SPITCH

Spitch Deployment Guide

Enterprise Solutions

Driven by Voice

Publication Date: 22/12/2016
Document Name: Spitch Deployment Guide
Document Version: 1.0

Published by: Spitch Solutions AG

Table of Contents

1 Before You Start

1.1 Introduction

This Deployment guide describes Spitch Voice Recognition and Speaker Verification solutions and shows the solutions architecture and explains how to install it in your organization.

From now on, voice recognition is referred as ASR (automatic speech recognition) and Speaker Verification is SV (speaker verification).

1.2 Audience

This Deployment guide is intended for Spitch Administrators and developers who use the Spitch solution. The document starts with an overview of Spitch and speech technology and continues to describe how to install the solution.

1.3 Prerequisites

You need to be familiar with the following technologies:

- HTTP protocol
- Linux OS
- System administration
- Command shell
- Scripting languages

1.4 Objectives

After you have read this guide, you will be able to:

- Understand how Spitch technology works in conjunction with your organization.
- Understand the Spitch environment and architecture.
- Install Spitch CodyFi and SignyFi
- Install Spitch VeryFi.
- Access speech recognition HTTP interface.
- Access Speaker VerificationHTTP interface.
- Access speech recognition MRCP interface.
- Access the storage REST API.
- Configure Spitch CodyFi and Spitch VeryFi.

2 Introduction to the Spitch Solutions

2.1 About Speech Recognition and Speaker Identification Technologies

Speech Recognition and Speaker Verification are increasingly embedded into every day's technology providing the ability to send voice messages or to perform voice search.

Both Speech Recognition and Speaker Verification use complex algorithms, and adopt a multi-disciplinary approach to combine the findings in digital signal processing, computational linguistics, computer science and artificial intelligence.

The multi-disciplinary approach supports a variety of models to obtain speech recognition, speaker verification and identification, topic discovery and sentiment analysis.

Approach to speech and voice technology is based on Deep Learning (DL).

Speech Recognition implies the ability of a system to understand words and sentences in a given audio segment, and transcribe them into a readable format.

Speaker Verification is the ability of a a system to ascertain the caller's identity based on the voice-print.

The caller's voice is checked against a set of hidden biometric parameters, which constitute the caller's voice-print.

The caller's voice-print must be previously stored into the system, through a process known as enrolment.

Typical applications are for fraudsters' detection, for example, within the banking services, and voice authentication.

2.2 Spitch Solutions

Spitch solutions exploit speech and voice technology to optimize knowledge work automation.

Spitch solutions are based on Automatic Speech Recognition (ASR), Speaker Verification (SV), Voice User Interface (VUI) and Speech analytics.

Spitch technologies allow you to extract, analyse and store information from unstructured voice data, and turn the information into meaningful, timely knowledge.

The Spitch suite consists of three solutions: Spitch CodyFi, Spitch VeryFi and Spitch SignyFi.

In addition, Spitch Lingware Development Portal provides customers and partners with a set of tools to develop their own speech solutions.

2.2.1 What Is the Spitch Lingware?

Spitch Lingware is a linguistic built-in resource that allows the Speech engine to carry out voice recognition.

Lingware consists of a lexicon, a language model and an acoustic model.

The Lingware is deployed to the Speech engine (back-end) to provide speech recognition.

The Lingware resource is language, domain and customer-dependent.

Typically, multiple Lingware resources are uploaded onto the recognition speech engine.

2.2.2 What Is CodyFi?

CodyFi is the ASR Platform. It guarantees highly accurate speech-to-text conversion in real time for Interactive Voice Response (IVR), Speech analytics, Quality Assurance (QA) monitoring, and other applications.

CodyFi can be based on Large Vocabulary Continuous Speech Recognition (LVCSR) or on closed grammars or on keyword spotting.

CodyFi supports a growing list of languages, which is continuously expanding, and allows organizations to add new languages to their current installation.

2.2.3 What Is VeryFi?

VeryFi is the Voice Biometrics Platform and enables Speaker Verificationand speaker identification, fraudsters detection, voice signature, and authentication.

Voice verification can run in parallel with speech recognition.

VeryFi can ascertain the caller's identity using voice-prints. A voice-print is a vector of hidden biometrics parameters specific to a person.

To perform SV, voice-prints are stored in the system. The process of storing voice-prints is known as enrolment.

The system automatically compares the caller's voice characteristics to the caller's voice-print.

VeryFi is able to perform speaker recognition in real-time.

Continuous voice-print updates reduce the false rejection rate.

The VeryFi platform takes into account phonetic structure of a particular language, which allows us to fine tune models for different languages and dialects.

2.2.4 What Is SignyFi?

SignyFi is a Semantic Interpretation Solution, and it supports topic discovery and sentiment analysis.

SignyFi uses CodyFi results to create semantic tags for categorization.

Each time that SignyFi analyses a conversation, it assigns semantic interpretations to content and resolves natural language ambiguity.

2.2.5 The Spitch Lingware Development Portal

The Spitch Lingware Development Portal provides you with access to a complete set of tools to customize your speech solution.

These unique cloud-based tools support private client data management, and crowd-sourced development process.

The Application Builder allows you to design expert grammars and to create customized vocabularies and language models for ASR.

Note

The installation of the Lingware and the use of the Lingware portal is not within the scope of this document.

2.2.6 Real-Life Applications

Spitch software is at the heart of many industries that seek to have a competitive edge in terms of both performance and security.

Spitch software applications are used, but not limited to :

- Banking and insurance industry
- Contact centers
- Telecom industry
- Medicine and Healthcare sector
- Government and Public sector

Spitch software applications include solutions for:

- IVR for Call Steering
- Automated Information Capturing
- Automated form filling
- Passive Voice Verification

2.2.7 IVR for Call Steering

Voice-driven IVR provides you with the ability to produce first call resolution and steer customer calls to appropriate agents in automated mode.

IVR tackles growing costs and customer dissatisfaction by delivering enhanced customer experience, costs reduction and increased agent efficiency.

The average call duration is reduced without penalizing the quality and accuracy of the exchange.

High accuracy and improved issue resolution rates are achieved through the seamless integration of both state-of-the-art speech recognition and semantic interpretation services, which facilitate automation of standard business processes.

2.2.7.1 Automatic Information Capturing

Spitch solution supports automated information capturing.

Automatic information capturing is used in a variety of settings, for example, to automatically process credit card payments over the phone.

This application can be used in conjunction with big data analytics to provide a more detailed understanding of customer needs, preferences and market organizations.

Automatic Information capturing assists your organization in reducing the complaints regarding payments, thus increasing your NPS, whilst reducing costs since PCI DSS compliance is no longer required for each agent.

2.2.7.2 Automatic Form Filling

Often it is necessary to extract and commit important customer information during the course of a call.

This is traditionally performed manually by the call centre agent during the customer conversation, or right after the call is finished. The process can be improved as well as sped up through the use of Spitch solutions.

Automatic form filling finds its application in the automation of repeated procedures, in many sector, including the Healthcare.

2.2.7.3 Passive Voice Verification

Privacy and security are central to many operations and transactions, and ultimately to determine a business reliability.

Banks, insurance companies and telecoms are required to verify customer identities for predefined customer calls. Traditional verification involves asking security questions such as the amount of the last transaction or a security question, for example, the mother's maiden name.

This is time consuming and often frustrating for customers who, sometimes, cannot recall their last transaction or the answer to the security question.

Spitch VeryFi solution provides this additional level of security applying text-independent Speaker Verification in a natural conversation.

3 The Spitch Solution Resources

Spitch uses a variety of technologies to carry out automatic voice recognition (ASR) and Speaker Verification(SV).

The speech engine is at the heart of the Spitch solution, and it is also the mechanism, through which speech recognition and Speaker Verificationare performed.

In addition to the speech engine, the following industry-standard resources are required:

- MRCP server

  MRCP server is never used with VeryFi.

  The Spitch MRCP server acts as a gateway that accepts MRCP/RTP requests and routes them over HTTP to the Spitch ASR engines.

  AND/ OR

  HTTPS Proxy

  HTTPS Proxy is required for VeryFi, and optional for CodyFi.

  The HTTPS Proxy server provides a way to send audio data to the speech ASR/SV engines over a secure TLS connection.

  HTTPS Proxy supports SSL encryption and TLS up to TLS 1.2.

- Storage sub-system (optional)

  The storage is mandatory to store the voice-prints used by VeryFi.

  CodyFi does not require mandatory use of storage.

