SIP identities.
- dynamicMappingDomain: Domain used for anonymous SIP identities.

```
[sipmappings]
dynamicMappingPrefix = <PREFIX>
dynamicMappingDomain = <DOMAIN>
```

6.3.5.9 Configuration examples

```
[sipmappings]
dynamicMappingPrefix = anonymous-
dynamicMappingDomain = anonymous-domain
```

The previous example, configures the system to assign to an anonymous user with the id: `abc`, the following SIP identity:

```
anonymous-abc@anonymous-domain
```

6.3.5.10 Logs: known log messages

- info: *Trying to add mapping with data: ${mappingData}* A `sip-mapping` is trying to be created
- info: *Trying to get mapping for owner: ${owner}* A `sip-mapping` is trying to be obtained to identify its owner
- info: *Trying to update mapping with data ${data}* A `sip-mapping` is trying to be updated
- info: *Trying to get owner by its mapping username: ${username}* The owner of a `sip-mapping` is trying to be identified by its username

6.3.5.11 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify that the *Message Broker* server is up and running.
- Verify that the service is connected to the *Message Broker*.
- Verify that in *Database* server is up and running.

A `Route error` message is shown when a service is trying to communicate with the SipMappings service and it is not properly connected to the *Message Broker*.

6.3.6 UsersGroup

Contents

6.3.6.1 Service name

- Formal name: UsersGroup
- System name: sippoas.sippo-server.usersgroup
- Service hierarchy: sippoas/sippo-server/

6.3.6.2 Main feature covered

This service is in charge of managing the creation, update and deletion of a group of users. It is used as a calling hunt group where a call placed to the group will result on a simultaneous ringing for all participants that stops as soon as one participant accept the incoming call.

Key concepts

usersGroup System element to group participants inside a element reachable as a calling hunt group. Normally referred as *Group*, once created it allows to an Owner participant to add users as participants, remove them, or edit the group name using the exposed API.

Participant One Sippo user that is included on the *usersGroup*. It could be added or removed by the owner of the *usersGroup*.

Owner Specific participant of a *usersGroup* that creates it. It have special permissions over the group in order to add or remove *participants* and edit the *usersGroup* details like the name.

6.3.6.3 Related services and dependencies

Sippo AS

- Message broker
- Database

Sippo framework

- @quobis/ipc (EventProducer, PeersProducer, RpcServer)
- @quobis/log
- @quobis/errors
- `sippoAS.sippo-server.Users`

6.3.6.4 Service interaction description

The service can be reached via REST interface or via RabbitMQ.

Independently of the transport used to reach the service, the main features carry out by this service are:

- Create a group.
- Get all created groups.
- Get a group by its identifier.
- Remove a user as participant of all groups where he is participating.
- Be able to update a group.
- Delete a group, that results on all participants removed. Once removed no more actions are allowed referred to the group (dial, edit, remove).
- Add a user as participant to a group.
- Remove a user as participant from a group.
- Remove all participants of a group.
- Know if a user is participating in a group.
- As participant, leave a group.

6.3.6.5 REST API endpoints exposed

- Create a group `POST /usersGroup`
- Get a list of all groups `GET /usersGroup`
- Remove a user from every group `DELETE /usersGroup/participations`
- Get a group by groupId `GET /usersGroup/:groupId`
- Update a group by groupId `UPDATE /usersGroup/:groupId`
- Delete a group, that results on all participants removed. `DELETE /usersGroup/:groupId Delete a group`
- Add a participant to a group `POST /usersGroup/:groupId/participants`
- Remove user from a group `DELETE /usersGroup/:groupId/participants`
- Remove all participants of a group `DELETE /usersGroup/:groupId/participations`
- Check if user exists in the group `HEAD /usersGroup/:groupId/participants/:userId`
- Leave group `DELETE /usersGroup/:groupId/leave`

6.3.6.6 SippoSDK direct implications

SippoSDK-JS

- Dedicated class to handle *usersGroup* classes

SippoSDK-iOS

- Dedicated class to handle *usersGroup* classes

SippoSDK-Android

- Not supported

6.3.6.7 Capability implications

Some capabilities are used associated to this service in the **ucc-client-web**. The capabilities are:

- `user-select-group`: when this capability is added to a user, he is able to change the group which belongs to
- `user-group`: when a user has this capability, he can see the groups that he is participating
- `user-group-contacts`: when this capability is added to a user, he is able to see the static contacts associated to a group.

6.3.6.8 Service configuration

Enabling the service is a matter of adding the following data to the `wac.ini`

```
[usersgroup]
backend[] = UsersGroup

[backend.UsersGroup]
module = UsersGroup
```

6.3.6.9 Logs: known log messages

Error level log messages

- `error: Specified participant does not exist'`: an error occurred when trying to add a user who doesn't exist to a group.
- `error: User ${user.id} is not the owner of the group`: an error occurred when a user tries to manage a group in which he is not the owner.

6.3.6.10 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify that the *Message Broker* server is up and running.
- Verify that the service is connected to the *Message Broker*.
- Verify that in *Database* server is up and running.

6.3.7 XMPP

Contents

6.3.7.1 Service name

- Formal name: XMPP
- System name: sippoas.sippo-server.xmpp
- Service hierarchy: sippoas/sippo-server/xmpp

6.3.7.2 Main feature covered

A standard XMPP server is included in Sippo solutions under service name **xmpp-server** (`sippoas/xmpp-server`), based on the well-know open-source package "Prosody".

The `xmpp` service is an internal component of the `sippo-server` responsible of integrating the `xmpp-server` with `sippo-server` for:

- user authentication
- contact management
- active chat list management
- group chat features
- access control

This service xmpp (sippoas/sippo-server/xmpp) has been developed to fulfill the above requirements and to serve as a connector between sippo-server and xmpp-server.

6.3.7.3 Related services and dependencies

The list below represents the services that have some kind of interaction with the xmpp service.

- Sippo AS
  - sippoAS.xmpp-server
  - sippoAS.message-broker
  - sippoAS.sippo-server.UsersGroup
  - sippoAS.sippo-server.domains
- Sippo MS
  - sippoMS.sfu.janus-wrapper

6.3.7.4 Service interaction description

This sub-chapter describes the interaction with the services introduced above. The following block diagram describes the relationships between these services.

6.3.7.4.1 SippoAS.xmpp-server

Interaction with xmpp-server is done via the XMPP protocol itself. This means that xmpp is seen as an XMPP-client from the point of view of the XMPP server, and the exchanged messages are just standard XMPP messages. An example of an XMPP message to add a new participant to a group would be:

```
<message
  xmlns="jabber:client"
  to="61362ae0-8911-11e9-9179-73a1dca97083@conference.quobis"
  id="122bb493-3cf8-4414-9cd6-87e852e03d23">
  <x xmlns="http://jabber.org/protocol/muc#user">
    <invite to="qa14@quobis"/>
  </x>
</message>
```

6.3.7.4.2 SippoAS.message-broker

The message-broker service is used by the xmpp service for communicating with other SippoAS services, like SippoAS.sippo-server.usersGroup.

With clients instantiated from the sippo-server's xmpp service.

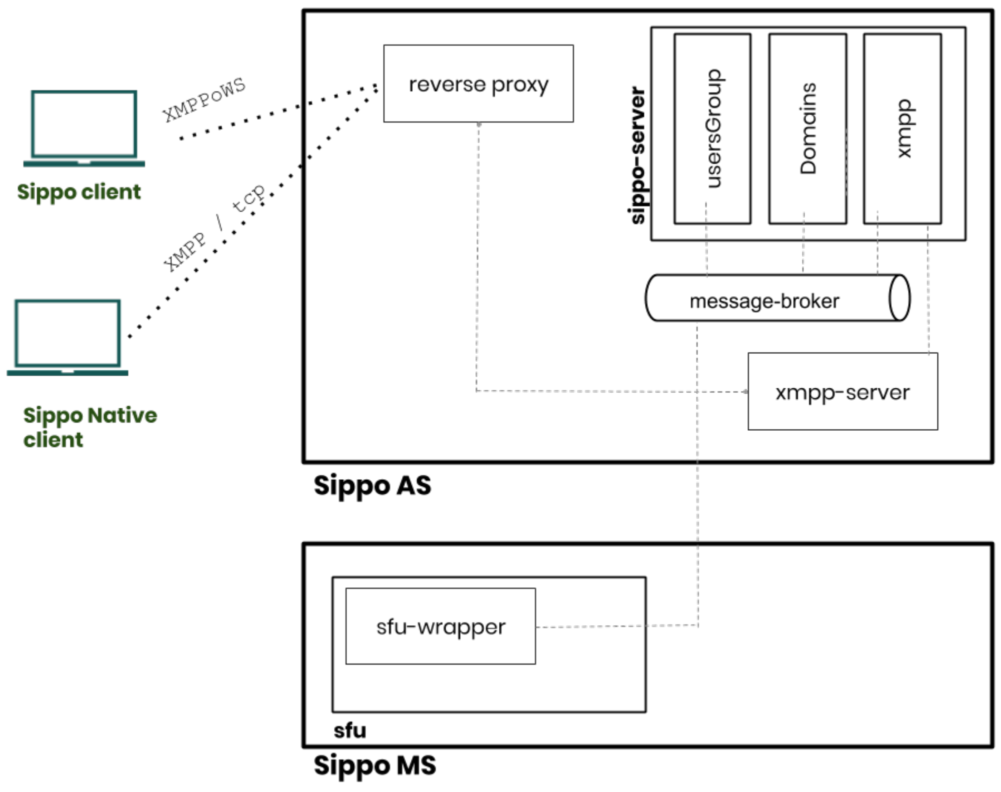


Fig. 1: An overview of the relationships between services

6.3.7.4.3 SippoAS.sippo-server.usersGroup

Groups chats in SippoAS are synchronized with the groups declared at the `SippoAS.sippo-server.usersGroup` service. The `SippoAS.sippo-server.xmpp` service listens to the following events in order to synchronize these groups and its participants. This communication is done via events sent through the `message-broker`.

- `groups.created`: a new group is created
- `groups.removed`: a group is destroyed
- `groups.user-joined`: a user joins a group
- `groups.leave-groups`: a user leaves all its groups

groups.created and **groups.removed**

These events will contain the affected group as payload.

groups.user-joined and **groups.leave-groups**

These events will contain both the affected user ID and the group as payload.

6.3.7.4.4 SippoAS.sippo-server.Domains

The XMPP server follows also the concept of domains for classifying users and provide multi-domain features. We use the `xmpp` service to keep them in sync with `sippo-server` domains.

The `sippoAS.sippo-server.xmpp` service listens to the following events in order to synchronize these groups and its participants. These communications are done via events sent through the `message-broker`.

- `domains.created`: a new domain is created

- `domains.deleted`: a domain is destroyed

domains.created and **domains.deleted**

These events will receive the full domain object as payload.

6.3.7.4.5 SippoMS.sfu.janus-wrapper

There are some use cases where audio and video conferences need to have a dedicated chat room that exists only while the conference is active. That behavior can be automated by listening to the events emitted by the `sippoMS.sfu.janus-wrapper` service, such as:

- `wrapper.userLogin`: a user enters a conference
- `wrapper.userLogout`: a user leaves a conference
- `wrapper.roomCreated`: a new conference is setup
- `wrapper.roomDestroyed`: a conference is destroyed

The `SippoAS.sippo-server.xmpp` service listens to these events via RabbitMQ and creates/destroys rooms in the `xmpp-server` accordingly.

6.3.7.5 REST API endpoints exposed

The `SippoAS.sippo-server.xmpp` service adds the following endpoinst to the Sippo REST API (SAPI). All of the following endpoints MUST be authenticated with the proper `Authorization` header.

- GET /sapi/xmpp/login
- PUT /sapi/xmpp/individualChatList/:jid
- GET /sapi/xmpp/individualChatList
- DELETE /sapi/xmpp/individualChatList/:jid

6.3.7.6 SippoSDK direct implications

SippoSDK versions since release 24.0.0 support XMPP chat service.

6.3.7.7 Capability implications

Users that need chat features must have assigned the `chat` capability. If no `chat` capability is present for a user, that user will have all chat features disabled.

6.3.7.8 Service configuration

Enabling the service is a matter of adding the line `[xmpp]` to `config/wac.ini` file.

Configuration itself is done at `config/xmpp.toml`.

All the parameters described below **must be** configured.

- `xmppUrl` the URL to the WebSocket where the XMPP server is listening
- `hostsAdminUrl` the URL to the websocket where XMPP server will receive virtualhost updates
- `adminToken` the `xmpp-server` admin login token, set it to the same, random, and long-enough value as in `prosody.cfg.lua`

6.3.7.9 Configuration example

Sample `config/xmpp.toml` file with the parameters completed with the default values:

```
[xmpp]
xmppUrl = "wss://web.sippo/xmpp-websocket"
hostsAdminUrl = "http://prosody:5280/quobis_virtualhosts/virtualhost"
adminToken = "PLEASE CHANGE ME"
```

6.3.7.10 Logs: known log messages

The `xmpp` service will log messages according the globally configured log level of the `sippo-server`.

Error level log messages

- `error: Failed to obtain [admin] XMPP client`: an error occurred when trying to instantiate an XMPP client for communicating with `xmpp-server`. Please review connectivity and configuration parameter `adminToken` (and its counterpart in `prosody.cfg.lua`).
- `error: Failed to create <domain> host in XMPP server`: an error occurred when `POST``ing to ``hostsAdminUrl` for creating a new domain.
- `error: Authenticate no-user error Bearer realm="Users", error="invalid_token"`: an error occurred while authenticating a user (will also appear for other kind of login, non-XMPP ones). In order to discover which user failed to log in, you should check `xmpp-server` logs.

Warning level log messages

- `warn: Failed to send invite to MUC ${muc} to user ${user}`: unable to send invitation for a group chat to a user joining a group or a conference. Please review `xmpp-server` logs for additional information.
- `warn: Failed to revoke group membership of ${member} from ${muc}: ${error}`: unable to revoke membership of user leaving a group or leaving a conference.. Please review `xmpp-server` logs for additional information.

Info level log messages

No log messages are defined for this level.

Debug level log messages

- `debug: Kicking ${username} from ${muc}`: when a user has been removed from a group chat.
- `debug: Received a new usersGroup creation`: when a new group event is received
- `debug: Received a usersGroup deletion`: when a deleted group event is received
- `debug: Received a change in a group participants (joining)`: when a new participant enters a group event is received
- `debug: Received a change in a group participants (leaving)`: when a participant leaves a group event is received
- `debug: Received a notification of a new domain`: when a new domain is created event is received
- `debug: Received a notification of a deleted domain`: when a domain has been deleted event is received
- `debug: Received a notification of a new participant in a conference`: when a new participant joins a conference event is received
- `debug: Received a notification of a participant leaving a conference`: when a participant leaves a conference event is received

- `debug: Received a new conference`: when a new conference has been created event is received
- `debug: Received a conference destroyed`: when a conference has finished event is received

Silly level log messages

- `silly: Starting endpoints`: when SAPI endpoints are being setup
- `silly: Connecting to message-broker`: when the connection to RabbitMQ is being established.
- `silly: ConferenceCreated payload ${json}`: the payload received for a new conference event.
- `silly: Destroying MUC ${muc}`: when the MUC of a conference is going to be destroyed.

6.3.7.11 Troubleshooting & caveats

- Whenever messages are sent to an offline (or non-existing) user, those will be stored in the `offline` storage. If such user eventually logs in, all those offline messages will be forwarded to her. Take this into account if offline users are receiving lots of messages, user may be flooded with messages when becoming online.
- When debug logs are enabled at `xmpp-server` service will a huge amount information of information on the actions taking place. Activate only to report problems to engineering. **Do not activate for production**
- Please note that Sippo only uses XMPP for instant messaging, not for presence updates.
- Please notice that currently standard XMPP clients are not supported as not all the features would be available for them.

6.3.8 Email service

Section contents

The SippoAS provides a service for sending emails. That service may be used by other services that need to reach users via email.

The emails to be sent can be created by the service's user or may use already defined templates. This allows the developer or the administrator to not need to define similar or equal emails more than once. Localized versions for such templates are also an option.

All the features provided by the email service are supported by the underlying library nodemailer. Check it out if you need something that's not explicitly supported by this service.

6.3.8.1 Configuring the service

The service is configured via the `wac.ini` file as usual. It contains a top-level entry named `[mailer]` that holds the service's details.

There is one mandatory option to be provided: `host`, which identifies the SMTP host.

Here we present the full list of available options:

host The host (name or IP address) of the SMTP server.

username The username for authenticating against the SMTP server.

password The password to use for the given username.

port The SMTP port to use (defaults to 587)

secure Whether to use TLS or not when talking with the SMTP. `false` still allow STARTTLS (defaults to false).

pooled Whether to use a connection pool or not (defaults to false)

from A forced From: address, ignoring each email's one. (defaults to the one given in the email object)

Note: Currently, only SMTP/S is supported

6.3.8.2 Creating new templates

Templates use the handlebars syntax. It's an easy and intuitive replacement syntax. Just wrap your items to be replaced with `{{ }}` and you're ready to go. Please take a look at provided template and website to learn more about it.

To have a template ready to use, please drop it in a new folder inside `templatePath` (variable under `[meetings]` entry) . For instance, if you want to add a new template for sending emails for meetings, you can create a new folder called `email` inside `templatePath` and copy there your template files.

6.3.8.3 Template naming rules

Template must adhere to strict but simple naming rules. As the email service allows to send both plaintext and HTML versions of an email, there has to be a template for each of those versions. The plaintext one is mandatory and the HTML is optional.

For the plaintext version, the file must be named `text.hbs`. The HTML template, is named `html.hbs`. If both templates are found, the email will have one alternate part for each of the versions.

For example, this might be the real contents of `templatePath`:

```
templatePath/
`- email/
   `- html.hbs
   `- text.hbs
```

6.3.8.4 Localization of templates

There is also support for localization of templates. From the administrator point of view it is as easy as creating a subdirectory in the template directory named after the locale to be used. Inside this folder we can also include two more files `from.hbs` and `subject.hbs` which are copied to the corresponding fields of the email. That is, for our example `email` template we can provide a Spanish translation by creating a folder `es` inside it with the corresponding localized templates.

```
templatePath/
`- email/
   `- html.hbs
   `- text.hbs
   `- es/
```

(continues on next page)

(continued from previous page)

```
       `- html.hbs
       `- text.hbs
       `- from.hbs
       `- subject.hbs
```

If no translation is requested, the `email` contents will be used.

Note: It's up to the service using the email service to ask for the localized versions of the email.

6.3.9 SendSMS service

Section contents

The `sendsms` service provides the ability to send SMS to mobile devices. This service may be used by other services that need to reach users via SMS.

The SMS will be created by already defined templates. These templates will include key words to be replaced by variable values.

In addition there is the possibility to create different SMS texts based on the language selected.

All the features provided by the sms service are supported by the underlying external service Beepsend. This external service has an API with which we can send SMS from the SippoAS.

To properly get this service working, it is necessary to create a profile on the provider platform. Check specific configurations on the corresponding backend configuration section.

6.3.9.1 Configuring the service

The service is configured via the `wac.ini` file as usual. It contains a top-level entry named `[sendsms]` that holds the service's details.

```
[sendsms]
backend[] = sendsms
; backend[] = bezeqsms
; backend[] = linkmobility
```

Optional paramters

- `templatePath`: This key will define the path where to look for templates (defaults to `<rootdirectory>/config/smstemplates`).

It can be used as an absolute path or as a relative path. E.g.

```
templatePath = ./otherfolder
```

In order to find the templates in `<rootproject>/config/otherfolder`.

6.3.9.2 Available backends

- **sendsms** send SMS using Beepsend API. The backend will connect using a connection token with the API and will send one or several messages. When a message has more than 70 characters, Beepsend will create two related messages.

```
[backend.sendsms]
module = sippo-sms-bs
urlService = https://api.beepsend.com
token = xxxxxxxxxxxxxxxxxxxxxxxxxx
from = Quobis
```

Backend parameters:

There are three mandatory options to be provided: `urlSevice`, `token` and `from`.

urlService: The BeepSend API URL.

token: Connection Token for authentication.

from: The sender id. It must be a string with one of the following formats:

- Alphanumeric. Maximum allowed characters are 11.
- MSISDN numbers or short numbers / codes. Between 9 and 17 chars.
- National format. Maximum length is 7 chars.

encoding: Allows you to specify the message encoding . Available options are UTF-8, ISO-8859-15 or Unicode. Defaults to UTF-8.

- **sippo-linkmobility** send SMS using the LinkMobility provider. To use it, enable it in the `wac.ini` file and set up the required parameters as follows:

```
[backend.linkmobility]
module = sippo-sms-linkmobility
login = username
password = <your_md5password>
from = QUOBIS
```

Backend parameters:

login Authentication parameter, as provided by LinkMobility

password Authentication parameter, as provided by LinkMobility. It MUST be the already MD5-hashed value.

from The string shown as the SMS originator, defaults to QUOBIS.

6.3.9.3 Creating new templates

Templates use the handlebars syntax. It's an easy and intuitive replacement syntax. Just wrap your items to be replaced with `{{ }}` and you're ready to go. Please take a look at provided template and website to learn more about it.

To have a template ready to use, please drop it in a new folder inside `templatePath`. For instance, if you want to add a new template for sending SMS for meetings, you can create a new folder called `myMeetingsSMS` inside `templatePath` and copy there your template files.

6.3.9.4 Template naming rules

Template must adhere to a strict but simple naming rules. As the sms service sends only plaintext, the plaintext template is mandatory. The file must be named `text.hbs`.

For example, this might be the real contents of `templatePath`:

```
templatePath/
`- myMeetingsSMS/
   `- text.hbs
```

6.3.9.5 Localization of templates

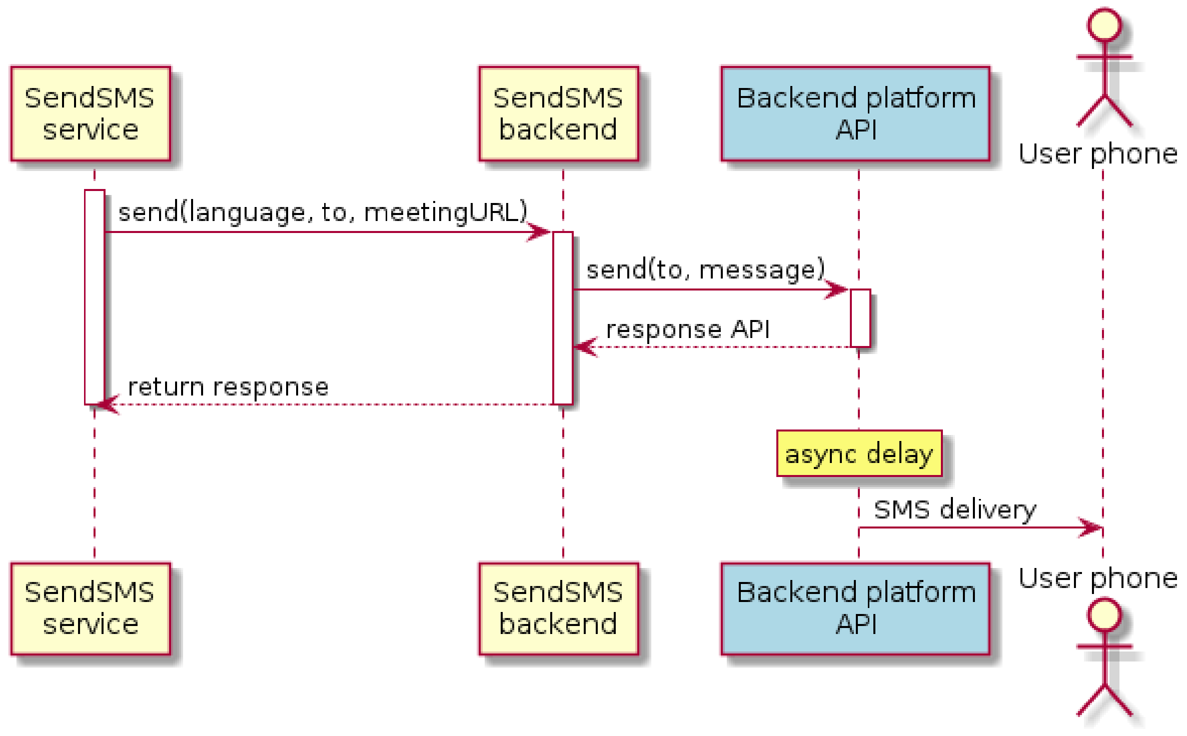
There is also support for localization of templates. From the administrator point of view is as easy as creating a subdirectory of the template directory named after the locale to be used.

That is, for our example `myMeetingsSMS` template we can provide a Spanish translation by creating a folder `es` inside it with the corresponding localized templates:

```
templatePath/
`- myMeetingsSMS/
   `- text.hbs
   `- es/
      `- text.hbs
```

If no translation is requested, the `myMeetingsSMS` contents will be used.

6.3.9.6 Activity diagram



6.4 QSS internal services reference

6.4.1 AudioMixersIO

Contents

6.4.1.1 Service name

- Formal name: AudioMixersIO
- Service name: SippoAS.qss.audiomixersio
- Service hierarchy: sippoas/qss/lib/services/audiomixersio

6.4.1.2 Main feature covered

This service was created to manage the connection with one or several *AudioMixer* instances.

Using the data given by the configuration file, the service will be able to connect to several *AudioMixer* servers, performs actions on them and listen to the events that the *AudioMixer* emits.

This service has a tight coupling to the *Trunk* service (`sippoas.qss.trunk`).

Note: This service is highly coupled to the Asterisk implementation of the *AudioMixer*

6.4.1.3 Related services and dependencies

- Sippo AS
  - Message Broker
  - Database
- Sippo framework dependencies
  - @quobis/ipc (Subscriber, Producer)
  - @quobis/log
  - @quobis/db

  - @quobis/errors
  - SippoAS.qss.trunk
- Sippo MS
  - AudioMixer
- External dependencies
  - asterisk-manager package (https://www.npmjs.com/package/asterisk-manager)

6.4.1.4 Service interaction description

This service will interact with the *Trunk* service through the *Message broker*.

When *Trunk* service sends a message to perform an action (for example, sends a INVITE method to the SIP network), it arrives to all running *AudioMixersIO* services. Then, each *AudioMixersIO* service has to decide if the message is for him or not. In order to know if it has to manage the message, the service checks if some of the *AudioMixer* that it manages is related with the incoming message. If it is, then the service will allows the action. Otherwise, the service will not process the message.

When an event arrives from one of the *AudioMixer* servers (for instance, a INVITE SIP method is received), the service will forward the event to the *Trunk* service via the *Message broker* and save some related data, like the Asterisk channel ID created, and an identifier of the *AudioMixer* instance source of the event (where the INVITE method was received). Although several instances of *Trunk* service are running, only one of them will process the event.

The following diagrams can help understanding better the flow between *Trunk* service and *AudioMixersIO* service:

Appendix B. Sippo internal UML diagrams

You can also expand information about the *Trunk* service at:

Trunk

Internal and external redirection

There are two internal processes on the *AudioMixers* where a call is moved or redirected. It is use an internal redirection when a resource is reallocated inside the AudioMixer instance. An external redirection is performed when it is required to move a resource from one AudioMixer to a different one.

For example, given two *AudioMixers*, called `AudOne` and `AudTwo` and an incoming call that arrives from SIP network and is processed by `AudOne`. After that, *Trunk service* creates a room and invites Alice (for the sake of brevity several process were skipped). The room created has its **AudioMixer audio room** associated in the `AudTwo`. So, when Alice accepts the incoming call, the incoming channel has to be inserted in the **AudioMixer audio room** associated to the room created. As the incoming channel belongs to a different *AudioMixer* than the one who manages the audio room, then a MOVE 302 action has to be performed to move the incoming channel to the *AudioMixer* where the **AudioMixer audio room** is.

6.4.1.5 REST API endpoints exposed

None

6.4.1.6 SippoSDK direct implications

None

6.4.1.7 Capability implications

None

6.4.1.8 Service Configuration

The configuration of this service is placed in the `config.json` file. There is an specific JSON node to configure the *AudioMixersIO* service:

```
{
  "audiomixersio": {
            "asterisk": {
                    "username": "quobis",
                    "password": "secret",
                    "context": {
                            "create": "quobis2pstn",
                            "redirectInternal": "quobis",
                            "redirectExternal": "call-transfer",
                            "sipToSipTransfer": "sip-to-sip-transfer"
                    },
                    "mixers": ["asterisk:5038"]
            }
      },
}
```

The meaning of each option is:

- `username`: credentials to connect with the Asterisk Manager Interface (AMI)
- `password`: credentials to connect with the AMI. All *AudioMixer* servers have to share the same username/password pair
- `context`: some actions to perform on the *AudioMixer* servers need a `context` field that has to be defined here and configured in the Asterisk dialplan
  - `create`: context used to create an invite to SIP net
  - `redirectInternal`: context used to insert a **AudioMixer channel** inside a audio room
  - `redirectExternal`: context used to perform a MOVE 302 action between **AudioMixers**, very common when several **AudioMixers** are deployed.
  - `sipToSipTransfer`: context used to be able to make a transfer when two SIP phone are involved
- `mixers`: list of *AudioMixer* IP:PORT that will be used via AMI

Note: All this parameters are required, when one of them is not given or with a invalid format, an error is returned and the service will not run up.

6.4.1.9 Logs: known log messages

Info messages

- `info: There is no asteriskId related with that resource:` This log appears when a message won't be processed by a *AudioMixersIO* service.

Error messages

- `error: Configuration file is not valid'` This log will appear some parameters to config the service is not given
- `error: Mixers it not well-formed. Expected ip:port list` This log will appear when the config option *mixers* is an empty array.
- `error: Mixer ${mixer} it not well-formed. Expected ip:port` This log will appear when the config option *mixers* has a value but it is not given with a list of ip:port

- `error: There is no AsteriskGateway that matches by AsteriskId` This log will appear when, even though the message that arrives to an *AudioMixersIO* contains a channel or `sipUri` that the *AudioMixersIO* can handle, for some reason the module inside the service that should be connected with the *AudioMixer* via AMI is not connected.
- `error: Invalid sipUri ${sipUri}` This log will appear when a message that arrives to the service contains a `sipUri` in an invalid format
- `error: Channel is not valid ${channel}` This log will appear when a channel sent in a message that arrives from *Trunk* doesn't have the format expected.

6.4.1.10 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify *Message broker* server is up and running.
- Verify the service is connected to the *Message broker* service.
- Verify that the configuration given is correct.

6.4.2 Conferences State

Contents

6.4.2.1 Service name

- Formal name: Conferences State
- Service name: SippoAS.qss.conferencesstate
- Service hierarchy: sippoas/qss/lib/services/conferencesstate

6.4.2.2 Main feature covered

This service is responsible for modeling and handling on going conferences, their state and events. When triggered by an event it takes actions over these conferences states.

List of conferences events:

- `ROOM_CREATED: 'conferenceState.roomCreated'`
- `ROOM_DESTROYED: 'conferenceState.roomDestroyed'`
- `INVITE_CREATED: 'conferenceState.inviteCreated'`
- `INVITE_DECLINED: 'conferenceState.inviteDeclined'`
- `INVITE_COMPLETED: 'conferenceState.inviteCompleted'`
- `USER_LEFT: 'conferenceState.userLeft'`

- `USER_JOINED: 'conferenceState.userJoin'`
- `FOREIGNER_JOINED: 'conferenceState.foreignJoin'`
- `TRANSFER_CREATED: 'conferenceState.transferCreated'`
- `TRANSFER_COMPLETED: 'conferenceState.transferCompleted'`
- `USER_DOWN: 'conferenceState.userDown'`
- `ERROR: 'conferenceState.conferenceStateError'`

Note: This service does not expose any type of interface and is nearly transparent to users.

6.4.2.3 Related services and dependencies

- @quobis/ipc (Subscriber, Producer)
- @quobis/log
- SippoAS.qss.rooms-basic
- SippoAS.qss.invites-rooms

6.4.2.4 Service Configuration

The configuration of this service is placed in the config.json:

```
// skipped code
"conferenceState": {
            "allowInvitesWhileInCall": true
    },
// skipped code
```

- `allowInvitesWhileInCall`: When this flag is true users can receive call invites while already on a call. This configuration option is dependent on *Rooms* service configuration.

6.4.2.5 Logs: known log messages

Level	Message
info	`Creating room state for the room ${roomId} with type ${type}`
info	`Destroying the room state of the room ${roomId}`
info	`pushing the invite ${inviteId} to the room ${roomId}`
info	`removing the invite ${inviteId}`
info	`removing the invite ${declinedInviteId}`
info	`user ${username} joined the room ${roomId}`
info	`foreign ${participantId} joined the room ${roomId}`
info	`user ${username} left the room ${roomId}`
info	`user ${username} is down cleaning conference state`
info	`A transfer with id ${transferId} was created in the room ${roomId}`
info	`Removing the transfer ${transferId}`
debug	`Deciding to destroy room ${room}`
info	`Destroying room ${roomId}`
debug	`can the user ${peer.username} be invited? ${canBeInvited}`
info	`Received request to list participants of room ${roomId}`
debug	`Participants of room ${roomId} are ${participants}`
debug	`Received request to get conference state for ${usersCount} number of users`
debug	`Permissions.canBeInvited:  ${peer}`

6.4.3 IO-websockets

Contents

6.4.3.1 Service name

- Formal name: IO
- Service name: SippoAS.qss.io-websockets
- Service hierarchy: sippoas/qss/lib/services/io-websockets

6.4.3.2 Main feature covered

This service is who exposes a websocket server which the clients can connect with QSS.

6.4.3.3 Related services and dependencies

- RabbitMQ
- @quobis/ipc (EventConsumer)
- @quobis/peers (PeersConsumer)
- @quobis/log
- @quobis/errors
- PeerService (qss/lib/services/peer)
- RegistryService (qss/lib/services/registry)

6.4.3.4 Service interaction description

If each message that arrives follows the JT protocol, the message is forward towards the `peerService`.