- Monitoring sub-system

Each cluster comprises a number of mandatory components and optional components as described in the Table 1: The Speech Cluster and Storage Cluster Components

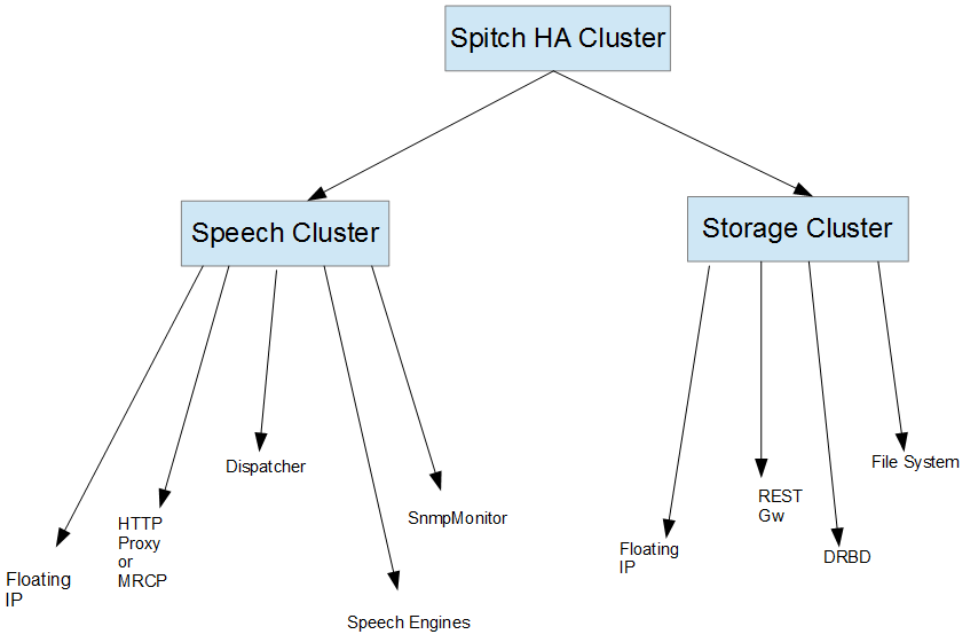

Illustration 1 The Spitch Resources at glance

3.1 What Is the CodyFi Speech Recognition Engine?

The Spitch ASR Engine is the core of the Spitch CodyFI and SignyFi system.

The Spitch engine receives ASR chunked HTTP POST requests either from the HTTP-Proxy or the MRCP server, and it decodes in real-time the audio chunks that are in the request body.

When the decoding is completed, the ASR engine sends an XML response back to the client.

The XML response includes the recognition results (CodyFi), and optionally, it can provide the semantic categories that are extracted from the result using a SSM – Statistical Semantic Model. (SignyFi).

For more information on how to access the CodyFi interface, see Speech Recognition HTTP Interface – CodyFi.

3.2 What Is the VeryFi Speaker Verification Engine?

The Spitch SV Engine is the core of the Spitch VeryFi system.

The Spitch SV engine receives HTTP POST requests from the HTTPS-Proxy to:

- Enroll (create) a voice-print.
- Update a voice-print.
- Verify a speech segment against a given voice-print.
- Delete voice-prints.

The speech engine sends an XML response back to the client with the results or the verification score.

For more information on how to access the VeryFi interface, see Speaker Verification HTTP Interface – VeryFi

3.3 The Dispatcher

The Dispatcher main role is to assign a MRCP or a HTTPS Proxy request to a speech engine resource that supports the speech request type and that is available to process the request.

The Proxy server queries server availability on the cluster, and the Dispatcher checks the next available server, and communicates it to the Proxy.

The algorithm to load-balance among a set of equivalent Speech Engines is the Round Robin.

The Dispatcher is automatically notified when new Speech Engines start,stop or if they fail, so that routing is dynamically adjusted during Speech Engines fail-over or when new engines are added or remove.

3.4 The MRCP Server

The Spitch MRCP server acts as a gateway that accepts MRCP/RTP requests and routes them over HTTP to the Spitch ASR back-end.

The MRCP contacts the Dispatcher in the cluster and maps a given grammar to a suitable speech engine (ASR) as shown in the MRCP Recognition Scenario diagram.

Upon selection of the speech engine, the MRCP converts incoming RTP traffic to HTTP chunked body and forwards it to the ASR speech engine.

At the end of the recognition process, the client receives an XML response.

For more information on how to access the CodyFi HTTP interface, see Speech Recognition MRCP Interface – CodyFi

MRCP server running on one vCPU can handle up to 100 unencrypted connections. If more connections are needed, several MRCP servers must be installed, in different clusters.

The MRCP server integrates with PBXs telephone systems for a number of IVR (Interactive Voice Response) and speech analytics applications:

- Freeswitch
- Avaya
- Genesys

  Additional IVRs can be supported on request.

A complete description of the MRCPv2 protocol can be found in the RFC standard 6787 (https://tools.ietf.org/html/rfc6787).

The MRCP server is based on the opensource UniMRCP stack, specifically the unimrcpserver .

The MRCP server is only required in case, the MRCP protocol is used.

3.4.1 The Voice Activity Detection Process with MRCP

The MRCP server performs voice activity detection.

The client sends the utterance to the MRCP server, which is responsible to detect voice activity within the utterance.

The MRCP forwards to the ASR speech engine only the portion of signal where voice activity was detected.

To prevent utterance truncation, a configurable number of milliseconds of the audio signal is pre-buffered and sent to the speech engine after voice activity is detected.

If no voice activity is detected, no data is forwarded to the ASR speech engine.

To view all the timeout values that control voice activity detection, see Speech Recognition MRCP Interface – CodyFiand Supported MRCPv2 Message Bodies.

3.4.2 The MRCP Server Workflow

The MRCP server workflow is described in Illustration 2 The MRCP Server Workflow.

The MRCP server tries to detect user activity without timing out during inactivity.

This is described in the Barge-in Mode section and the Waveform-URI section.

The MRCP server records audio data for all the recognised requests and uses start-input-timers to define whether it is in barge-in mode and non-barge-in mode.

If the input timers are set to true, the inactivity timer starts operating in non-barge-in mode.

If the input timers are set to false, the server tries to detect user activity without timing out during inactivity.

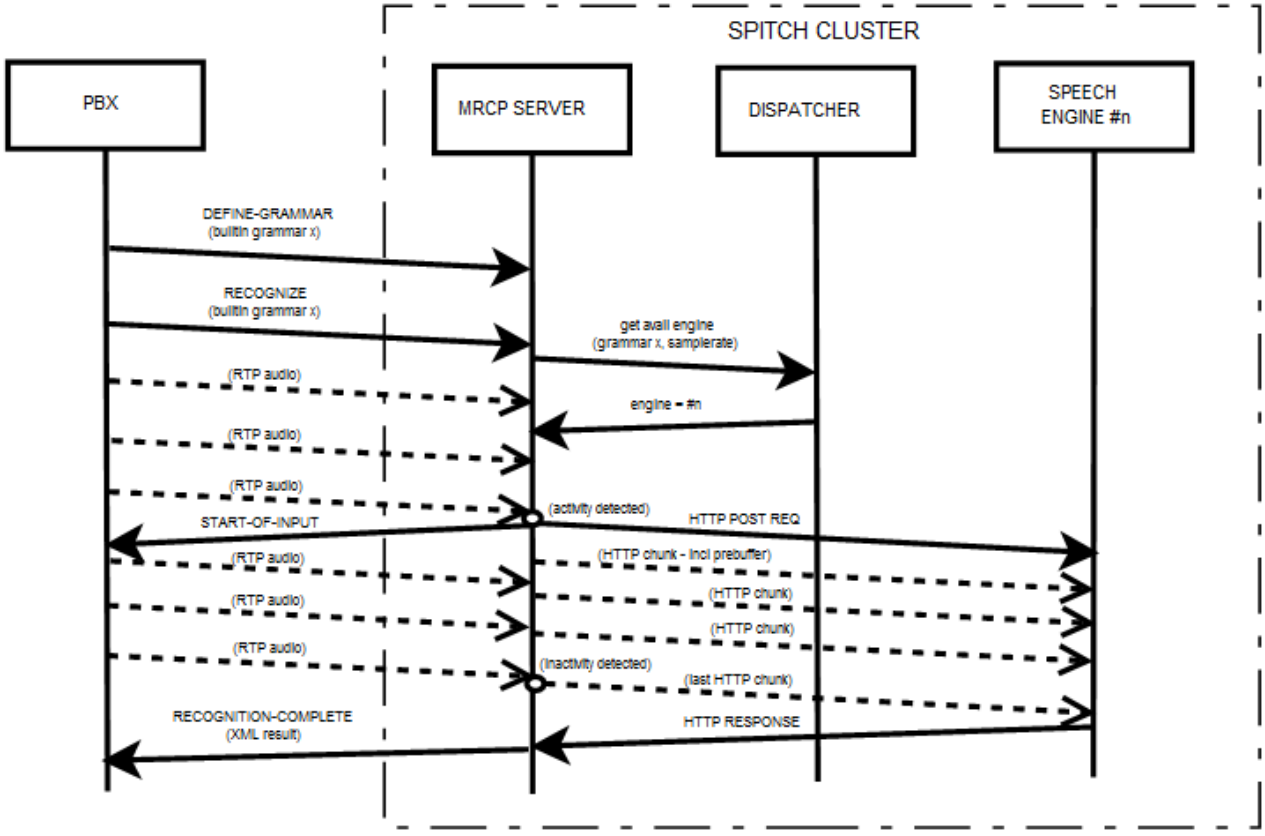

Illustration 2 The MRCP Server Workflow

3.4.2.1 Barge-in Mode

With regard to voice activity detection, the MRCP server supports barge-in and non barge-in modes.