6.4.3.5 REST API endpoints exposed

None

6.4.3.6 SippoSDK direct implications

None

6.4.3.7 Capability implications

None

6.4.3.8 Service Configuration

The configuration of this service is placed in the `config.json`:

```
// skipped code
"io": {
  "websockets": {
    "iface": "0.0.0.0",
    "port": 8118,
    "protocol": "jt",
    "ssl": {
      "key": "/tls/key.pem",
      "cert": "/tls/cert.pem"
    }
  }
},
// skipped code
```

Note:

- If a new instance is going to be launched in the same server then we must use a different port.
- If the instance is going to be launched in a different server then the same port can be used.
- The Reverse Proxy must balance the load in a sticky way.

Where the meaning of each field is:

- **iface**: IP where the websocket server will be launched
- **port**: port where the websocket server will be launched
- **protocol**: protocol expected
- **ssl**: wait X milliseconds before removing a peer from the room when
  - **key**: Private key
  - **cert**: Public certificate of the exposed

6.4.3.9 Logs: known log messages

Level	Message
Info	`Connection from origin ${request.origin} rejected`
Info	`client connected with id ${peer.id}`
Info	`drop connection`
Info	`client disconnected with reason ${resonCode}/${description}`

6.4.3.10 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify RabbitMQ server is up and running.
- Verify the service is connected to RabbitMQ.
- Verify a new server in the port specified in the configuration was launched

6.4.4 Invites

Contents

6.4.4.1 Service name

- Formal name: invites
- Service name: SippoAS.qss.invites-rooms
- Service hierarchy: sippoas/qss/lib/services/invites

6.4.4.2 Main feature covered

This service is in charge of notifying to a user when other one wants to invite him to a room.

6.4.4.3 Related services and dependencies

- RabbitMQ
- @quobis/ipc (EventProducer, EventConsumer)
- @quobis/log
- @quobis/db
- @quobis/errors
- PeerService (qss/lib/services/peer)
- RegistryService (qss/lib/services/registry)

- ConferenceState (qss/lib/services/conferenceState)
- Resolver (qss/lib/services/resolver)
- AgentAssigner (sippo-server/services/AgentAssigner)

6.4.4.4 Service interaction description

This service creates and updates invitations (invite/inviteRequest methods):

- The invite method is sent to another peer to invite to join sender room. This is the method commonly used to start the calls.
- The inviteRequest method is sent to another peer in order to ask for an invite to join receivers room. This method is used for anonymous call from link, when a anonymous user wants to insert in a meeting or to establishing incoming calls from PSTN.

The invite services leverages the Push service exposed by sippo-server to send Push notifications to mobile devices.

6.4.4.5 REST API endpoints exposed

None

6.4.4.6 SippoSDK direct implications

None

6.4.4.7 Capability implications

None

6.4.4.8 Service Configuration

The configuration of this service is placed in the `config.json`:

```
// skipped code
"invites": {
  "basic": {
    "removeAfter": 10000,
    "resolveGroups": false,
    "resolvePush": false,
    "legacyUserGroupResolution": true,
    "wac": {
      "address": "http://wac:8000",
      "username": "admin@quobis",
      "password": "<YOUR_PASSWORD_HERE>"
    }
  },
  "parallelSip": false,
  "onlyOneDeviceWithInvitations": false
}
// skipped code
```

Where the meaning of each field is:

- **resolvePush**: when set to true, invites will try to send invitation by push to the users who are logged in a mobile phone. Will be ignored if **legacyUserGroupResolution** is set to false
- **wac**: sippo-server API URL and credentials to be able to interact with the Sippo AS Push service.

  - **address**: sippo-server URL
  - **username**: administrator user username
  - **password**: administrator user password

- **removeAfter**: wait X milliseconds before removing a candidate from the invite when that candidate (a registry object) unregisters or loses connection with qss. His value as default should be 0.
- **resolveGroups**: when is true, invites service will be capable of retrieving the participants in a group to invite them to a conference. It's essential to perform the 'invite a group' feature. Will be ignored if **legacyUserGroupResolution** is set to false
- **legacyUserGroupResolution**: when is true, invites service will retrieve candidate information as needed from the *sippo-server*, when false it will use the *Reachability* service in the *sippo-server* to get all the candidate information that the *sippo-server* could provide just in one request, significantly improving performance. For backwards compatibility if not specified it is set to true.
- **parallelSip**: This flag enables the parallel sip behavior. When it is activated, an invite is sent to the PSTN net and it come back before inviting to the room to the person who was invited. When this flag is false, invites service works as normally.
- **onlyOneDeviceWithInvitations**: when is true avoids that a user can establish two calls in differents devices. When the user is on two devices (mobile and desktop for example) and has an ringing invite in each device, when the user accepts a invite, the other one will be stopped by this service.

6.4.4.9 Logs: known log messages

Level	Message
Debug	`invite peerId(${peer.id}), to(${to}), inviteId(${inviteId}), context(${JSON.stringify(context)})`
Debug	`Trying to get candidates for ${to}`
Debug	`Trying to resolve group ${to}`
Debug	`got candidates ${candidates}`
Debug	`Notifying that user ${userId} joins to the room ${roomId}`
Debug	`INVITE_COMPLETED will be notified`
Debug	`Notifying that invite ${inviteId} for room ${roomId} was accepted by ${whoAccepted}`
Debug	`Notifying user ${userId} leaves ${roomId}`
Debug	`Cleaning data for room:  ${roomIdToClean}`
Debug	`accept peerId(${peer.id}), from(${from}), inviteId(${inviteId})`
Debug	`decline peerId(${peer.id}), from(${from}), inviteId(${inviteId}), errorCode(${errorCode})`
Debug	`"send invitation cancel", invite, "to", candidate`
Debug	`"send invitation", invite, "to", candidate`
Debug	`"will reject pending invitations of", item, "except", peer`
Debug	`"resuming invite", type, peer, message`
Debug	`stopInvite with the inviteId ${inviteId} for the callee ${callee}`
Debug	`"will update invite candidates with registry", invite, registry`
Debug	`"onPeerDisconnected will remove as candidate from", invites`
Debug	`No candidates found for resource:  ${to}`
Debug	`Resource '${to}' is a ${candidatesType} with candidates:  ${candidates}`

6.4.4.10 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify RabbitMQ server is up and running.
- Verify the service is connected to RabbitMQ.
- Url of sippo-server API is correct.

6.4.5 Meeting signaling

Contents

6.4.5.1 Service name

- Formal name: Meeting signaling
- Service name: SippoAS.qss.meeting-basic
- Service hierarchy: sippoas/qss/lib/services/meeting

6.4.5.2 Main feature covered

This service is in charge of create a room for a meeting created via `sippo-server`.

6.4.5.3 Related services and dependencies

- RabbitMQ
- @quobis/ipc (EventConsumer)
- @quobis/log
- @quobis/db
- @quobis/errors
- SippoAS.qss.rooms-basic
- SippoAS.sippo-server.Meetings

6.4.5.4 Service interaction description

This service manages rooms for the meetings created by meetings service in sippo-server. This service will receive a meeting Uri (wac-meeting:idMeeting) from a user request. He will validate against sippo-server that the meeting is valid and then will try to get a room for the meeting. If someone already accessed to the same meeting (using the same meetingUri), the room was already created, so the service will only get the room. In another case, the service will use the rooms service to create a room. After that, meeting service will insert the user in the ACL of the room and the room will be returned.

Moreover, this service provides the feature of joining a meeting using his DDI through PSTN.

6.4.5.5 REST API endpoints exposed

None

6.4.5.6 SippoSDK direct implications

None

6.4.5.7 Capability implications

None

6.4.5.8 Service Configuration

The configuration of this service is placed in the `config.json`:

```
// skipped code
"meeting": {}
// skipped code
```

Note: this service doesn't need more arguments as configuration.

6.4.5.9 Logs: known log messages

Level	Message
Error	*An error ocurred while trying to obtain the meeting info by the pin ${pin}*, error
Warn	`Meeting with pin ${pin} not found`
Info	`Trying to get meeting room by pin ${pin}`
Info	`Meeting room of pin ${pin} is ${meetingRoom.roomId}`
Debug	`Peer ${peer.id} wants to join to ${meetingUri}`
Debug	`add the user ${username} to the acl of the room ${roomId}`
Debug	*Obtained meeting for pin ${pin}*, meeting
Debug	`Trying to know if the meeting ${meetingId} is valid`

6.4.5.10 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify RabbitMQ server is up and running.
- Verify the service is connected to RabbitMQ.

6.4.6 Quick Conference

Contents

6.4.6.1 Service name

- Formal name: Quick Conference
- Service name: SippoAS.qss.quickConference.QConference
- Service hierarchy: sippoas/qss/lib/services/quickConference

6.4.6.2 Main feature covered

This service is in charge of getting a room when a user wants to join one by using a conference URI.

6.4.6.3 Related services and dependencies

- RabbitMQ
- @quobis/ipc (EventConsumer)
- @quobis/log
- @quobis/db
- @quobis/errors
- SippoAS.qss.rooms-basic

6.4.6.4 Service interaction description

This service is in charge of getting a room when a user wants to join one by using a conference URI. This service will receive a conference URI from a user.

This service supplies as well a mechanism to validate if the conference URI used to get a room is valid. The conference URI needs to match with the configuration param `uriRegex` defined in the quickConference setting in the config.json file.

6.4.6.5 REST API endpoints exposed

None

6.4.6.6 SippoSDK direct implications

None

6.4.6.7 Capability implications

None

6.4.6.8 Service Configuration

The configuration of this service is placed in the `config.json`:

```
{
 "quickConference": {
                        "uriRegex": "^wac-conf\\d{20}$"
              }
}
```

Note: this service doesn't need more arguments as configuration.

6.4.6.9 Logs: known log messages

Level	Message
Debug	*Peer ${peer.id} wants to join ${conferenceUri}*

6.4.6.10 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify RabbitMQ server is up and running.
- Verify the service is connected to RabbitMQ.

6.4.7 Rooms

Contents

6.4.7.1 Service name

- formal name: rooms
- service name: SippoAS.qss.rooms-basic
- Service hierarchy: sippoas/qss/lib/services/rooms

6.4.7.2 Main feature covered

This service manages the creation/update/deletion of rooms from the QSS in the Janus instance through dispatcher.

6.4.7.3 Related services and dependencies

- RabbitMQ
- @quobis/ipc (EventProducer, EventConsumer)
- @quobis/log
- @quobis/errors
- PeerService (qss/lib/services/peer)
- RegistryService (qss/lib/services/registry)
- ConferenceState (qss/lib/services/conferenceState)

6.4.7.4 Service interaction description

This service is highly configurable. According to the configuration, we can fit some customer-specific requirement. For example, we could indicate in the config.json file if a room has to have an owner or not or how much participants can be inside a room.

6.4.7.5 REST API endpoints exposed

None

6.4.7.6 SippoSDK direct implications

None

6.4.7.7 Capability implications

None

6.4.7.8 Service Configuration

The configuration of this service is placed in the config.json:

```
// skipped code
"rooms": {
  "collectionType": "janus",
  "setOwner": true,
  "updateRoomOnwerWhenLeaves": false,
  "janusAddress": "http://dispatcher:8020",
```

(continues on next page)

(continued from previous page)

```
  "removeAfter": 0,
  "participantsLimit": 5,
  "notInACall": true,
  "userIdAsRoomId": true
},
// skipped code
```

Where the meaning of each field is:

- janusAddress: URL of the dispatcher to be contacted for room management.
- removeAfter: wait X milliseconds before removing a peer from the room when that peer (a registry object) unregisters or loses connection with qss. His value as default should be 0 (leave room immediately).
- participantsLimit: max number of participants which can be inside a room. When this num is reached, an error is returned when adding a new user to the room.
- notInACall: When this flag is false, a user can create such rooms as he wants. If this flag is true, a user will be able to create a new room depending on how the conference state service was configured.
- userIdAsRoomId: When this flag is true, the roomId will be equals to the username of who creates the room.
- collectionType: indicate if it is used the dispatcher or a mongo collection to save data related with the rooms. In order to “janusAddress” makes sense, this key has to be filled with “janus”.
- setOwner: If this flag is true, the rooms have an owner and only an owner could perform some actions in the room.
- updateRoomWhenOwnerLeaves: If this flag is true, when the owner of the room leaves it then a new owner is chosen from the available users in the room.

6.4.7.9 Logs: known log messages

Level	Message
Debug	*“create”*, peer, record
Debug	*“update”*, peer, id, acl, record, dryRun
Debug	*“close”*, peer, id,
Debug	*“leave”*, peer, id,
Debug	*“destroying the room”*, roomId
Debug	*“try to know if the user can create a room”*, peer

6.4.7.10 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify RabbitMQ server is up and running.
- Verify the service is connected to RabbitMQ.
- Verify the dispather is running in the address specified in config.json

6.4.8 Trunk

Contents

6.4.8.1 Service name

- Formal name: Trunk
- Service name: SippoAS.qss.trunk
- Service hierarchy: sippoas/qss/lib/services/trunk

6.4.8.2 Main feature covered

This service is designed to manage all business logic related with calls to and from the SIP network. In this service, it is processed and saved relevant information for a call that goes through the SIP network.

Review the *Appendix B. Sippo internal UML diagrams* section to follow the explanation of this service.

SipEntry

`SipEntry` is a new concept added with this service. It is way of referring to the data saved in the *Trunk* service that is required to:

- identify whom created a call through the SIP net,
- on a parallel sip environment, the direction of the call (incoming: from SIP to system, or outgoing: from system to SIP)

There are other other important internal data that is stored on this `SipEntry`.

```
SipEntry {
  id: string;
  requestID: string;
  from: string;
  to: string;
  sipUri: string;
  roomId: string;
  type: 'incoming' | 'outgoing';
  status: 'initial' | 'active';
  channel: string;
  transferring: boolean;
  callID: string;
  XWebRTCSession: string;
  parallelSipOutgoing: boolean;
```

(continues on next page)

(continued from previous page)

```
  parallelSipIncoming: boolean;
}
```

Channel

Another key concept is `Channel`. It identifies the resource created by the *AudioMixer* to insert the SIP audio inside a conference room.

For example, when a call is created towards SIP `INVITE`, a `Channel` is created and saved in a `SipEntry`. After that, the call can be stopped from the application side, sending a `hangup()` command that results on a SIP `HANGUP` method that is sent using the `Channel` data on the corresponding *AudioMixer* instance. Other SIP actions like send DTMF or transfer the channel to another destination requires the `Channel` data.

Note: It is important to remark that each `Channel` must be associated with a `SipEntry`.

Parallel SIP implications

Regarding *Parallel sip* environment, the main idea is that each participant in a conference will be placed in his own *AudioMixer* audio room, but all participants in the conference will be placed in the same *SFU* video room. To handle participants location correctly, the `prefix.parallelSip` option has to be configured in this service.

A point to outline is that each room created (using the *Dispatcher* service) will have an associated `sipUri` that is formed by the prefix `888`, 8 numbers, an `@` symbol and the IP address where the *AudioMixer* is listening (for instance, sip:88812345678@ip-asterisk). The 8 numbers are the identifier of the *AudioMixer*'s room audio.

Table 1: sipUri syntax

`888`	`12345678`	`@`	`ip-asterisk`
PREFIX	AudioMixer room audio		AudioMixer IP

That being said, to avoid that Bob gets inside Alice room audio, the invite sent to Bob contains a new field with the `sipUri` that Bob client has to use to join to the *AudioMixer*. This `sipUri` is related with the Alice audio room on the next way:

`${prefix.parallelSip}888xxxxxx12345678@ip-asterisk`

Being `prefix.parallelSip` an extension that has to be defined in dialplan of the *AudioMixer* and matches with value defined in the key `prefix.parallelSip` in the `config.json` file. The section `xxxxxx` are six random numbers created on runtime to get a unique parallel sip audio room for each participant that is invited in this environment. So finally, when Bob accepts the incoming call and join the conference, he will be inserted in his own audio room (`xxxxxx12345678`).

This service has a tight coupling to the *AudioMixersIo* service, *Peer* Service and *Invites* Service.

More info:

- *Invites*
- *AudioMixersIO*

6.4.8.3 Related services and dependencies

- Sippo AS
  - Message Broker
  - Database
- Sippo framework dependencies

  - @quobis/ipc (Subscriber, Producer, Consumer, Publisher, EventConsumer, RPCProxyClient, RPCClient, EventProducer)
  - @quobis/log
  - @quobis/db
  - @quobis/errors
  - SippoAS.qss.audiomixersio
- Sippo MS
  - AudioMixer

6.4.8.4 Service interaction description

This service will interact with *AudioMixersIo*, *Peer* and *Invites* service through the *Message Broker* service.

When a message arrives via the *Message Service* to the service, only one of the running *Trunk* services will process the message and execute the needed business logic.

In order to allow a user to send an invitation to SIP network, the Trunk service has to be successful create an identifier at the *Database* in the initialization process of the service. This identifier will be used by *Invites* service to know if the *Trunk* service is ready to receive messages.

To know more about the interaction between services with the other services, have a look to the following diagrams:

Appendix B. Sippo internal UML diagrams

6.4.8.5 REST API endpoints exposed

None

6.4.8.6 SippoSDK direct implications

None

6.4.8.7 Capability implications

None

6.4.8.8 Service Configuration

The configuration of this service is placed in the `config.json`:

```
{
  "trunk": {
    "prefix": {
        "parallelSip": "777"
    },
    "match": "<regularExpression>"
  },
}
```

The meaning of each option:

- `prefix`

  - `parallelSip`: prefix that will be used to create the parallel sip room. If this field is undefined, the code `777` will be used. The value of this field has to be related with the *AudioMixer* configuration.

- `match`: so that a call is sent towards SIP net, the to part has to match with the regular expression added in this field. If this field is not given, `^tel: .*$` expression will be used.

6.4.8.9 Logs: known log messages

Info messages

- `info: Trunk Registry was created with success`: This log appears when a registry for Trunk service is created when initializing the service.
- `info: Trunk service was initialized with this config: match (${match}) parallelSipPrefix (${prefix.parallelSipPrefix})`: This log appears during the setup process when the service configuration was validated and it is available to use in the service.

6.4.8.10 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify that the *Message Broker* server is up and running.
- Verify that the service is connected to the *Message Broker*.
- Verify that in *Database* there is a registry of `pstn` type.

6.4.9 WatchDogInvites

Contents

6.4.9.1 Service name

- Formal name: WatchDogInvites
- Service name: SippoAS.qss.watchdog.invites
- Service hierarchy: sippoas/qss/lib/services/watchdog/invites

6.4.9.2 Main feature covered

This service was created to stop active invites (in ringing state) after a configurable period of time.

Using the data given by the configuration file, the service will periodically check if there are active invites and stop them if they exceed the configured expiration time.

This service has a tight coupling to the *Invites* service (`sippoas.qss.invites-rooms`).

6.4.9.3 Related services and dependencies

- Sippo framework dependencies
  - @quobis/ipc (RPCServerIPC, RPCClientIPC)
  - @quobis/log
  - @quobis/error
  - SippoAS.qss.invites
- QSS dependencies
  - Invites

6.4.9.4 Service interaction description

Given Alice and Bob as to users from the system. When Alice calls Bob, the *Invites* service from the QSS creates an invite to keep that transaction state.

Lets say the WatchDogInvites service is configured with a 5 second period and the expiration time is set to 30 seconds. When 30 seconds have elapsed since the invite's creation the service will detect the invite is in a ringing state and will try to decline it, canceling the transaction both for Alice and for Bob.

The service's aim is to have a way of switching from scenarios where the invite system is handled locally or externally. In an externally handled context Sippo wac does not know both, what the user will do on its SIP network and the invites duration, thus this service must be deactivated

6.4.9.5 REST API endpoints exposed

None

6.4.9.6 SippoSDK direct implications

None

6.4.9.7 Capability implications

None

6.4.9.8 Service Configuration

The configuration of this service is placed in the `config.json` file. There is an specific JSON node to configure the *WatchDogInvites* service:

```
{
"watchdog": {
                    "invites": {
                           "period": 5000,
                           "expiration": 30000
                    }
             },
}
```

The meaning of each option is:

- `period`: Indicates the time period for the service to check for active invites.
- `expiration`: Indicates the minimum time an invite will be in the ringing state since it was created, before it is stopped by the service.

Note: All this parameters are required, when one of them is not given or with a invalid format, an error is returned and the service will not run up.

6.4.9.9 Logs: known log messages

Debug messages

- `debug: invite declined`: This log appears when a invites exceeds it s expiration time and is successfully declined by the service.

Error messages

- `error: got error declining invite` This log will appear when the service detects an expired invite but fails to decline it.
- `error: watchdog invites error: <err>` This log will appear when the service fails to retrieve the active invites.

6.4.9.10 Troubleshooting & caveats

If the service is not working as expected, a few steps to follow:

- Verify *Invites* service is up and running.
- Verify that the configuration given is correct.

CHAPTER

SEVEN

TROUBLESHOOTING

7.1 Logs

Sippo Wac services are run through Kubernetes and Docker containers and print log messages to the standard output.

Logs produced by the different services that make up the cluster are often the best place to start when troubleshooting a variety of issues. This section explains how to obtain logs for these purposes.

7.1.1 Docker based deployment

In a bash shell on the machine running the docker images type the following command:

```
docker ps
```

This will output a list of all the running containers in that machine with its associated container ID.

To show logs for all the running processes running on the cluster, in a shell in the same directory as your docker compose file, type the following:

```
docker-compose -f logs
```

Although each one of the containers is represented by a different color, visualizing logs from all the different services at once can be confusing so the container ID can be added to the previous command to filter results:

```
docker -f logs <container-id>
```

7.1.2 Kubernetes based deployment

A complete description on how to obtain pod logs for the kubernetes based deployment can be found *Initial steps after installation*

7.2 Sippo Server logs

Sippo server manages most of the logic so a good strategy when dealing with troubleshooting is to start with this container. To access its logs do as with any other service.

7.2.1 Logging format

Sippo server logs follow the next format:

```
[UTC-time-timestamp] [File/path/generating/the/log] logLevel: Message
```

7.2.2 Log levels

Each log message has an associated log level which indicates the severity of the information being logged:

Level	Description
silly	More detailed service operation messages than debug
debug	Detailed service operation messages to help troubleshoot issues
info	Information messages that represent service actions
warn	Warning messages that indicate potential issues
error	Service error messages

Sippo server's default logging level is set to *debug* and can be changed through the `config/default.toml` file as shown in the following example.

```
[log]
level = "info"
relativePaths = true
```

Note: Logging level cannot be changed during run time. This means the cluster has to be destroyed and deployed again for changes to take effect.

7.2.3 Elasticsearch configuration

Elasticsearch (ES) is an indexing engine that is built specifically to have almost real-time search results, optimised for searching text strings. It is built in Java and runs as a separate server/process.

Sippo can use Elasticsearch to organize the log data from a Sippo-server instance into datastores, or indices. Then, through a Kibana web UI users and administrators can create rich visualizations and dashboards with the aggregated data.

The sending of logs to a Elasticsearch instance is enabled in the same `config/default.toml` file, where the URI of the Elasticsearch instance must be configured:

```
# When uncommenting these options, the logs will send to an Elastic Search instance.
# When this option is uncommented, it must specify the host and the index to use. The
# level of the log to send to elasticSearch can be specified with the option 'level'.
# Note: Elasticsearch uses a mmapfs directory by default to store its indices. The default␣
↪operating
# system limits on mmap counts is likely to be too low, which may result in out of memory␣
↪exceptions.
# More info: https://www.elastic.co/guide/en/elasticsearch/reference/current/vm-max-map-count.
↪html
#
#[log.elasticSearch]
#host = "http://elasticsearch:9200"
#index = "wac"
#level = "silly"
```

7.2.4 Push Notifications

Push notifications provide a mechanism to send messages to clients through an auxiliary channel provided by the system, allowing the application to not have always an open connection.

This auxiliary channel can be provided by Apple Push Notification Service (for iOS devices) or by Firebase Cloud Messaging (for Android devices).

If correctly set up this feature prints logs for the following scenarios:

1. User **<userId>** notifies of new device token when it logs in a new device

```
[2020-02-27 10:48:15.069] [lib/services/PushNotifications/PushNotificationsService.js] debug:␣
↪Received new token notification by userID. Token: <token>, tokenType: <tokenType>, deviceId:
↪<deviceId>
```

2. Service requests device tokens for a given user and sends a push notification.

```
[2020-02-27 10:48:15.069] [lib/services/PushNotifications/PushNotificationsService.js] debug:␣
↪Received request to obtain the push token ids by the user <userId>
[2020-02-27 10:48:15.069] [lib/services/PushNotifications/PushNotificationsService.js] debug:␣
↪Received request to send the following notification <notification> to the <tokenId>
```

This can have two possible outcomes:

either the push notifications are delivered to the service provider

```
[2020-02-27 10:48:15.069] [lib/services/PushNotifications/PushNotificationsService.js] debug:␣
↪Push notification sent successfully with: token type <type> and token <token>
```

or the sippo server cannot connect APN FCM.

```
[2020-02-27 10:48:15.069] [lib/services/PushNotifications/PushNotificationsService.js] error:␣
↪Error sending Push notification to <APN/FCM> with: token type, <type>, token, <token> and␣
↪error <error>
```

The error sent back can mean a different thing for APN or FCM.

APN

- `405` Bad Request
- `403` There was an error with the certificate or with the provider authentication token
- `410` The device token is no longer active for the topic
- `413` The notification payload was too large
- `429` The server received too many requests for the same device token
- `500` Internal server error
- `503` The server is shutting down and unavailable

FCM

- `BadRequestError` MissingRegistration
- `BadRequestError` InvalidRegistration
- `BadRequestError` InvalidParameters
- `BadRequestError` InvalidDataKey
- `BadRequestError` InvalidTtl
- `BadRequestError` InvalidApnsCredential
- `BadRequestError` MismatchSenderId
- `BadRequestError` InvalidPackageName
- `BadRequestError` InvalidJSON
- `InvalidPushTokenError` NotRegistered

- `PayloadToLarge` MessageTooBig
- `InternalServerError` Unavailable
- `InternalServerError` InternalServerError
- `InternalServerError` Unknown FCM error
- `ToManyRequestsError` DeviceMessageRateExceeded
- `ToManyRequestsError` TopicsMessageRateExceeded
- `AuthorizationError` AuthorizationError

CHAPTER

EIGHT

HOW TO GET SUPPORT

Quobis is committed to provide a excellent post-sales support to partners and end customers. Technical support on all Quobis products can be obtained through the following methods:

- **Phone**: +34 902 999 465
- **Help desk**: http://support.quobis.com
- **Email**: support@quobis.com

Please note that you need a valid support contract to get access to the support services. Contact your sales representative for more information.

Appendix table:

CHAPTER

NINE

APPENDIX A. VMWARE OVA DEPLOY

Chapter contents

9.1 Introduction

On this guide we will start from a bare-metal server where we will install and configure everything to get a Sippo AS and a Sippo MS running over a single host VMware vSphere infrastructure. So we will get a all-in-one deployment.

Note: This deployment is only recommended for PoC, testing and development. For a production deploy, use Ansible installer and check the corresponding documentation.

The Sippo deployment covered in this guide will involve three steps:

- Installing VMware vSphere ESXi 5.x
- Configuring VMware vSphere ESXi networking
- Deploying Sippo OVA on the VMware vSphere ESXi

You must check your current infrastructure to continue on the appropriate section:

- VMware infrastructure already settled: Go to “Configure networking with vSphere Client”
- VMware network prepared for Sippo cluster deploy: Go to “Installing OVA Sippo package on the ESXi Host”

9.2 Installing VMware vSphere ESXi 5.x

Since the Sippo deployment could have (on the laboratory version) less requirements than VMware vSphere ESXi 5.x, remember that your server must support at least this minimum requirements on the resulting virtual machine:

Architecture	Intel x86 processors, 64 bits
Number of CPU cores	2x CPU core
Processor speed	2.0 GHz or higher
RAM	At least 8 GB memory
HDD	20 GB hard drive or higher
Network Interfaces	Single Ethernet 1000base-TX NICs
IP Addressing	One (1) IP address

Note: For a production environment please contact the Quobis sales team in order to review HW specifications according to your estimated traffic usage.

VMware vSphere ESXi 5.x installation instructions are as follows:

1. Register on VMware's website. A customer account is needed for download VMware products. Download the iso image from http://www.vmware.com/go/get-free-esxi.

   Note: For HP servers look for the HP version of the installer ISO of VMware vSphere ESXi.

   Note: On the License & Download section of the vmware section take note of your LICENSE KEY provided, it's needed to activate your ESXi free edition.

2. On your server use the downloaded and burned CD to install the ESXi version 5.x. NOTE: the installation of the VMware vSphere will format and delete HDD of your server.
3. Check your BIOS to select the boot device, in our case select CD/DVD as the predefined unit to boot from.
4. After boot (from CD/DVD), two boot options from CD/DVD could appear, choose 0 (DEFAULT) and press ENTER.
5. On boot GRUB/ISOLINUX select ESXi Installer and press Enter.
6. Continue installation process:
   1. Press ENTER to start installation process.
   2. Press F11 to accept license.
   3. Select the HDD disk to install ESXi and press Enter.
   4. If a previous ESX OS is found you can update or overwrite your current installation.
   5. Provide password for username "root". NOTE: It is not recommended, but password blank will be accepted, password can be set later on "Configure Management Network" option.
   6. Press F11 to confirm HDD partitioning and wait during install process.
7. When installation was complete remove the installation CD/DVD and reboot.
8. After boot up select F2 to customize system/setup.
9. Log into system using root credentials and navigate to "Configure management network"
   1. Choose "Network adapters" if you want to modify your physical NIC for ESXi use.
   2. Choose "IP configuration" for network setup (Static IP or DHCP)
   3. Explore other network options (DNS, IPv6) as needed.
10. Press Y (yes) to apply changes to management network.

11. At this stage, you should be able to connect to the VMware vSphere Client using the IP address provided in Step 12 and your root credentials defined in step 10.

    Note: A vSphere client for Windows could be downloaded using a web browser and pointing it to the vSphere management IP.

12. Follow to the next section to configure VMware vSphere network for your Sippo OVA deploy.

9.3 Configure networking with vSphere Client

Once the ESXi 5.0 is installed on the server, VM networking must be configure to connect VM to the physical network. To do that, vSwitches need to be created onto the host by using either vSphere or vCenter API/WEB client.

Note: The easier way to obtain a vSphere client for Windows is downloading it using a web browser from the vSphere management IP.

A minimal introduction about VMware networking is needed. For our purposes, we need to understand that one physical interface is reserved for the VMware (vmnic0), and the others could be used for Virtual Machines (VM image vSwitch).

These two functions (Kernel and VM usage) are separated on virtual ports. We will use VM ports and keep Kernel port by default.

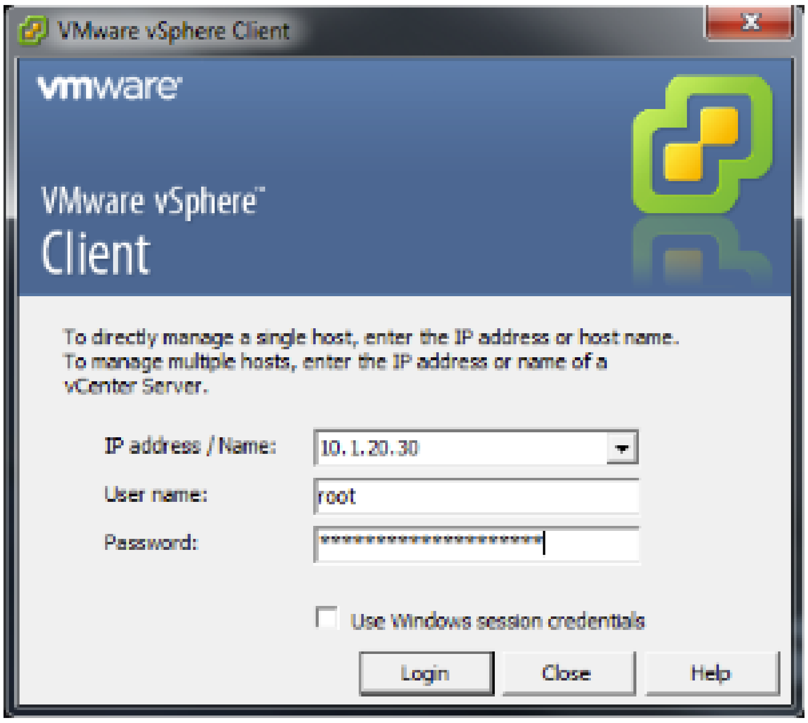
Another VMware concept is the vSwitch, that separate usages on the same physical interface. If you assign multiple physical interfaces to the same vSwitch these network interfaces will act in HA redundancy.

Finally, the Sippo OVA will come with three virtual interfaces (eth0, eth1, eth2), one is for management & SSH, the second for the PUBLIC service interface, and the third for the INTERNAL REST API services. Do your VMware networking having this planning in mind.

VMware vSphere comes with a default management vSwitch used for management purposes. Is the NIC where you connect for setup and other administrative tasks. Since we need that our Virtual Machines (VM) use some physical interface, we need to create a VM vSwitch dedicated for each network. Multiple networks could be tagged on the same physical NIC using 802.1q tagging.

In this example, a server with four physical NICs is assigned separate vSwitches in order to configure two different networks, one for the Sippo AS/MS public interface, and other one for the Sippo AS REST interface.

1. Load vSphere client on your PC.
2. Connect to ESXi host with IP address created in the Step 9 of previous section and login with root privileges.


VMware vSphere Client

VMware vSphere Client

To directly manage a single host, enter the IP address or host name.
To manage multiple hosts, enter the IP address or name of a vCenter Server.