- Barge-in mode

  The PBX sends a RECOGNIZE request with the **start-input-timers** header set to “false”, and plays back a prompt to the caller. The server then tries to detect user activity but it will not time out on inactivity.

  When the prompt ends, the PBX signals to the MRCP server with a **START-INPUT-TIMERS** message, to instruct the MRCP to start counting inactivity.

- Non barge-in mode

  The PBX sends the RECOGNIZE after the prompt has finished.

  The RECOGNIZE message contains the header start-input-timers set to “true”, so that the inactivity timer starts operating immediately.

In both modes, the MRCP server signals to the PBX with a **START-OF-INPUT** message after activity is detected.

In the case of barge-in mode, this message typically causes the PBX to abort the prompt.

3.4.2.2 Waveform-URI

The MRCP server records the audio data received in the RTP payloads for all RECOGNIZE requests as separate WAV files (i.e., one WAV per RECOGNIZE request).

The client may request an URI for the recorded audio, by means of the “Save-Waveform” header in the RECOGNIZE or SET-PARAMS message.

In this case, the MRCP server returns a “Waveform-URI” header in the RECOGNITION-COMPLETE containing an URI to retrieve the WAV file via HTTP, as shown in Table 6: Supported MRCPv2 Messages and Headers.

The basename of this URI is configurable.

For MRCP configuration, see Speech Recognition MRCP Interface – CodyFi.

3.4.2.3 Supported Grammars

The Spitch MRCP server works with pre-compiled, built-in grammars that are uploaded on the ASR speech engines.

The built-in grammars may be GRXML or LVCSR grammars, and they must be prepared in advance using the provided Spitch tools.

The use of a DEFINE-GRAMMAR message isxcvs depicted in MRCP Recognition Scenario diagram as shown in Illustration 2 The MRCP Server Workflow

The built-in grammar may be specified directly in the body of the RECOGNIZE message.

3.4.2.4 Supported Audio Codecs

The MRCP server supports specific audio codecs that differ from the audio codecs of the HTTP Proxy.

Audio codecs are negotiated between the client and the server, upon establishing the MRCP session, by an exchange of SDP using the SIP protocol.

Spitch supports the main codecs used by PBXs:

- PCMU (g711u PCM u-law 8kHz)
- PCMA (g711a PCM a-law 8Khz)
- L16/96/8000 (linear LPCM 8kHz)
- PCMU/97/16000 (PCM u-law 16kHz)
- PCMA/98/16000 (PCM a-law 16kHz)
- L16/99/16000 (linear PCM 16kHz)

Additional codecs can be supported on request.

3.4.3 Summary of MRCP Server Characteristics

The MRCP server supports:

- MRCPv2 TCP with underlying SIP protocol (TCP and UDP)
- Codecs: LPCM (linear PCM 8 and 16Khz), PCMU (g711u), PCMA (g711a).

  Additional codecs can be added on request.
- MRCP recognition messages and headers.

  For more information, see Supported MRCPv2 Message Bodies

  Additional recognition messages can be added on request
- Barge-in and non-barge recognition scenarios.
- Voice activity detection with pre-buffer of audio data to avoid data loss (truncation) of utterances in barge-in scenarios.
- Recording of incoming audio data in WAV format, providing Waveform-URI on request.
- TCP keep-alive on the SIP connections to reset the connections in the event of network interruptions.
- Concurrent ASR sessions connecting to the same or different Spitch ASR back-ends.
- Lightweight architecture, can be collocated with or decoupled of actual ASR back-end nodes to support different load requirements.
- Integration with Spitch process dispatcher to provide load-balancing among symmetric ASR back-end nodes.
- Development in a continuous architecture framework, enforces regression testing on every new release, minimizing bugs.
- Statistics via the Simple Network Manager Protocol (SNMP).
- Rotating logs.

3.5 The HTTPS Proxy Server

The HTTPS Proxy server provides a secure way to post audio data in HTTP mode, over a secure TLS connection, and it supports SSL encryption and TLS up to TLS 1.2.

The HTTPS proxy provides a REST entry point to the speech cluster, and there is typically, only one Active Proxy for each Speech Engine cluster.

The HTTPS Proxy interfaces to the clients through HTTPS instead of the SIP/MRCP/RTP protocols as used by the MRCP server.

A floating IP provides a unique IP for a service that can be moved to a different node at any time, in case of node failure.

The unique IP allows the client to reach the Proxy server, and to send requests to this unique floating IP, thus ensuring continuous availability.

Through the Proxy server, the audio is sent in HTTP POST chunked mode.

The audio can be in Opus/Ogg or in RAW pcm format (in case of ogg/opus stream, it converts it to PCM before sending it to the Speech Engine).

Additional codecs can be added on request.

The HTTPS Proxy can send Audio Files (raw PCM + wav PCM header or ogg/opus) to the internal Spitch Storage, where a cache mechanism avoids uploading identical audio files.

In case no speech engine supports the native audio frequency, the HTTPS Proxy can down-sample audio to 16k or 8k.

For more information, see section on

Proxy Server

Summary of Spitch Proxy Server Characteristics

The Proxy server is able to:

- Handle concurrent HTTPS requests.
- Decrypt incoming requests and encrypt outgoing responses.
- Provide a valid SSL certificate to be checked by the appropriate client.
- Support the HTTP API.
- Support OPUS-encoded content.
- Transcode to PCM before sending to the speech engine.

  RAW PCM is the only audio format that the speech engine interprets. If other formats are used such as Ogg/opus, the proxy converts it to RAW PCM before sending it to the speech engine.
- Provide high-availability.
- Be vertically scalable.

3.6 What Is Cluster Monitoring?

Cluster monitoring is an optional functionality that provides a system health check and alerts of potential problems.

Cluster monitoring is done via the SnmpStatsMonitor agent.

The SnmpStatsMonitor agent provides Speech statistics, and the node state alerts are sent via the Pacemaker stack.

For more information on the SnmpStatsMonitor implementation and configuration, see:

What Is SNMPStats Monitor?

How to Access the SnmpStatsMonitor

SnmpStats Monitor

For information and a description of all the supported statistics, you can request the MIB file.

3.7 What Is SNMPStats Monitor?

The SNMPStats Monitor is an agent that provides general SNMP scalar statistics and system monitoring capabilities.

The SnmpStatsMonitor supports MIB-2 standard statistics along with custom Spitch statistics.

Cluster monitoring is always supported for the Speech cluster, through the SnmpStatsMonitor.

The Storage cluster does not support Snmp Monitoring.

3.8 What Is the Storage Cluster?

The Storage cluster is the place where wav files and voice prints are stored. Waf files and voice-prints may be retrieved for future comparison.

The Storage cluster provides High-Availability to guarantee service continuity.

The data is continuously replicated between the master node and the slave node, so if a node fails or the FS/Disk is corrupted no data is lost.

The storage allows for both storage and retrieval of Audio files and Voice Prints, and it allows attaching meta-data to an Audio or Voice Prints file.

Since the SV engine uses the storage to upload and retrieve Voice Prints, the storage is mandatory only if you are operating a Verification cluster with multiple nodes running the Verification engine. However, depending on your organizational requirements, you may also want to adopt the storage cluster to store the recognition audio files.

The Logical Volumes for audio and voice-prints diagram shows how you could partition the storage if you use it for audio files and voice-prints.

The centralized storage is shared between all aggregated clusters in the Spitch environment, and is accessible by the Voice-print back-end (HTTPS-PROXY and SV speech engines).

The storage cluster is directly accessed by:

- HTTPS Proxy for audio uploading
- SV engines for Enrolments and Voice Prints retrievals.

The Storage is accessed via a REST interface, over SSL plus Basic Authentication.

For information on how to configure the Storage, see:

Storage REST API

Read-only access is provided from outside the storage cluster, while Write access is only allowed from Speech Engine clusters (HTTPS Proxy and verification engines).

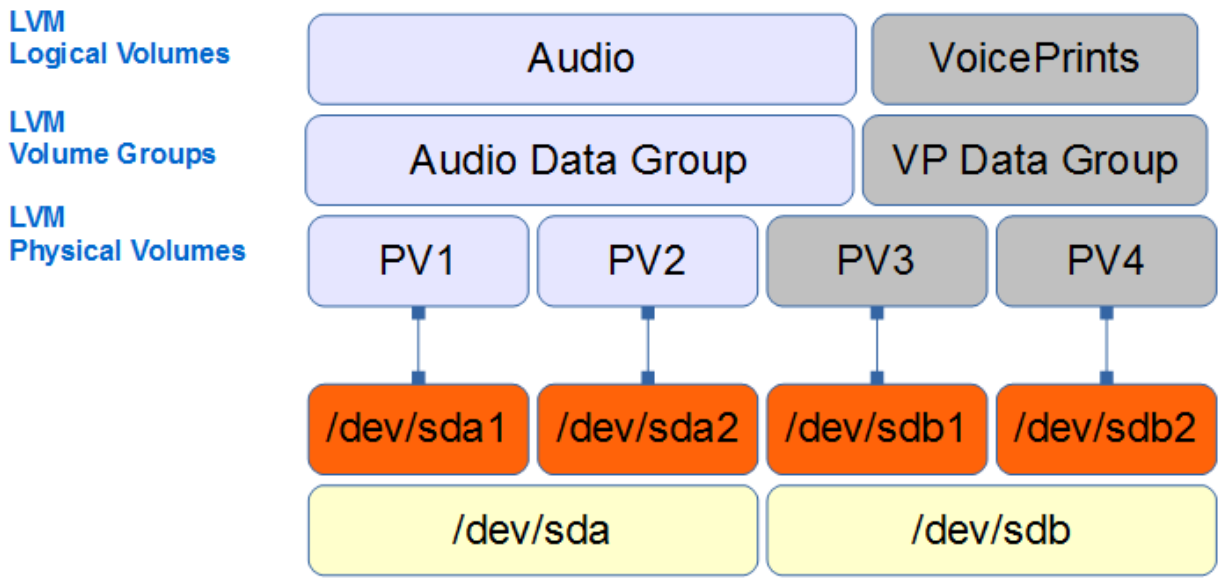

Illustration 3: Logical Volumes for audio and voice-prints

3.9 Storage Cluster Components

The Storage cluster consists of:

- Storage REST Gateway
- DRBD

3.9.1 The Storage REST Gateway

The Storage gateway is the entry point to the storage cluster.