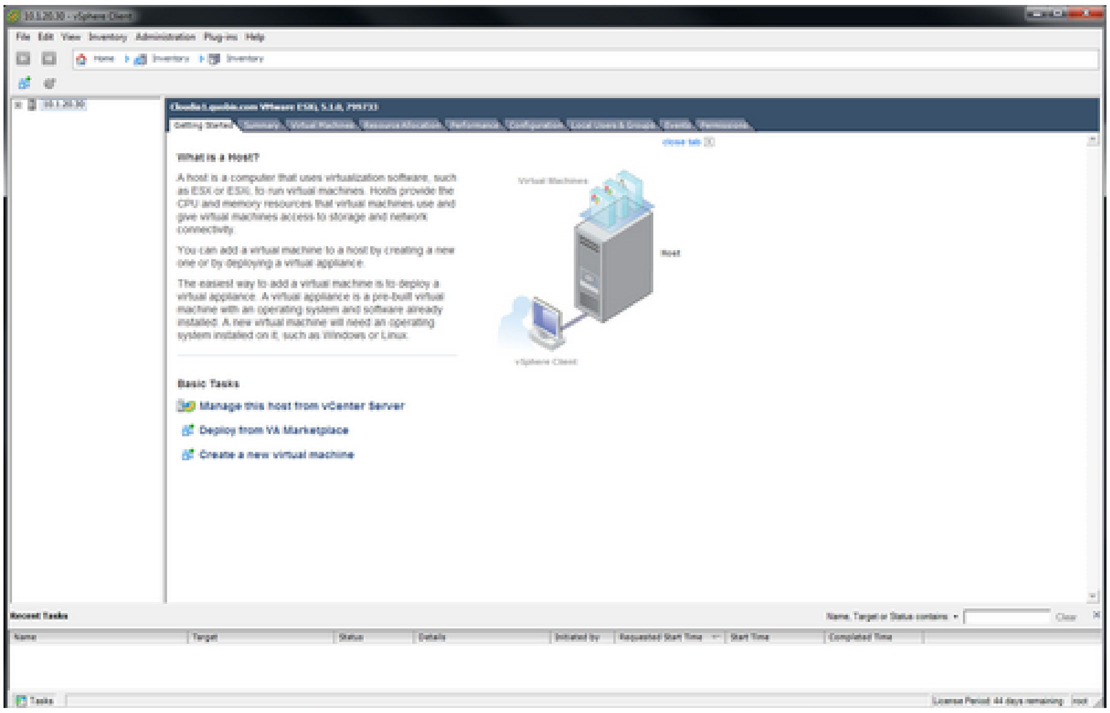
IP address / Name: 10.1.20.30

User name: root

Password: ********************

Use Windows session credentials

This host should show up at the left panel. Click on inventory if needed.


Cloudia1.qeobis.com VMware ESXi, 5.1.0, 799733

What is a Host?

A host is a computer that uses virtualization software, such as ESX or ESXi, to run virtual machines. Hosts provide the CPU and memory resources that virtual machines use and give virtual machines access to storage and network connectivity.

You can add a virtual machine to a host by creating a new one or by deploying a virtual appliance.

The easiest way to add a virtual machine is to deploy a virtual appliance. A virtual appliance is a pre-built virtual machine with an operating system and software already installed. A new virtual machine will need an operating system installed on it, such as Windows or Linux.

Basic Tasks

Manage this host from vCenter Server

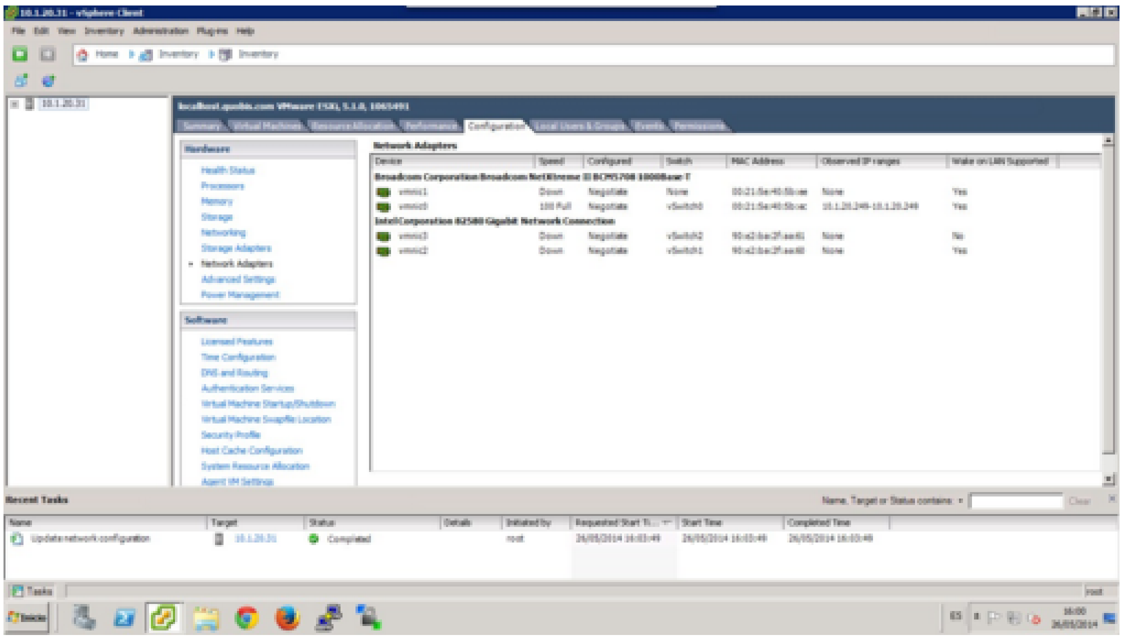
Deploy from VA Marketplace

Create a new virtual machine

Note: The first time that you connect to your ESXi Server, you must add a license of the ESXi, check the section 3.2.1 step 1 to find where to obtain your license key. Select the host, click on the Configuration tab and after it select “Licensed features”. Click “Edit. . . ” on the upper right corner to add your free license key.

3. Select the configuration and go to Network Adapters; in this server there are 4 NICs. The vSwitch0 is created by default and is shared NIC used for the ESXi kernel host and the management net-

work. This image show your physical network adapters, check connectivity, speed negotiation and other parameters to ensure your network connection.



10.1.20.31 - vSphere Client

localhost.quobis.com VMware ESXi, 5.1.0, 1065491

Summary | Virtual Machines | Resource Allocation | Performance | Configuration | Local Users & Groups | Events | Permissions

Hardware: Health Status, Processors, Memory, Storage, Networking, Storage Adapters, Network Adapters, Advanced Settings, Power Management

Software: Licensed Features, Time Configuration, DNS and Routing, Authentication Services, Virtual Machine Startup/Shutdown, Virtual Machine Swapfile Location, Security Profile, Host Cache Configuration, System Resource Allocation, Agent VM Settings

Network Adapters

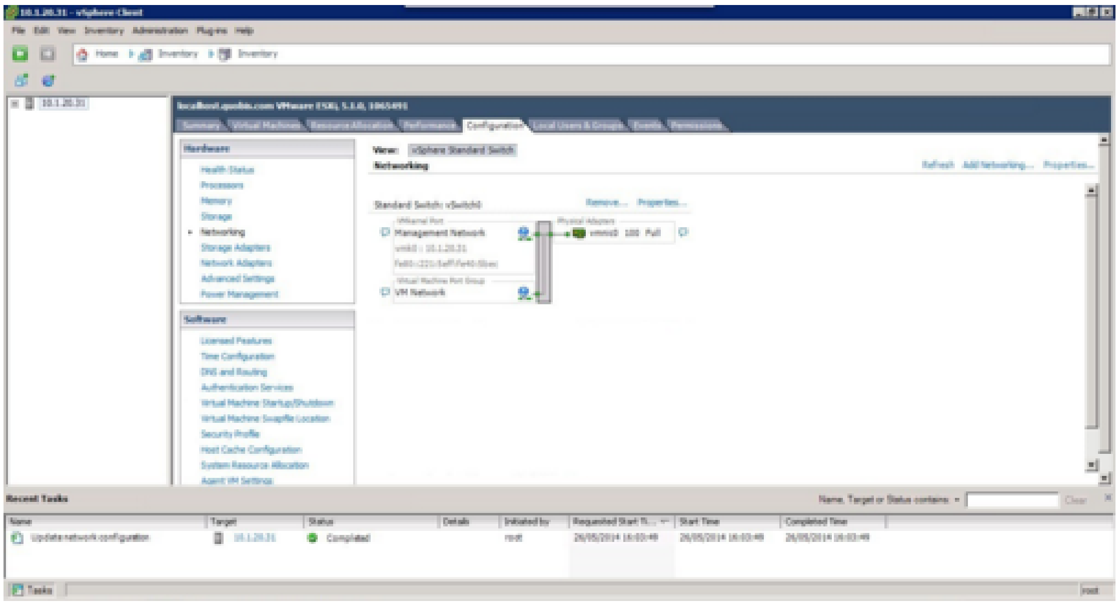
Device	Speed	Configured	Switch	MAC Address	Observed IP ranges	Wake on LAN Supported
Broadcom Corporation Broadcom NetXtreme II BCM5708 1000Base-T						
vmnic1	Down	Negotiate	None	00:21:5e:40:5b:ae	None	Yes
vmnic0	100 Full	Negotiate	vSwitch0	00:21:5e:40:5b:ac	10.1.20.249-10.1.20.249	Yes
Intel Corporation 82580 Gigabit Network Connection						
vmnic3	Down	Negotiate	vSwitch2	90:e2:ba:2f:ae:61	None	No
vmnic2	Down	Negotiate	vSwitch1	90:e2:ba:2f:ae:60	None	Yes

Recent Tasks

Name	Target	Status	Details	Initiated by	Requested Start Ti...	Start Time	Completed Time
Update network configuration	10.1.20.31	Completed		root	26/05/2014 16:03:49	26/05/2014 16:03:49	26/05/2014 16:03:49

4. Next step is to configure network vSwitches of type virtual machine, not kernel mode. These will be used for Sippo AS/MS public interface and Sippo AS REST API respectively. For that, follow the next points:

   Select Networking on the left tab, it gives info about the default vSwitch0 which is the mandatory management network. The next screen should appear:



10.1.20.31 - vSphere Client

localhost.quobis.com VMware ESXi, 5.1.0, 1065491

View: vSphere Standard Switch

Networking

Refresh | Add Networking... | Properties...

Standard Switch: vSwitch0 — Remove... | Properties...

VMkernel Port: Management Network — vmk0 : 10.1.20.31 — fe80::221:5eff:fe40:5bac

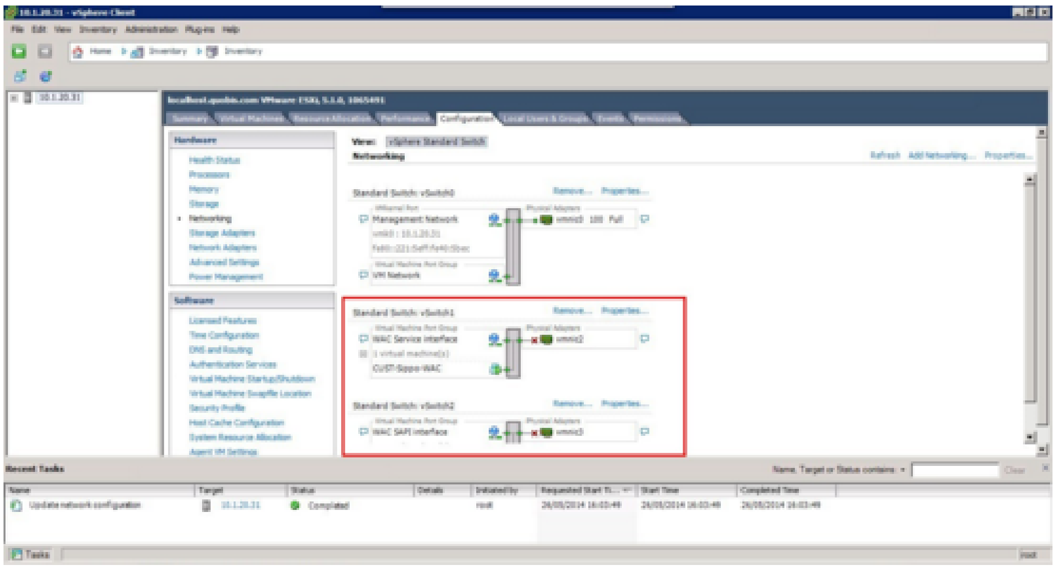
Virtual Machine Port Group: VM Network

Physical Adapters: vmnic0 100 Full

Recent Tasks

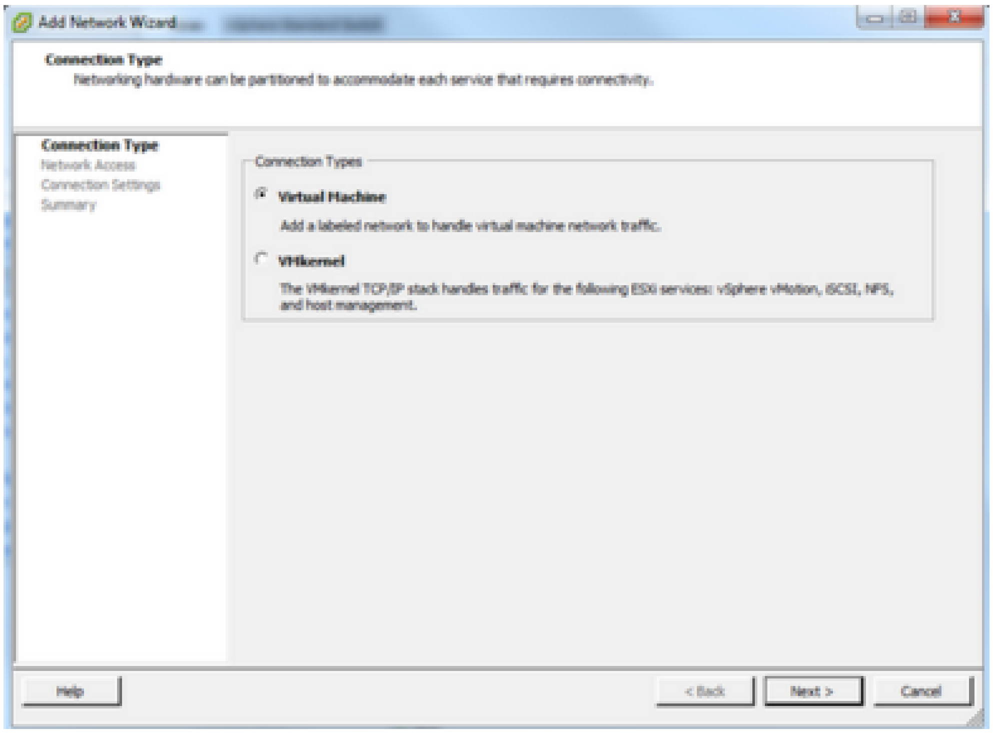
Name	Target	Status	Details	Initiated by	Requested Start Ti...	Start Time	Completed Time
Update network configuration	10.1.20.31	Completed		root	26/05/2014 16:03:49	26/05/2014 16:03:49	26/05/2014 16:03:49

Our objective is to create two new vSwitches as you can see on the next screenshot:


10.1.20.31 - vSphere Client
File Edit View Inventory Administration Plug-ins Help
Home Inventory Inventory
localhost.quobis.com VMware ESXi, 5.1.0, 1065491
Summary Virtual Machines Resource Allocation Performance Configuration Local Users & Groups Events Permissions
Hardware
Health Status
Processors
Memory
Storage
Networking
Storage Adapters
Network Adapters
Advanced Settings
Power Management
Software
Licensed Features
Time Configuration
DNS and Routing
Authentication Services
Virtual Machine Startup/Shutdown
Virtual Machine Swapfile Location
Security Profile
Host Cache Configuration
System Resource Allocation
Agent VM Settings
View: vSphere Standard Switch
Networking
Refresh Add Networking... Properties...
Standard Switch: vSwitch0 Remove... Properties...
Management Network
VM Network
vmnic0 100 Full
Standard Switch: vSwitch1 Remove... Properties...
WAC Service interface
CUST-Sippo-WAC
vmnic2
Standard Switch: vSwitch2 Remove... Properties...
WAC SAPI interface
vmnic3
Recent Tasks
Update network configuration 10.1.20.31 Completed root
Tasks

To do that, follow next steps:

1. Select “Add Networking” on the top right corner.
2. Select “Virtual machine” option and click next.


Add Network Wizard

Connection Type
Networking hardware can be partitioned to accommodate each service that requires connectivity.

Connection Type
Network Access
Connection Settings
Summary

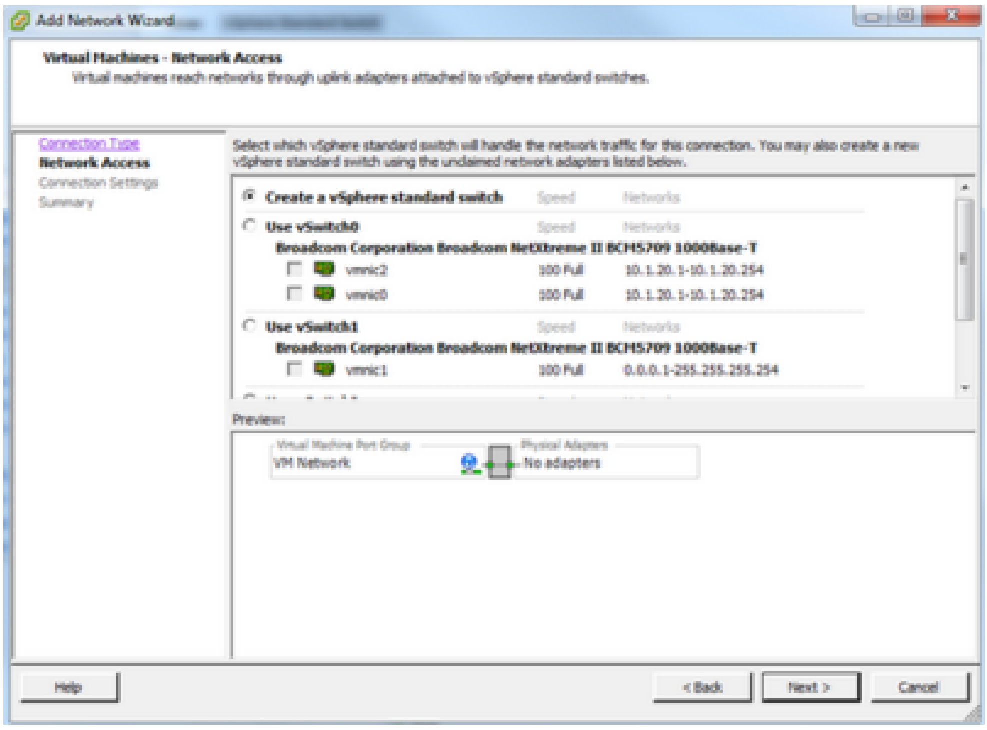
Connection Types

[✓ **Virtual Machine**]
Add a labeled network to handle virtual machine network traffic.

VMkernel
The VMkernel TCP/IP stack handles traffic for the following ESXi services: vSphere vMotion, iSCSI, NFS, and host management.

Help   < Back   Next >   Cancel

3. Select the onboard Ethernet adapter vmnic1 (for example) and click next.


Add Network Wizard

Virtual Machines - Network Access

Virtual machines reach networks through uplink adapters attached to vSphere standard switches.

Select which vSphere standard switch will handle the network traffic for this connection. You may also create a new vSphere standard switch using the unclaimed network adapters listed below.

	Speed	Networks
(●) **Create a vSphere standard switch**		
( ) **Use vSwitch0**		
Broadcom Corporation Broadcom NetXtreme II BCM5709 1000Base-T		
☐ vmnic2	100 Full	10.1.20.1-10.1.20.254
☐ vmnic0	100 Full	10.1.20.1-10.1.20.254
( ) **Use vSwitch1**		
Broadcom Corporation Broadcom NetXtreme II BCM5709 1000Base-T		
☐ vmnic1	100 Full	0.0.0.1-255.255.255.254

Preview:

Virtual Machine Port Group: VM Network
Physical Adapters: No adapters

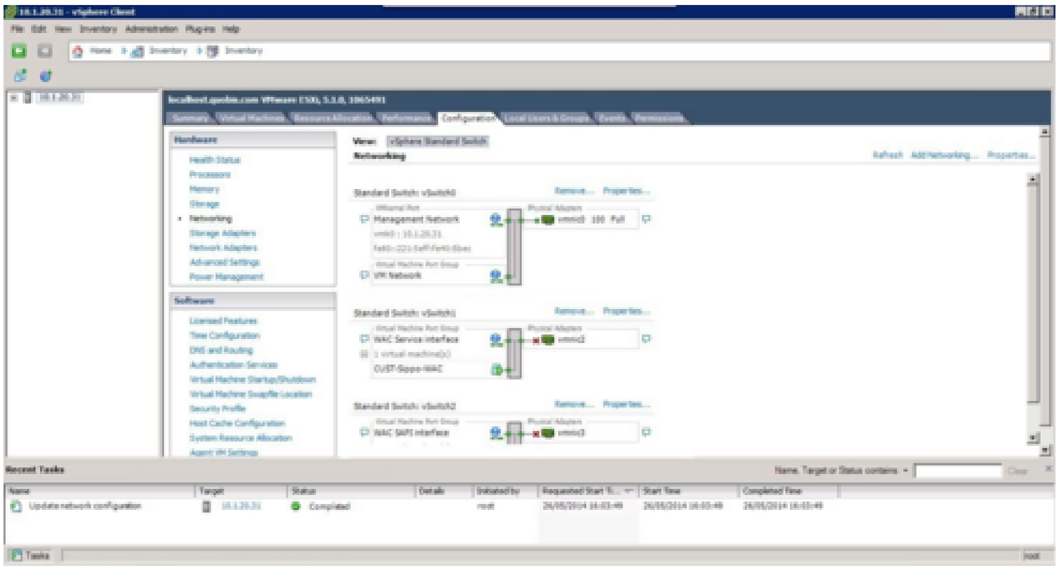
4. Change “Network” label to an appropriate name.

Note: Ethernet interfaces shown above should be named more generic, such as “net_212.145.x.x” etc. rather than “VM LAN 2”.

---

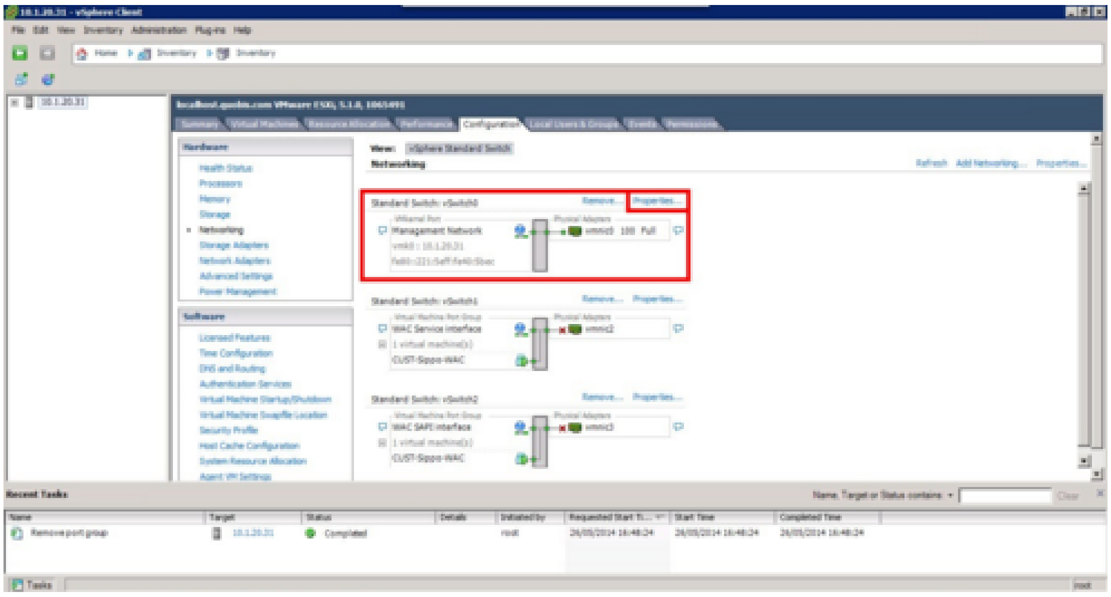
5. Click Next and then Finish.

1. Repeat previous steps to create another vSwitch (for Sippo AS REST API use). Assign in this case vmnic2 (for example) for dedicated interfaces.
2. The configuration will have three vSwitches mapped to the four physical adapters:
   1. One vKernel for ESXi management
   2. One vSwitch for Sippo AS/MS public interface
   3. One vSwitch for Sippo AS REST API


10.1.20.31 - vSphere Client
File Edit View Inventory Administration Plug-ins Help
Home Inventory Inventory
localhost.quobis.com VMware ESXi, 5.1.0, 1065491
Summary Virtual Machines Resource Allocation Performance Configuration Local Users & Groups Events Permissions
Hardware
Health Status
Processors
Memory
Storage
Networking
Storage Adapters
Network Adapters
Advanced Settings
Power Management
Software
Licensed Features
Time Configuration
DNS and Routing
Authentication Services
Virtual Machine Startup/Shutdown
Virtual Machine Swapfile Location
Security Profile
Host Cache Configuration
System Resource Allocation
Agent VM Settings
View: vSphere Standard Switch
Networking
Refresh Add Networking... Properties...
Standard Switch: vSwitch0 Remove... Properties...
Management Network
VM Network
Standard Switch: vSwitch1 Remove... Properties...
WAC Service interface
CUST-Sippo-WAC
vmnic2
Standard Switch: vSwitch2 Remove... Properties...
WAC SAPI interface
vmnic3
Recent Tasks
Update network configuration
Completed
Tasks

3. If you want to assign more physical NICs to one vSwitch, add or modify networks inside a vSwitch, add a VLAN tag or you want to make other changes on it. Click on the desired vSwitch Properties and follow the instructions on the screen.

   For example, we removed the VM Network created by default to keep the eth0 (vmnic0) as dedicated for the ESXi server.


10.1.20.31 - vSphere Client
File Edit View Inventory Administration Plug-ins Help
localhost.quobis.com VMware ESXi, 5.1.0, 1065491
Configuration
View: vSphere Standard Switch
Networking
Standard Switch: vSwitch0 Remove... Properties...
Management Network
Standard Switch: vSwitch1 Remove... Properties...
WAC Service interface
CUST-Sippo-WAC
Standard Switch: vSwitch2 Remove... Properties...
WAC SAPI interface
CUST-Sippo-WAC
Recent Tasks
Remove port group
Completed
Tasks

At this point, you have the VMware vSphere ESXi 5.x operational on the hardware and ready to deploy the Sippo OVA.

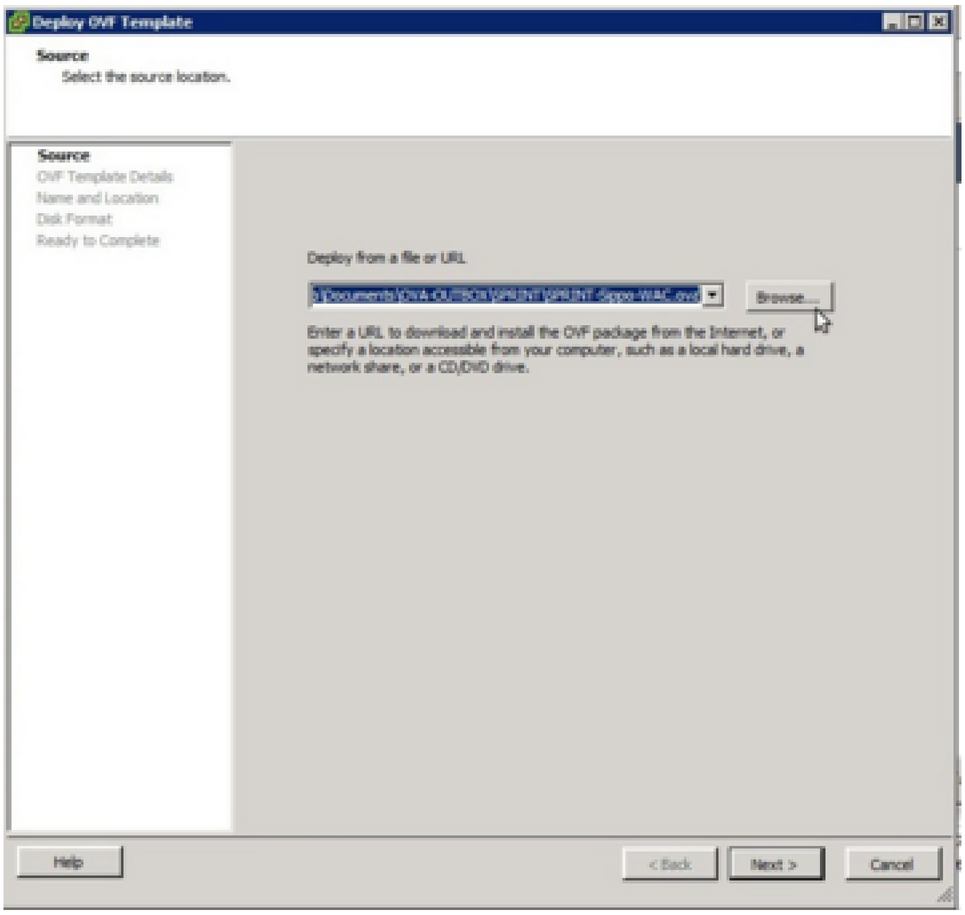
9.4 Installing Sippo OVA package on the ESXi Host

All Sippo Package VM images are deployed with OVA file extension. OVA installation instructions are as follows:

1. Select “File > Deploy OVF Template” on the top left corner.

   Note: SIPPO OVA files have a default configuration with 2 CPU cores, 2 GB RAM and 80 GB hard disk, 64-bit application, and 2 NIC Interfaces. Please ensure the ESXi host has sufficient capacity to host this configuration.

2. Select Source and Browse to the location of the .ova file and select next.



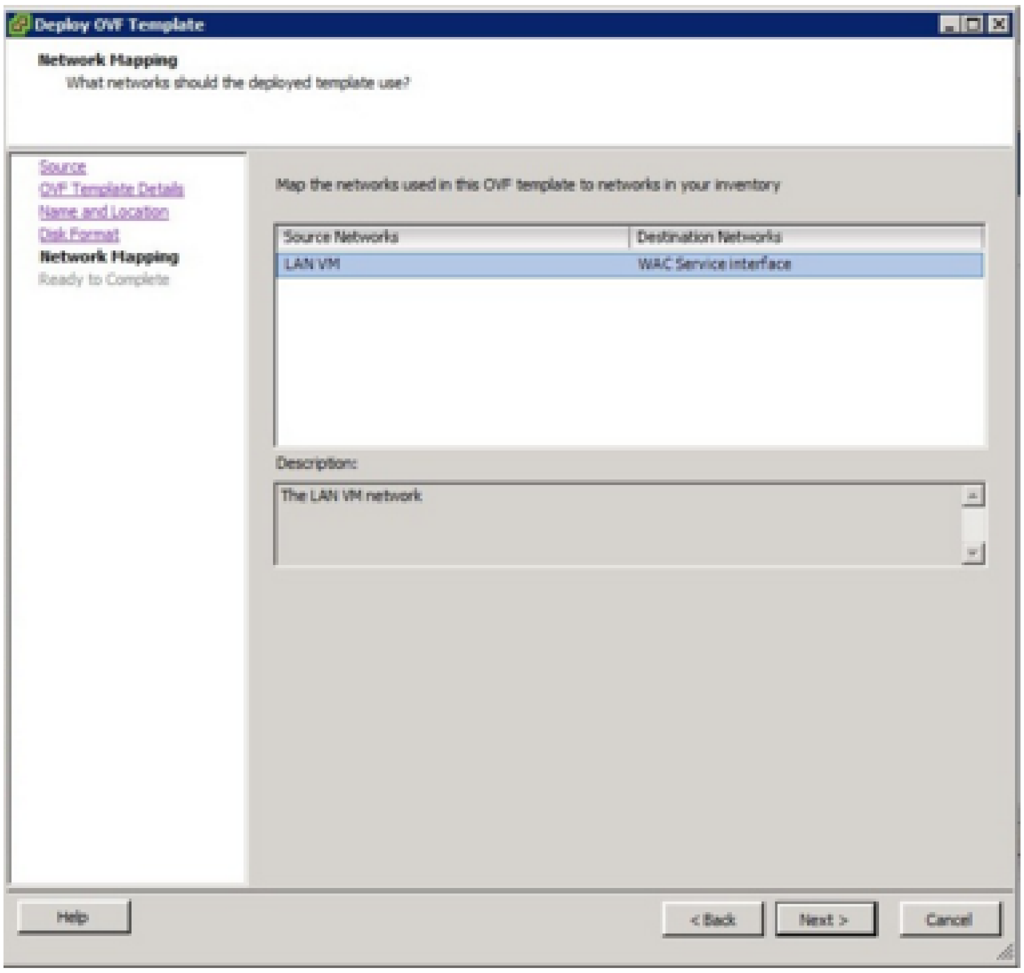
Source
Select the source location.

Deploy from a file or URL

[illegible]\Documents\OVA-OUTBOX\SPRINT\SPRINT-Sippo-WAC.ova

Enter a URL to download and install the OVF package from the Internet, or specify a location accessible from your computer, such as a local hard drive, a network share, or a CD/DVD drive.