The storage REST Gateway comprises:

- A NGINX server
- LUA scripts

The NGINX server is collocated with the DRBD master along with the floating IP.

In case the DRBD master fails over, NGINX daemon restarts on the new master DRBD.

3.9.2 DRBD

DRBD is a replicated storage solution that provides storage HA.

DRBD mirrors the content of devices such as hard-disks, partitions and logical volumes between servers.

3.10 Spitch Automatic Speech Recognition Workflow

The ASR workflow is described in illustration 4.

The third-party application initiates the recognition process.

The third-party application, typically a PBX system, uses HTTP POST requests over TLS if using the Proxy, or the SIP/MRCP/RTP protocols if using the MRCP server.

The Dispatcher in the speech cluster selects an appropriate ASR speech engine to process the request.

The HTTP Proxy or the MRCP forwards the the request to the selected speech engine, where the Spitch ASR engine performs speech recognition. The results are then sent back to the third-party application.

The third-party application processes the results of the Spitch ASR engine.

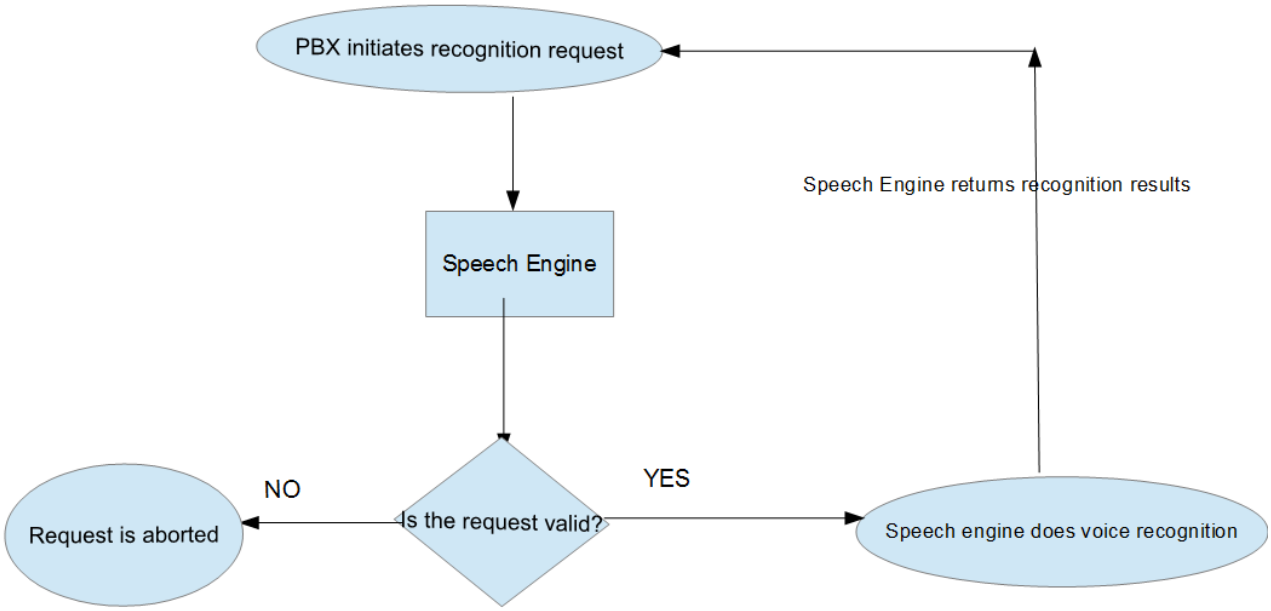

Illustration 4: Speech Recognition Workflow

3.11 Spitch Speaker Verification Workflow

The SV workflow is described in illustration 5.

The third-party application initiates the verification process and sends information about the grammar to be used for the verification and the audio to the Spitch ASR engine.

The third-party application, typically a PBX system, sends HTTP POST requests to the HTTPS Proxy using SSL or TLS.

The Dispatcher in the speech cluster selects an appropriate SV speech engine to process the request.
The HTTPS Proxy forwards the request to the selected speech engine, where the Spitch SV engine performs speaker verification.

The Spitch SV engine compares the voice sent in the request with the stored voice-print to ascertain the caller's identity. Spitch performs speaker verification and sends the results back to the third-party application.

The third-party application processes the results of the Spitch SV engine.