3. Change the name of the VM to reflect the role of the VM. For example, “Sippo_PoC”
4. On the Disk format page, you can use different HDD storage options, click Thin or Thick if your custom OVA allows it. After that, click Next.
5. In the next dialog, map the networks on the VM template onto the networks of the host VM. Click on each network in turn, and select the appropriate vSwitch network from the drop-down. The example network mapping could be:
    1. “Network adapter 1” for PUBLIC_LAN
    2. “Network adapter 2” for MGMT_LAN
    3. “Network adapter 3” for SAPI_LAN



Map the networks used in this OVF template to networks in your inventory

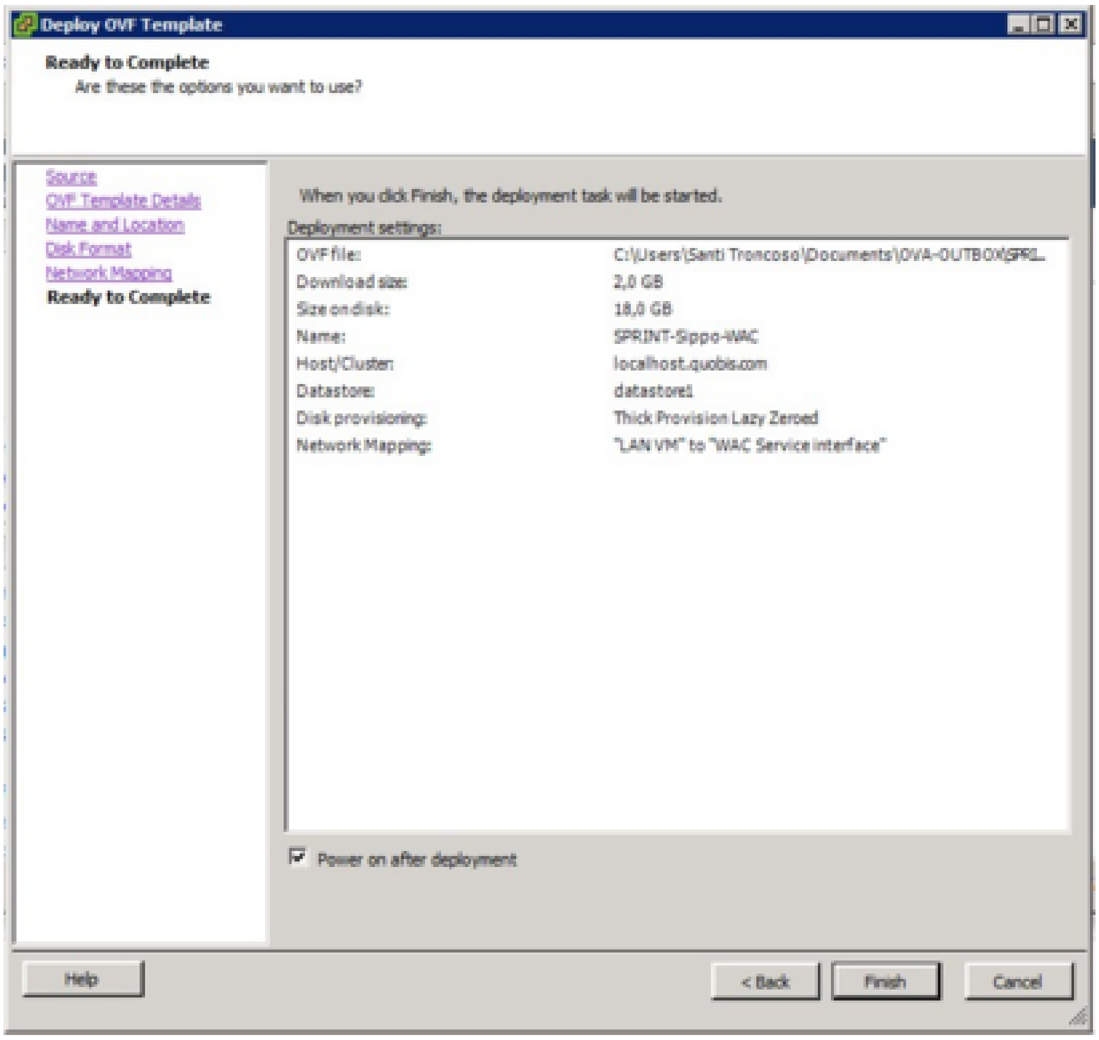
Source Networks	Destination Networks
LAN VM	WAC Service interface

Description:

The LAN VM network

Note: Please refer to Interface Mapping section when you need to map each of the two virtual network interfaces to a corresponding vSwitch instance, to ensure external access via the physical Ethernet ports of the ESXi host server. Choose a default network for any unused ports, based on their type (media/signaling vs management/control).

6. Confirm and apply the new VM configuration by clicking Finish. You can check the “Power on after deployment” for auto-launch your Sippo AS/MS Server after finish deploy.



When you click Finish, the deployment task will be started.

Deployment settings:

OVF file:	C:\Users\Santi Troncoso\Documents\OVA-OUTBOX\SPRI...
Download size:	2,0 GB
Size on disk:	18,0 GB
Name:	SPRINT-Sippo-WAC
Host/Cluster:	localhost.quobis.com
Datastore:	datastore1
Disk provisioning:	Thick Provision Lazy Zeroed
Network Mapping:	"LAN VM" to "WAC Service interface"

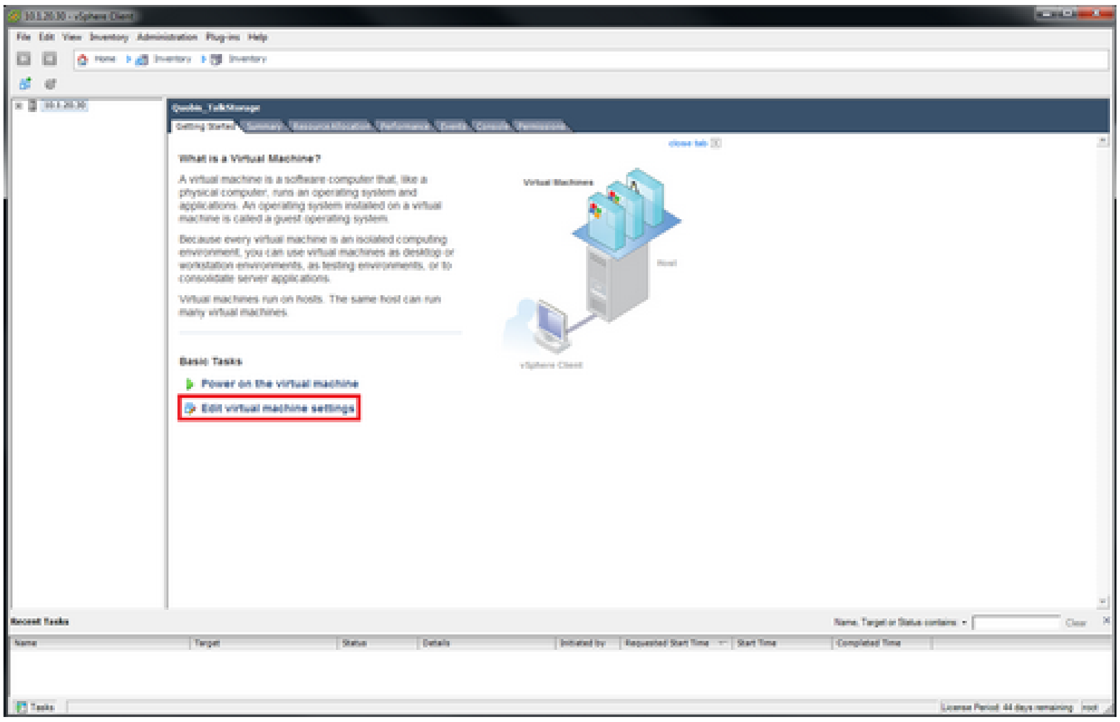
☑ Power on after deployment

7. The SIPPO OVA package is now deployed as a Guest on the VM Host and ready for use.

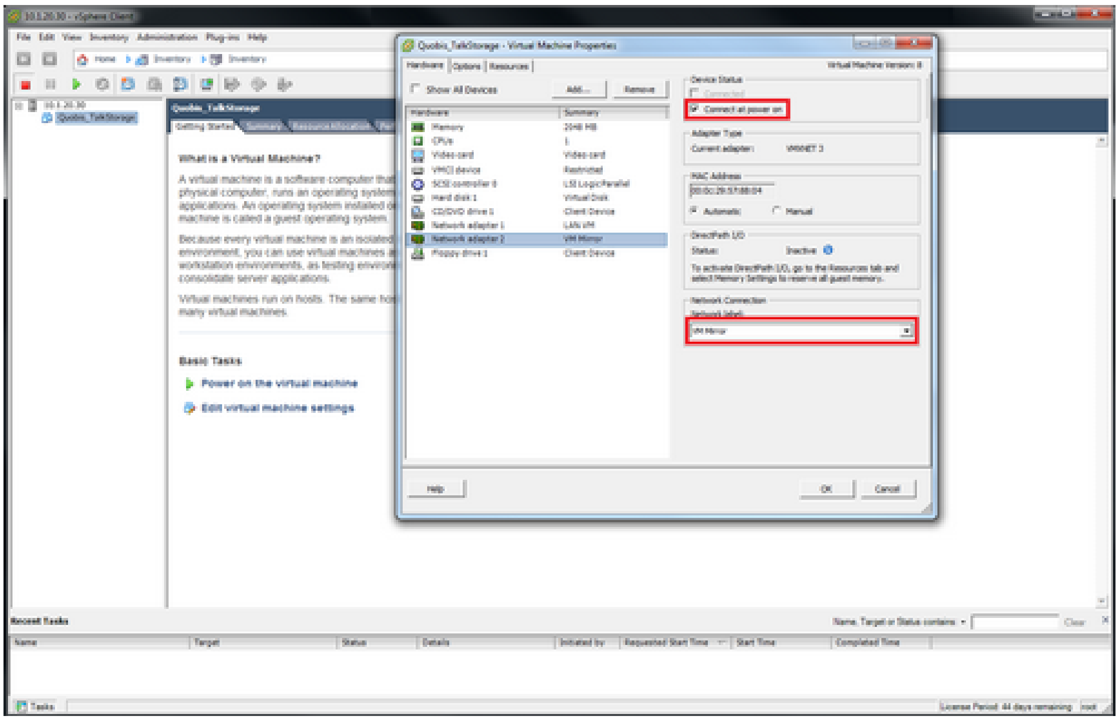
9.5 Configuring the VM on the ESXi Host

Configuring the VM instructions are as follows:

1. Expand the left panel, under the ESXi host should now show the new VM.
2. Click “Edit virtual machine settings” on the new VM.


What is a Virtual Machine?
Basic Tasks
Power on the virtual machine
Edit virtual machine settings

3. Enable “Connect at power on” located on the upper right corner on both Network adapters. And check that “Network Adapter 1” will use LAN VM and “Network adapter 2” will use VM Mirror “Network Connection” (in this case).



CHAPTER

TEN

APPENDIX B. SIPPO INTERNAL UML DIAGRAMS

Contents

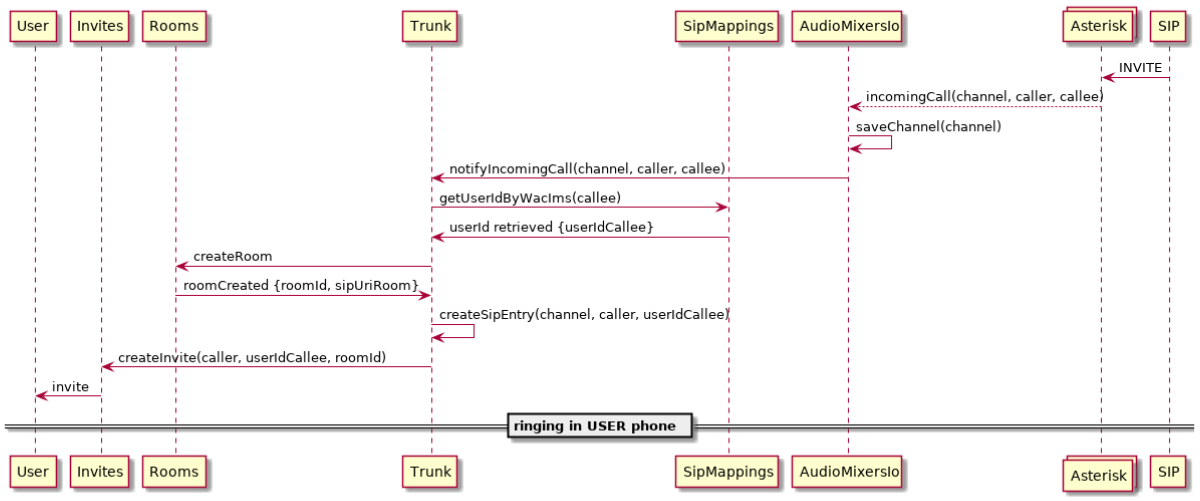
10.1 QSS

Included here all available UML diagrams defined for **QSS**.

10.1.1 Interaction between Trunk and AudioMixersio services

10.1.1.1 Calls from SIP to user

Creation of the call

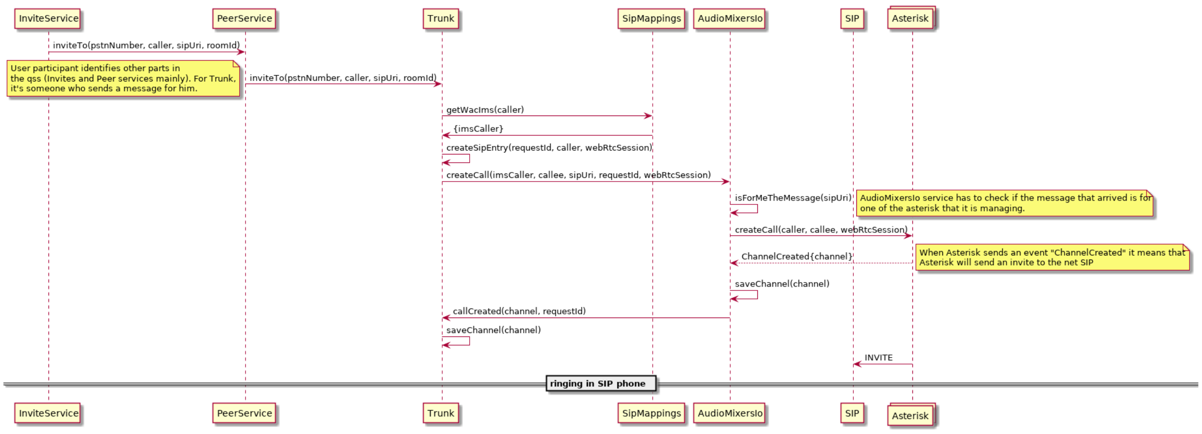

User
Invites
Rooms
Trunk
SipMappings
AudioMixersIo
Asterisk
SIP
INVITE
incomingCall(channel, caller, callee)
saveChannel(channel)
notifyIncomingCall(channel, caller, callee)
getUserIdByWacIms(callee)
userId retrieved {userIdCallee}
createRoom
roomCreated {roomId, sipUriRoom}
createSipEntry(channel, caller, userIdCallee)
createInvite(caller, userIdCallee, roomId)
invite
ringing in USER phone

From this scenario the following cases can occur:

- SIP phone cancels the invitation
- User accepts the invitation
- User rejects the invitation

10.1.1.2 Calls from user to SIP

Creation of the call


InviteService
PeerService
Trunk
SipMappings
AudioMixersIo
SIP
Asterisk
inviteTo(pstnNumber, caller, sipUri, roomId)
inviteTo(pstnNumber, caller, sipUri, roomId)
getWacIms(caller)
{imsCaller}
createSipEntry(requestId, caller, webRtcSession)
createCall(imsCaller, callee, sipUri, requestId, webRtcSession)
isForMeTheMessage(sipUri)
createCall(caller, callee, webRtcSession)
ChannelCreated{channel}
saveChannel(channel)
callCreated(channel, requestId)
saveChannel(channel)
INVITE
ringing in SIP phone

User participant identifies other parts in the qss (Invites and Peer services mainly). For Trunk, it's someone who sends a message for him.

AudioMixersIo service has to check if the message that arrived is for one of the asterisk that it is managing.