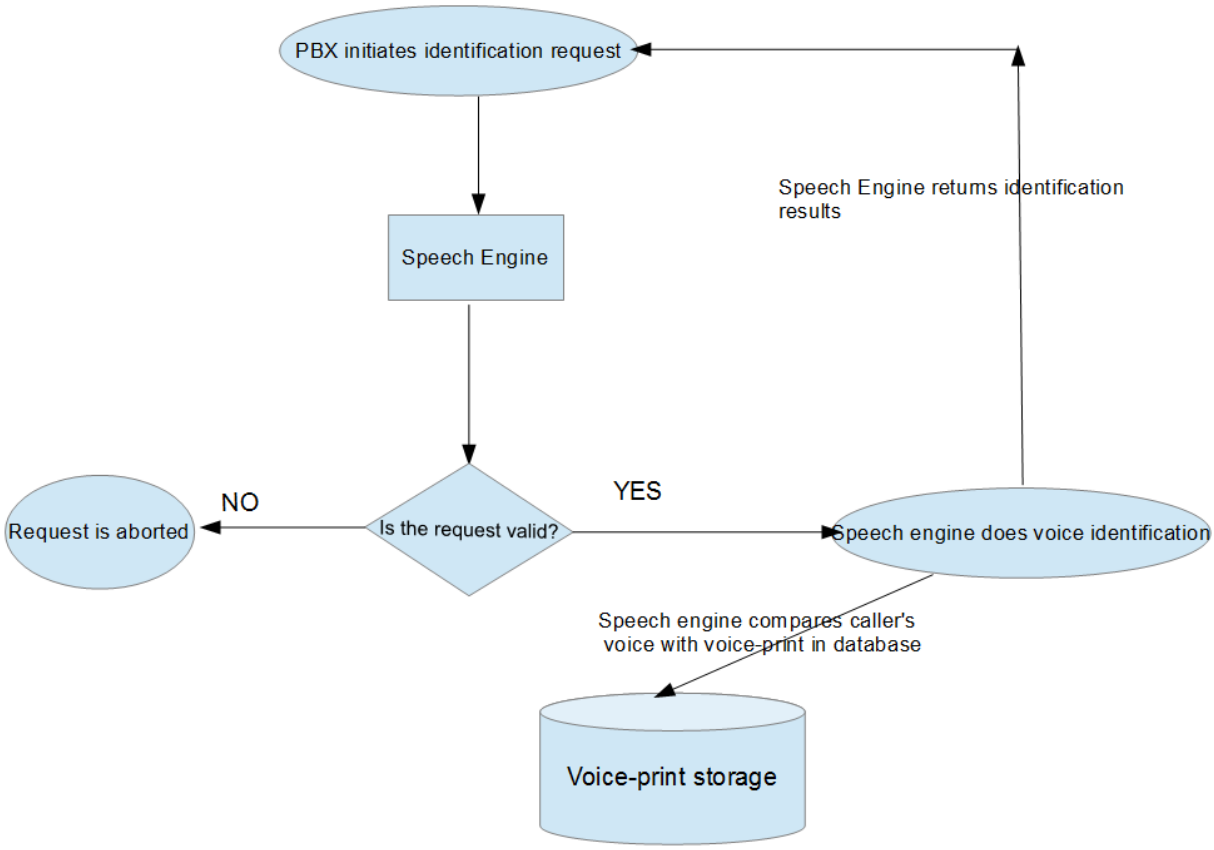

Illustration 5: Speaker Identification Workflow

4 The Spitch Cluster Architecture and Topologies

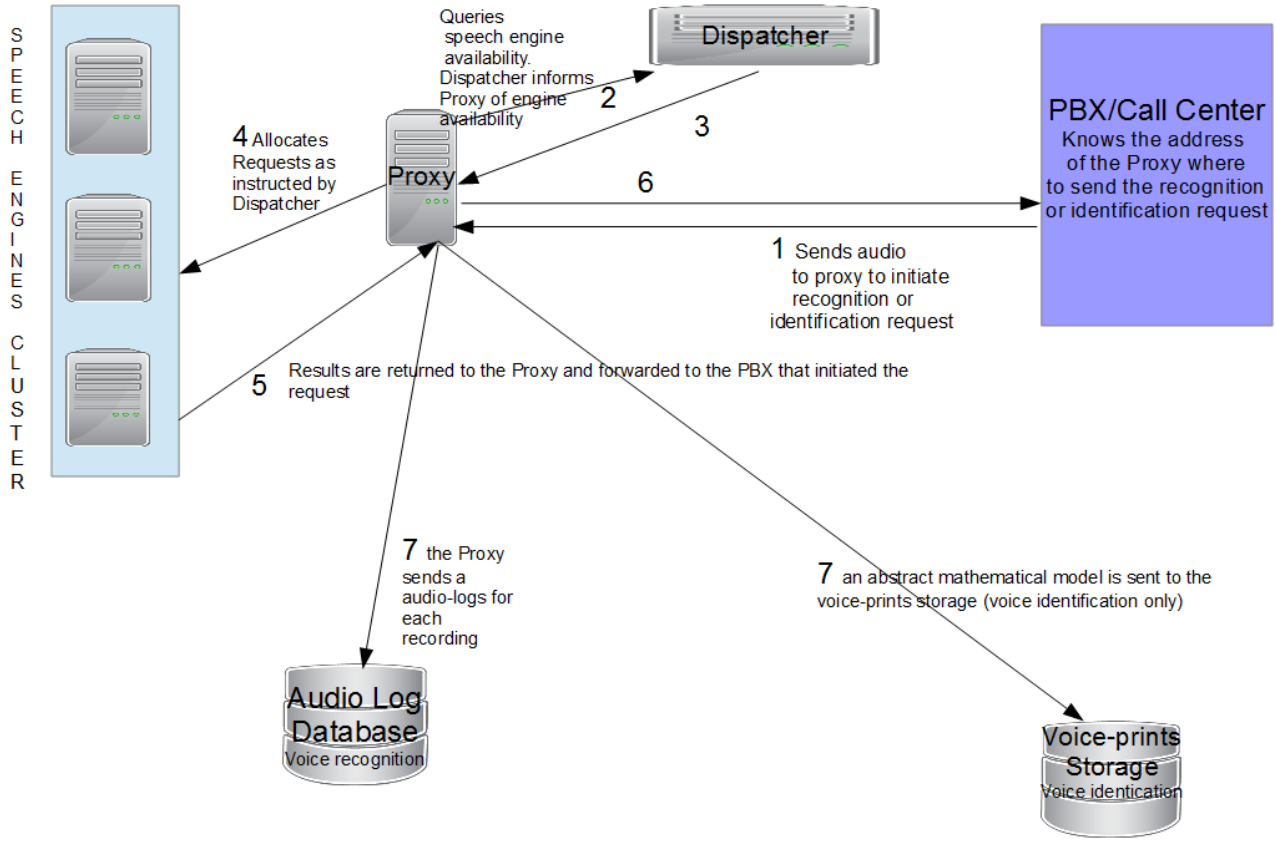

Illustration 6 The Spitch HA Framework

1. The PBX sends an audio to the proxy to initiate the speech recognition or the speaker identification request.
2. The Proxy queries speech engine availability with the Dispatcher.
3. The Dispatcher informs the proxy on where to route the request.
4. The Proxy sends the request to the speech engine as instructed by the Dispatcher.
5. The selected speech engine returns the results of the request to the Proxy.
6. The Proxy forwards the request results to the PBX.
7. The Proxy sends the recording to the audio-log database for voice recognition or to the voice-prints storage for voice identification.

   Note

   The term Proxy is used to refer either to an MRCP Proxy in the case of ASR, or an HTTPS Proxy for ASR and SV.

4.1 Architecture

The Spitch architecture is organized in clusters within an HA framework.

The minimal deployment consists of one cluster over two nodes.

The Spitch ASR consists of a Speech cluster, in this deployment storage is optional.

Spitch SV, and ASR with SV require both a Speech cluster and a Storage cluster. In this deployment, the storage is mandatory.

The Spitch cluster is built for standard x86-based servers (64 bit).

A full system may comprise several speech clusters, plus load-balancing to allow for scalability.

Spitch recommends to operate one cluster per LAN.

The recommended maximum number of nodes per cluster is 32 nodes.

The Spitch cluster architecture may vary depending on your organizational requirements, as shown in Topologies

Each cluster comprises a number of mandatory resources and optional resources as described in the Table 1: The Speech Cluster and Storage Cluster Components

Table 1: The Speech Cluster and Storage Cluster Components

Storage Cluster Components	Speech Cluster Components
Floating IP	Floating IP
REST GW	HTTPs Proxy OR MRCP server
DRDB	Dispatcher
File System	Speech Engine(s)
	ClusterMonitorSnmp

4.1.1 The Spitch Cluster Environment Characteristics

The Spitch cluster runs on Linux, and uses Pacemaker/Corosync to support multiple communications and Open Source DRBD as an additional fail-safe mechanism for the cluster storage.

You configure Spitch resources in Pacemaker, and define location, memory, CPU and order constraints for each cluster.

The Spitch cluster adopts a HA framework to support small and large deployments, up to thousands of nodes, whilst providing fault-tolerance, High-Availability, scalability and security.

The Spitch cluster can run in a private or local network.

The HA framework detects hardware and software faults, and implements a fail-over strategy to provide a continuous and seamless service, in case one or more cluster resources encounter a problem.

4.1.2 Physical Architecture

The physical architecture shows the resources that are required to set up the Spitch environment, and how each resource is linked to other resources.

The Spitch cluster is deployed in your network, and consists of one or more speech servers, which host the speech engines, storage and either a MRCP server or a HTTPS Proxy server..

The HTTPS Proxy communicates with your organization via a TLS secure connection.

For very small deployments (two nodes), you can run both speech engines and storage on the same two nodes (one cluster with two nodes), which means that all the cluster resources can run on the same two-nodes cluster:

- HTTPS Proxy server and/or MRCP server
- ASR speech engines (speech recognition – CodyFi and SignyFi)
- SV speech engines (Speaker Verification– VeryFi)
- Storage components (mainly DRBD + the storage REST gateway)

Illustration 7 CodyFi Speech Cluster Engine,Illustration 9: VeryFi Speech ClusterandIllustration 10 CodyFy and VeryFi Speech Cluster show three possible deployments:

- Only CodyFi
- Only VeryFi
- Both CodyFi and VeryFi speech engines

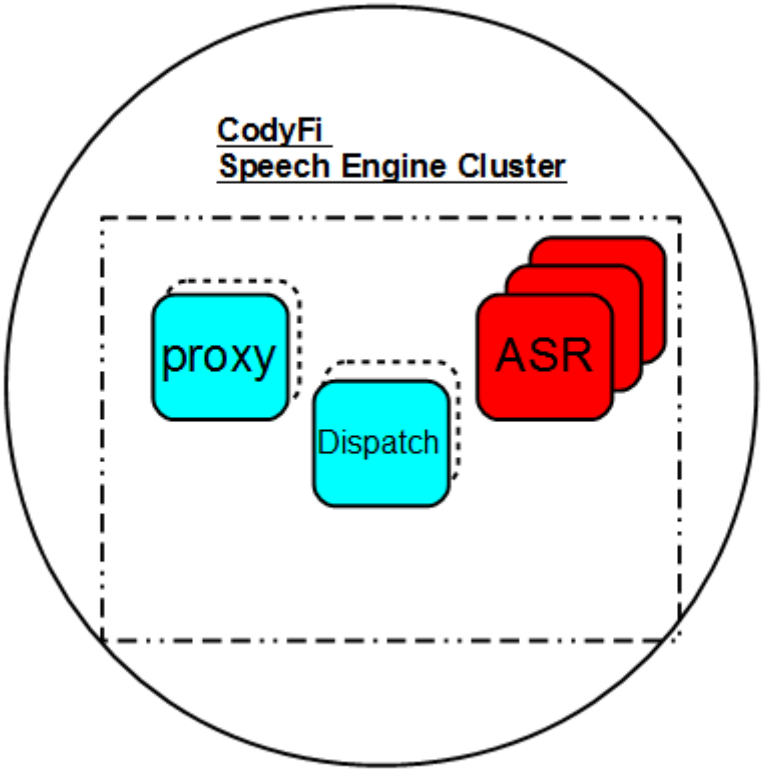

Illustration 7 CodyFi Speech Cluster Engine

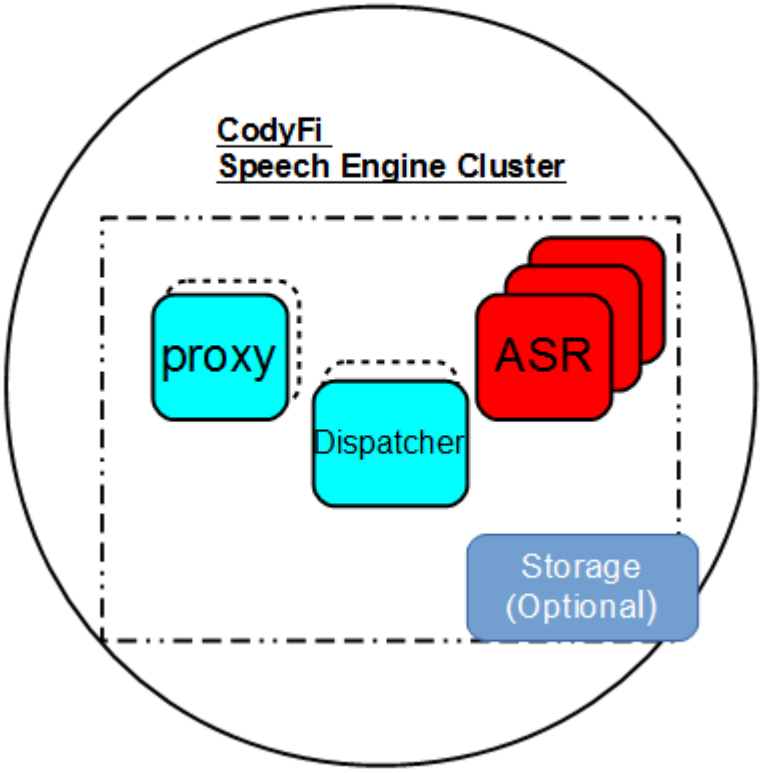

Illustration 8: CodyFi Speech Cluster with Storage

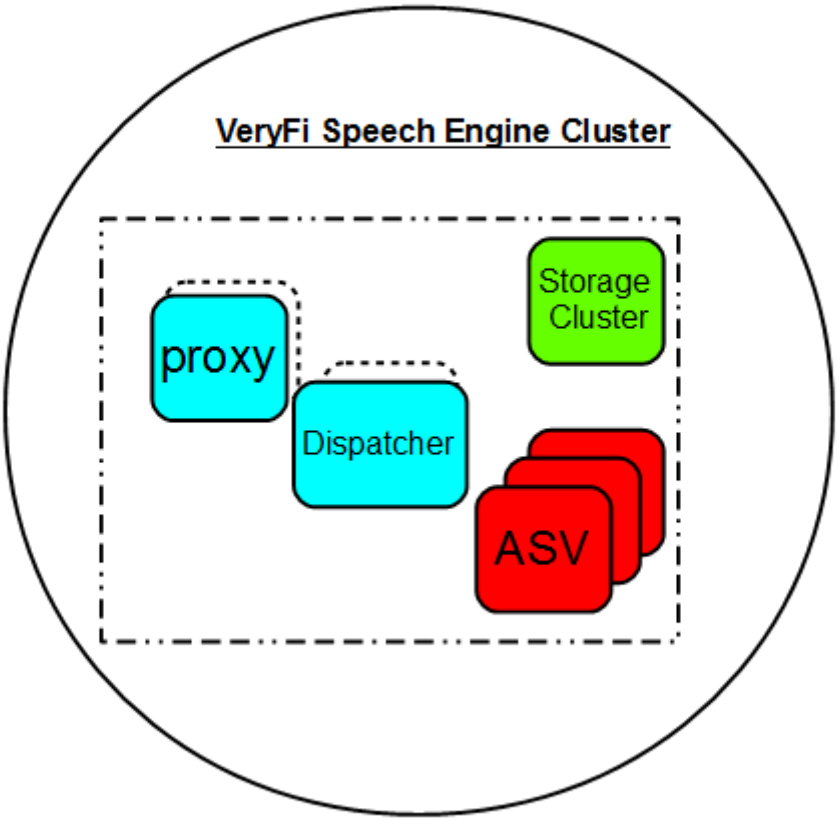

Illustration 9: VeryFi Speech Cluster

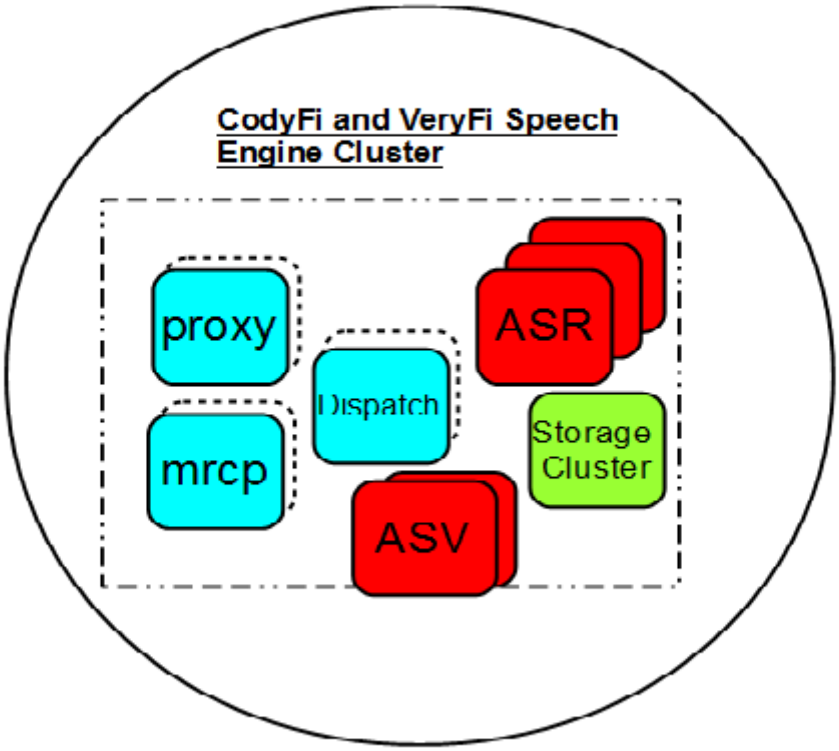

Illustration 10 CodyFy and VeryFi Speech Cluster

4.2 Topologies

The deployment scenarios provide a visual representation of how CodyFi and VeryFi can be set up.

Traditional topology examples are based on a small single-server installation and on multiple servers installation.

Both the small single-server installation and the multiple servers installation allow for more deployment scenarios.

The deployment scenario that you choose is determined by:

- Organizational requirements
- Company constraints
- The number of servers

  The number of servers can vary depending on your on-premises environment.
- The solution that you choose

  CodyFi or VeryFi or both.
- The protocol that you use

  MRCP or HTTPS

Two traditional topologies are:

- Full-cluster deployment from 2 to 32 nodes.

  A two nodes Cluster can include a Storage clusters.

  The Storage cluster is mandatory if you deploy SV, and it is optional if you deploy ASR.
- Cluster aggregation

For detailed deployment recommendations for each component, see Deployment Recommendations.

4.2.1 Two-Nodes to 32 Nodes Cluster Deployment

The minimum deployment is on two-nodes and has one cluster. The maximum deployment is of 32 nodes per cluster.

The cluster may include embedded storage, as shown in Illustration 2 The MRCP Server Workflow. Alternatively, a separate, dedicated storage cluster can be set up.

If you require to use the Verification engine (SV) using a Proxy service, and the Recognition engine (ASR) using a MRCP service, you install one pair of MCRP servers and one pair of HTTPS Proxy servers with active/back-up in the same cluster.

This deployment is characterized by:

- Two partitions and two Open source DRBDs

  This architecture is designed to split audio data from voice-prints data.
- One DRBD that runs on each node and is collocated with the nginx gateway

  Each node has two partitions, which are created during the node set-up.

  DRDB runs on both nodes, one instance is the master DRBD, the other is the Slave DRBD.
- Only one REST gateway with two ports access

  One port for audio files and one port for voice prints

A Spitch cluster consists of the following resources:

- One Active Https-Proxy

  This is the only entry point to the cluster since the ASR and SV Speech engines are not accessible from outside the cluster.

  If your deployment includes ASR and SV in the same cluster, both an MRCP server and a Proxy server are required, granting two entry points.

  All the HTTP traffic goes through the proxy. A back-up mechanism grants continuity of service if the node on which the Proxy is running fails. In this case, the Proxy is restarted on the other node.

  The proxy is always mandatory if you want to access the cluster via our REST interface.

- One Active MRCP Server only for ASR deployment.

  This is mandatory if you want to use the MRCP protocol for Speech Recognition. This is the only entry point to the cluster since the ASR speech engines are not accessible from outside the cluster.

  All MRCP traffic goes through the MRCP server.

  The Active MRCP server runs on a node. In case the node fails, the MRCP backup restarts running on the other node

  The MRCP Server is only mandatory if you want to access the cluster via the MRCP interface.

- N Active ASR and SV Speech Engines

  Typically, the HA stack runs at least one ASR or SV engine per node to leverage your CPU.

- One Active Dispatcher

  To minimize latency and increased performance, we recommend to deploy the dispatcher with the HTTPS Proxy server or MRCP Server on the same node.