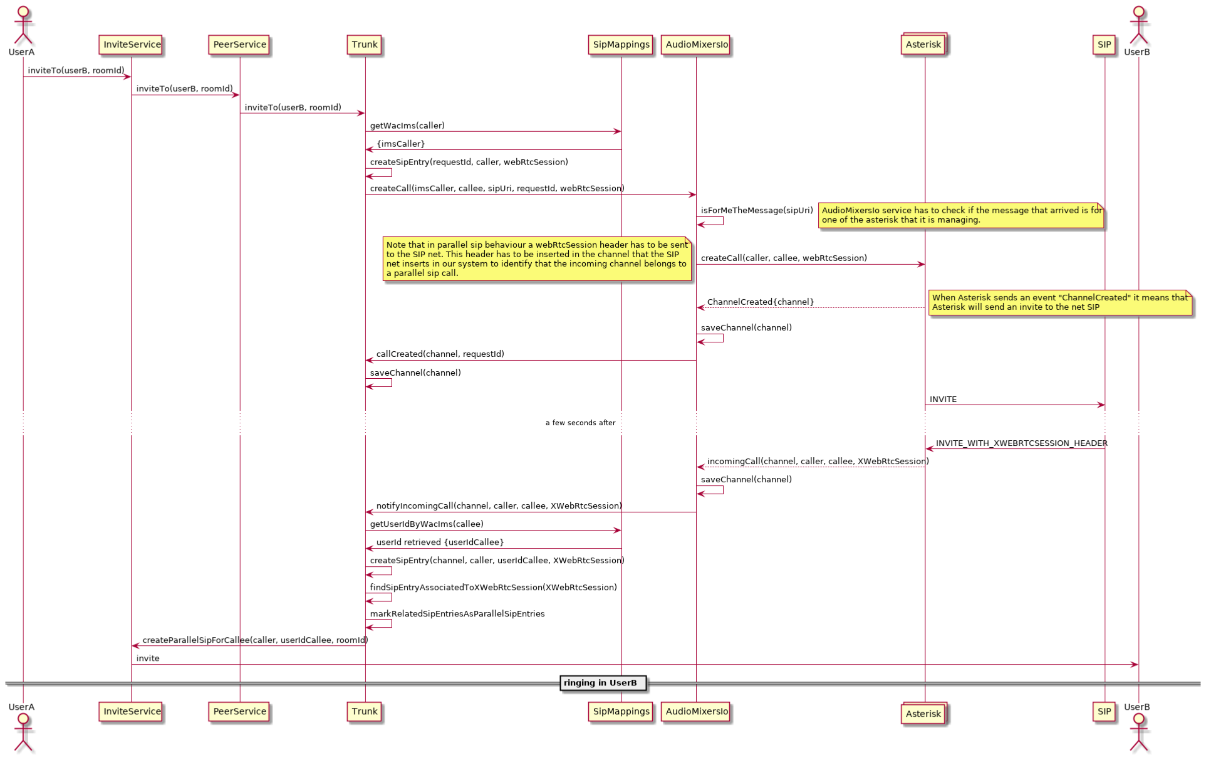
When Asterisk sends an event "ChannelCreated" it means that Asterisk will send an invite to the net SIP

From this scenario the following cases can happen:

- SIP accepts the invitation
- SIP rejects the invitation
- User cancels the invitation
- User leaves a established call
- SIP leaves a established call

10.1.1.3 Parallel SIP

Creation of the call


UserA
InviteService
PeerService
Trunk
SipMappings
AudioMixersIo
Asterisk
SIP
UserB
inviteTo(userB, roomId)
inviteTo(userB, roomId)
inviteTo(userB, roomId)
getWacIms(caller)
{imsCaller}
createSipEntry(requestId, caller, webRtcSession)
createCall(imsCaller, callee, sipUri, requestId, webRtcSession)
isForMeTheMessage(sipUri)
AudioMixersIo service has to check if the message that arrived is for one of the asterisk that it is managing.
Note that in parallel sip behaviour a webRtcSession header has to be sent to the SIP net. This header has to be inserted in the channel that the SIP net inserts in our system to identify that the incoming channel belongs to a parallel sip call.
createCall(caller, callee, webRtcSession)
ChannelCreated{channel}
When Asterisk sends an event "ChannelCreated" it means that Asterisk will send an invite to the net SIP
saveChannel(channel)
callCreated(channel, requestId)
saveChannel(channel)
INVITE
a few seconds after
INVITE_WITH_XWEBRTCSESSION_HEADER
incomingCall(channel, caller, callee, XWebRtcSession)
saveChannel(channel)
notifyIncomingCall(channel, caller, callee, XWebRtcSession)
getUserIdByWacIms(callee)
userId retrieved {userIdCallee}
createSipEntry(channel, caller, userIdCallee, XWebRtcSession)
findSipEntryAssociatedToXWebRtcSession(XWebRtcSession)
markRelatedSipEntriesAsParallelSipEntries
createParallelSipForCallee(caller, userIdCallee, roomId)
invite
ringing in UserB

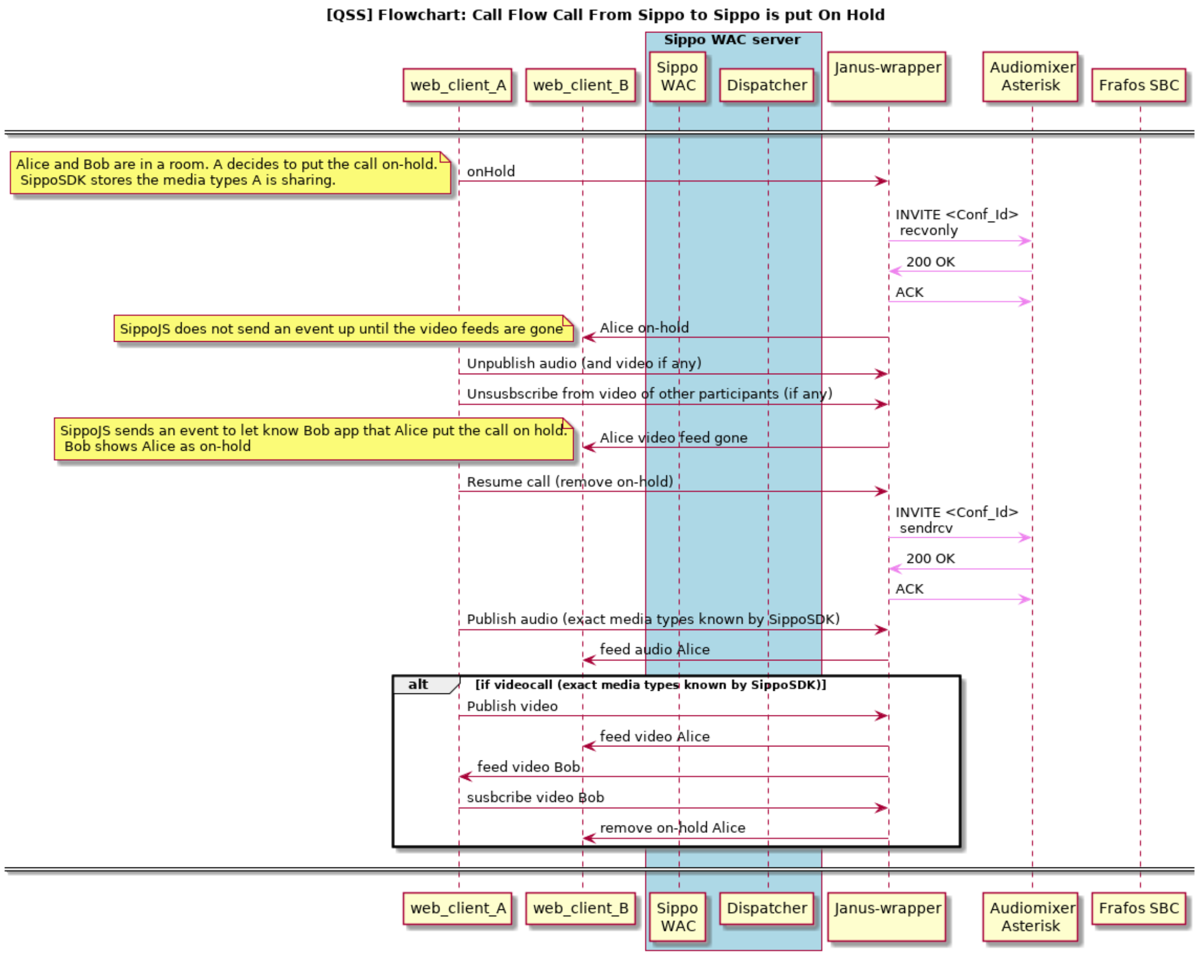
From this scenario the following cases can take place:

- Callee accepts the invitation
- Callee rejects the invitation
- Caller cancels the invitation
- User leaves the established call

10.1.2 Calls

Included here all available UML diagrams defined for **calls** behavior on Sippo solutions

Call is put on hold


[QSS] Flowchart: Call Flow Call From Sippo to Sippo is put On Hold
Sippo WAC server
web_client_A
web_client_B
Sippo WAC
Dispatcher
Janus-wrapper
Audiomixer Asterisk
Frafos SBC
Alice and Bob are in a room. A decides to put the call on-hold. SippoSDK stores the media types A is sharing.
onHold
INVITE <Conf_Id> recvonly
200 OK
ACK
SippoJS does not send an event up until the video feeds are gone
Alice on-hold
Unpublish audio (and video if any)
Unsusbscribe from video of other participants (if any)
SippoJS sends an event to let know Bob app that Alice put the call on hold. Bob shows Alice as on-hold
Alice video feed gone
Resume call (remove on-hold)
INVITE <Conf_Id> sendrcv
200 OK
ACK
Publish audio (exact media types known by SippoSDK)
feed audio Alice
alt
[if videocall (exact media types known by SippoSDK)]
Publish video
feed video Alice
feed video Bob
susbcribe video Bob
remove on-hold Alice

10.1.2.1 Parallel SIP

Included here all available UML diagrams defined for **parallel-sip** feature

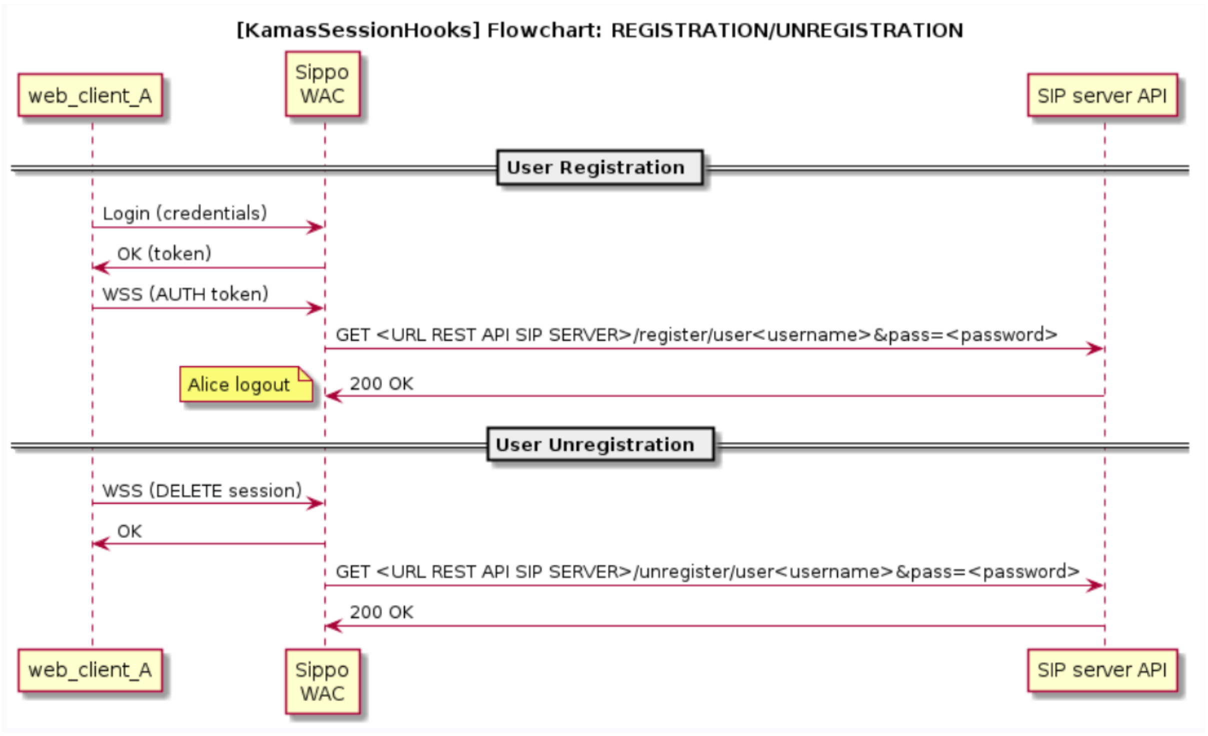
Here we have two scenarios:

- Outgoing call from Sippo to SIP
- Outgoing call from Sippo to Sippo

10.2 Sippo Server

Included here all available UML diagrams defined for **Sippo server**.

KamasSessionHooks


[KamasSessionHooks] Flowchart: REGISTRATION/UNREGISTRATION
web_client_A
Sippo WAC
SIP server API
User Registration
Login (credentials)
OK (token)
WSS (AUTH token)
GET <URL REST API SIP SERVER>/register/user<username>&pass=<password>
Alice logout
200 OK
User Unregistration
WSS (DELETE session)
OK
GET <URL REST API SIP SERVER>/unregister/user<username>&pass=<password>
200 OK
web_client_A
Sippo WAC
SIP server API

CHAPTER

ELEVEN

GLOSSARY

Agent Usually this name is used to identify the human user for a RTC application. Generally identified with a contact center customer engagement role.

audiomixer (amx) SIP back to back user agent used for SIP trunking interconnection. It is also mixes the audio of the conferences and calls the SIP trunking. It enables the adaptation to the particular needs of the SIP trunk. Currenlty implemented using Sangoma Asterisk

browser-stack SDK provided by the community or the browser vendors handle the access to media devices and the WebRTC API. It provides the functionalities required to obtain the media that is transferred to the other side/party in order to achieve a real time communication. It is necessary to make sure that the application will work in different browsers and platforms.

Customer Usually this name is used to identify the human user for a RTC application. Generally identified as the party that receives or consumes products or services.

Database (db) A database is an organized collection of long-lasting data used by the different service to store the information which must be shared between service instances and be persistent across restarts.

Environment Computer system in which a computer program or software component is deployed and executed.

Hardware Collection of physical elements that constitutes a computer system.

HTML5 application The business logic application, the one that covers the interface between the user and the functionalities and cover the part that is not technologically related. It is the core of the *use case*.

Identity provider Third party role that grants and assure the valid identity of an entity. Usually refer to this on certification and authentication procedures.

Login User access data to a specific feature or application.

message-broker (mb) Messaging broker that handles communication of messages and events between independent internal Sippo services. Currently implemented using RabbitMQ

Network Collection of terminal nodes and connected links which allows computers to exchange data.

OAuth2 Standard authenitication protocol for double side proof of authentication.

oauth2-proxy (oa2p) Service developed by Quobis to handle access delegation based on open standard OAuth2. It enables the authentication of SippoSDK applications and also plays the role of identifying the authentication provider based on its configuration. It also allows the Sippo solutions to handle non-standard OAuth2 access delegation systems by the implementation of ad-hoc custom modifications.

rec-post-proc (rpp) Quobis development used to generate media recordings from the media captured in the sfu and audiomixer.

Register Infrastructure element that stores dynamic data about an endpoint and allows to identify and locate the network element and its proper unique ID.

REST API REpresentational State Transfer Application Programming Interface. Set of protocols, routines and tools to coordinate two services or pieces of software.

Reverse-proxy (rp) A type of proxy server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client, appearing as if they originated from the proxy server itself.

RTC Abbreviation of Real Time Communications

SAPI Service API –> Third party applications interface. This interface is used to communicate the Sippo WAC with third party services or applications. For example, when federating the authentication via a OAuth2 connection.

Service provider Third party or outsourced suppliers.

sfu Network element to handle WebRTC media streams. It receives streams from each conference participant and forwards them to the rest of participants. It does not do media processing: it does not change neither the codec nor the bit rate and resolution of the video flows. It is able to play the WebRTC gateway role providing media interworking: WebRTC to SIP media profiles, DTLS-SRTP to RTP and timestamps modifications. When used with codecs which support Scalable Video Codec (SVC) it enables the delivery of flows different bit rates and resolutions to different endpoints based on defined rules.

sfu-wrapper Linked directly to the sfu, the wrapper grants connection between the sfu unit and the sfu-dispatcher to handle redundancy and scaling. It also receives direct connections from the applications in order to join to rooms and negotiate the characteristics of the media flows published and received.

Sippo Family of Quobis products dedicated to the RTC applications. Including the Sippo WAC, Sippo Collaborator and others.

Sippo application server (Sippo AS) Entity that manages and implements all the application logic. The Sippo AS serves the WebRTC applications, to achieve some of its features the Sippo AS is integrated with third party elements to cover the signaling part of the WebRTC communication, and for other context actions, like authentication, recordings, storage and similar.

Sippo Connector WAC service that uses a third party service (hosted at the same machine or at an external one) in order to provide a specific feature. Example: Google connectors provide *Google Contacts* and others.

Sippo ecco Telco-class ready-to-use OTT client fully integrable with IMS, VoLTE, VoWifi and MVNO architectures and AAA elements.

Sippo hub Enterprise-grade server that enables companies with customer-care applications based on multimedia real time communications to defeat the challenges of digital transformation while keeping the existing call center infrastructure.

Sippo wac Telco-class real-time communication application server for the exposure of APIs. It provides tools for developers to create real-time communication apps while managing all the complexity related to device-fragmentation, AAA interconnection, service enablement and troubleshooting and user management

sippo-server (ss) Quobis development that handles all the application logic and orchestration of different internal services of the Sippo solutions.

SippoSDK The SippoSDK acts as an application abstract layer to interact with the different services of the Sippo AS and provides the functionality to the HTML5 application. It provides the ability to implement the same application with different WebRTC gateway vendors. This avoids a vendor lock-in and provides extra functionalities in order to have freedom to choose the better technical solution for each use case.

Software Part of a computer system that consists of encoded information or computer instructions. On the present document we will refer to the developed solution to cover the customer requirements.

turn-server (turn) Backend support for TURN protocol. This element implements connectivity protocols used to achieve real-time connectivity with endpoints which are behind NAT and restrictive firewalls. Currenlty implemented using coturn

User Person using a generic system of a commercial product or service.

vendor-stack Javascript, Android and iOS SDK provided by the vendor of WebRTC gateway. This element is typically not open-sourced and implements proprietary protocols. It enables the interaction between Sippo SDK and the WebRTC infrastructure.

Videoconference Communication between two locations by simultaneous two-way video and audio transmissions.

WAC Abbreviation of WebRTC Application Controller

WAPI WAC API –> Sippo applications interface. It is the standard way to communicate a Sippo application with the Sippo AS (application server).

web-server (ws) The primary function of a web server is to store, process and serve web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML, which normally include images, CSS style sheets and Javascript files in addition to the text content. Currently implemented using NGINX

WebRTC Web Real-Time Communication) is an API definition that supports browser-to-browser applications for voice calling, video chat, and P2P file sharing without the need of either internal or external plugins.

WebRTC Application Controller Network element designed to expand the *Application Server* definition granting all required features for WebRTC applications.

xmpp-server (xmpp) Messaging XMPP server to handle asynchronous message exchange between SippoSDK clients. Currently implemented using Prosody

HTTP ROUTING TABLE

INDEX