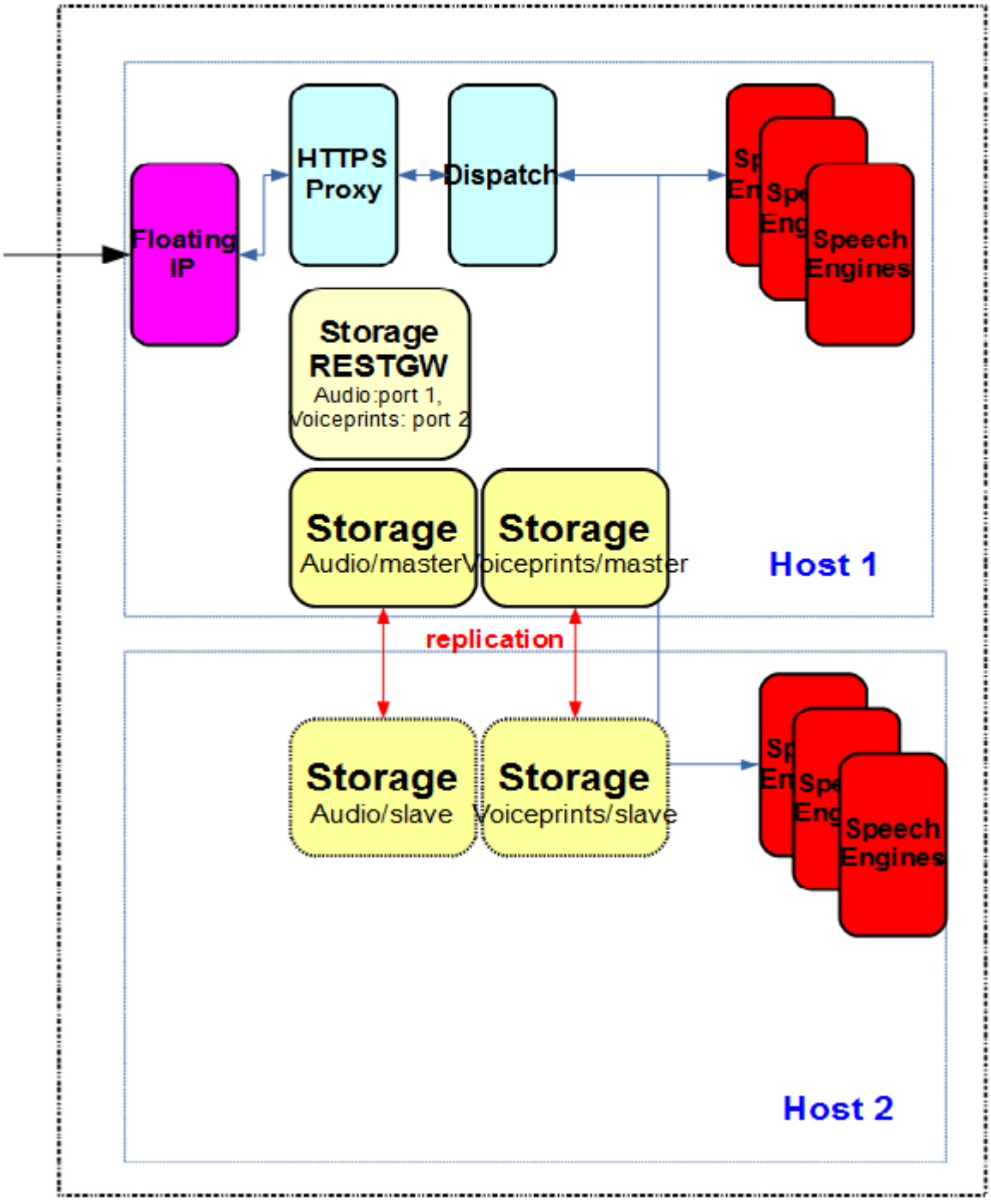

Illustration 11 Two-Nodes Speech cluster with embedded storage

4.2.2 Deployment with a Dedicated Storage Cluster

Your installation consists of two clusters, and each cluster has two nodes.

In this instance, you use a two-nodes cluster for the speech engines, and the second cluster is dedicated to storage.

The two-nodes Cluster drawing shows the architecture of a dedicated storage cluster on two nodes.

Note
The storage cluster is always running on two nodes.

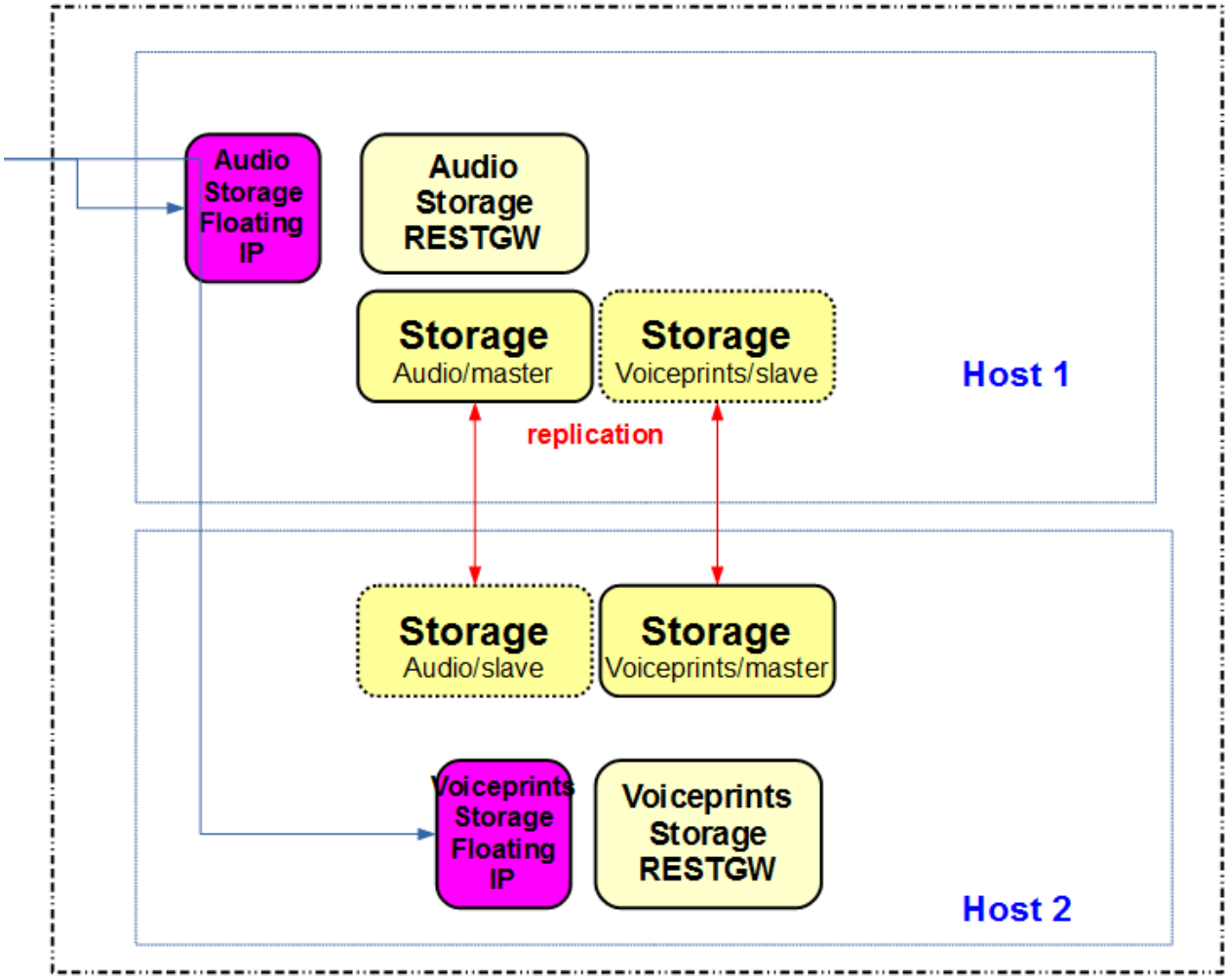

Illustration 12 Two-nodes cluster with dedicated storage

4.2.3 Cluster Aggregation Deployment

Cluster aggregation is recommended for more articulate and resource-intensive networks.

Both the Automatic Speech Recognition engine (ASR) and the Speaker Verification(SV) engine are resource-intensive, and cluster aggregation can address potential performance issues providing increased computing capabilities, and increased throughput.

Each cluster must have its own:

- Proxy server

  OR

  MRCP server

  If it uses the MRCP protocol.
- Dispatcher
- Speech engines

Note

If you require SV or to store ASR audio for post-processing purposes, you must allocate a dedicated cluster for the Storage.

In this scenario, all clusters must be able to access the storage cluster.

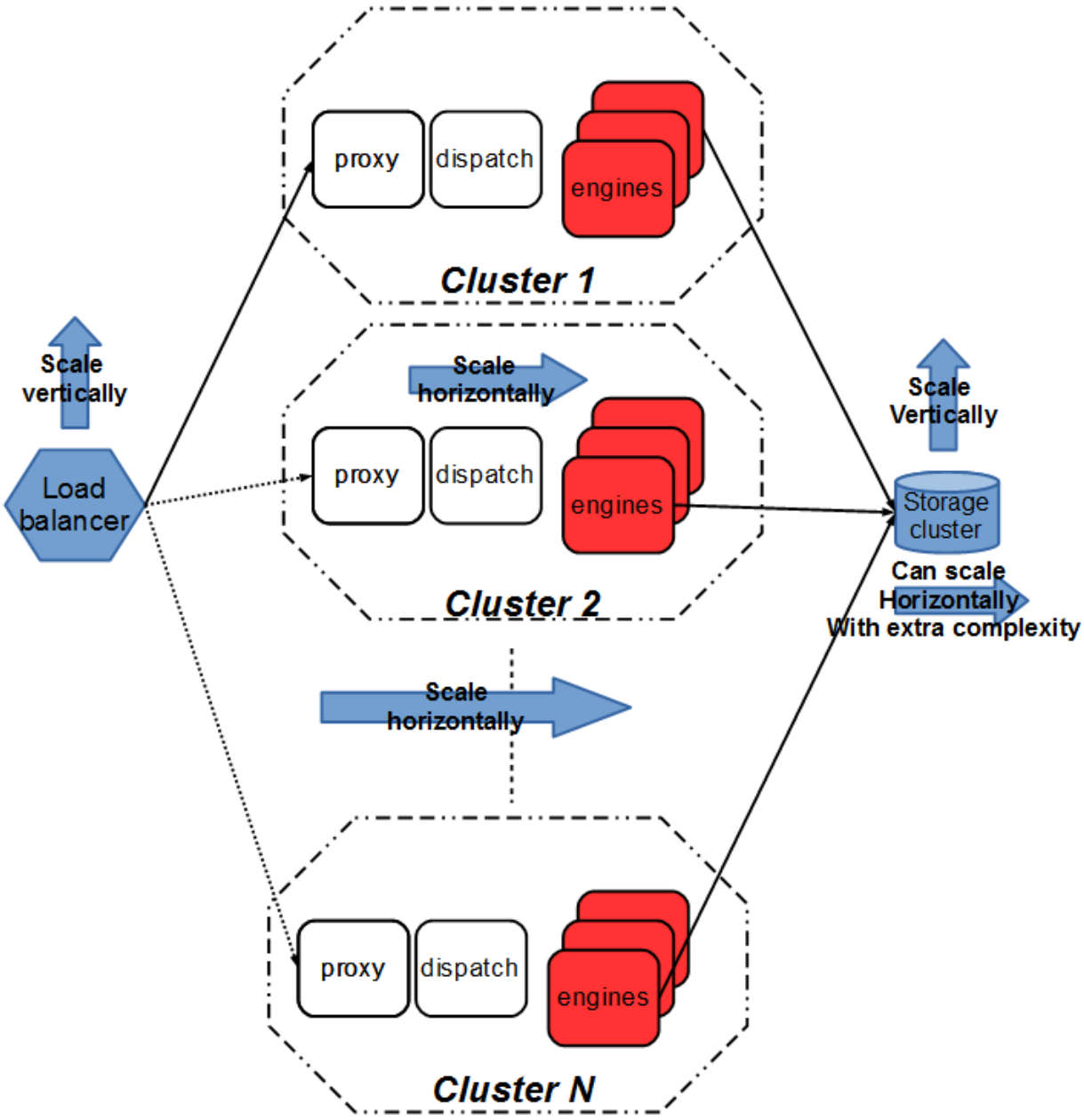

Illustration 13 Cluster Aggregation

4.2.4 Scalability

Spitch solution is scalable, and allows you to manage growth on the network by scaling across a cluster of different nodes (horizontal scalability) or machines (vertical scalability).

Scalability means that you can seamlessly add one or more speech engines.

Horizontal scalability allows leveraging hardware resources by adding (or removing) nodes.

All the CPU-intensive back-end resources can be instantiated on multiple nodes in a N-active mode.

Vertical scalability allows you to add or remove resources to or from a single node.

The Spitch HA framework allows you to add clusters to the back-end through cluster aggregation.

For more information, see Cluster Aggregation Deployment

The Spitch cluster uses a single High-Available Dispatcher process that acts both as a gateway and load-balancer for the number of speech servers in your environment.

In a multiple clusters environment (cluster aggregation), a Dispatcher is used for each cluster.

To add a speech engine it is necessary that:

- The host is plugged to the LAN/VLAN.
- The required packages are installed on the host.

For information on how to install the Speech engine, see Deployment Recommendations and Installing the Spitch HA Clusters

When you add a speech recognition engine, the Corosync stack notifies the Dispatcher that a new grammar was uploaded.

The Dispatcher includes the new speech recognition engine in the round-robin.

The HTTP Proxy is informed that a new speech engine is available where the Recognition requests can be directed.

Scalability Factors

Factors that help you to determine your scaling-up strategy are:

- Number of speech engine installations
- Number of grammars
- Number of clusters
- Number of calls

4.3 Load-balancing

Load-balancing can be achieved either between clusters (inter-cluster load-balancing) or between speech engines (intra-cluster)

4.3.1 Static load-balancing

The Dispatcher adopts static load-balancing to ensure that the requests load is distributed across the speech engines (intra-cluster).

The Dispatcher assumes that:

- All the speech engines nodes are identical in terms of processing power.
- All the Recognition and Verification jobs require the same time to be processed.

For each new request, the Dispatcher does a round-robin, and returns an engine or an IP address where to forward the request.

4.3.2 Spitch Inter-Cluster Load-balancing

Spitch inter-cluster load-balancing is an optional functionality.

Inter-Cluster load-balancing is used when a single cluster cannot meet the load/capacity requirements and several clusters need to be configured.

Spitch uses LVS (Linux Virtual Server) for cluster load-balancing.

LVS is implemented in the kernel and can run in three modes:

- LVS-NAT

  Packets are received from users, destination ports and IP addresses.

- LVS-DR

  LVS-DR implements direct routing: Packets received from the user are sent directly to the server.

- LVS-TUN

  Packets received from users are encapsulated first, and then sent to the server.

LVS is placed at the front-end of the back-end speech cluster.

Two instances of LVS in Active/backup mode are run on a Pacemaker cluster to prevent downtime in case of failure.

Subscribers access the network through a single public IP, the incoming HTTPS connections are distributed to each individual cluster.

The Load-Balancer can only load-balance HTTPS requests, since MRCP traffic cannot go through the load-balancer.

Information on inter-cluster load-balancing is available on request.

4.4 Deployment Recommendations

Spitch solutions require hardware and third-party software resources to be installed together with the Spitch software.

In particular, Spitch uses open source Pacemaker to create and configure the Spitch cluster.

Deployment of the Spitch solution starts with the installation of the Spitch software (CodyFi, VeryFi).

Installation refers to the process of uploading and unpacking the Spitch solution modules to the back-end, and deploying the contents.

Creation and configuration of the Spitch cluster is carried out with Pacemaker.

When the Spitch software is installed, you proceed to install Pacemaker to create and configure the Spitch cluster, and finally you install the Storage cluster.

This sequence is highlighted in Installing the Spitch HA Clusters

4.4.1 Dispatcher

It is recommended to collocate the Dispatcher with HTTPS-PROXY and/or MRCP server.

To view how to install and configure the Dispatcher, see Configuring Utilization Strategy and Speech Cluster Configuration & Installation.

4.5 MRCP Server

Configuration of the MRCP server is based on the opensource unimrcp configuration file format.

All configuration options are described in:

http://www.unimrcp.org/manuals/pdf/ServerConfigurationManual.pdf

Spitch also supports custom parameters for the recognition engine (<engine> tag), as shown in the Recognition Engine Parameters.

Table 2: Recognition Engine Parameters

Param	Value	Description
Waveform-uri-base	URL string	URL base for the waveforms recorded by the MRCP server.
Prebuffer-ms	Number in miliseconds	Length of pre-buffered audio prior to voice activity detection
Dispatcher-host	IP:port	IP and port where the dispatcher process is running

Example:

```
<param name="waveform-uri-base" value="http://172.30.30.110:8000"/>
<param name="prebuffer-ms" value="2000"/>
<param name="dispatcher-host" value="127.0.0.1:8081"/>
```

Note

The dispatcher is usually collocated with the MRCP server (or the HTTPS proxy) on the same node. If this is not the case, a floating IP, instead of the localhost IP, has to be specified.

The HTTP server that serves the MRCP waveform-URIs is also collocated so that it has access to local wav files stored in /var/opt/spitch/mrcp/var.

However, the floating IP address of the MRCP server has to be specified in the 'waveform-uri-base", instead of its private address. This is the IP address that the client requires to connect to the server.

4.5.1 Proxy Server

The proxy is configured through a asrproxy.xml file. The asrproxy.xml file is specified via the command line by entering:

```
asrproxy -c <path>
```

<path> points to the directory that contains the asrproxy.xml file.

The xml contains the tags as shown in the example.

```
<config>
    <dispatcher>
       <host>localhost</host>
       <port>8081</port>
       <timeoutMs>50</timeoutMs>
    </dispatcher>
    <!-- if you want to support unsecure SSLv2v3 comment tls version -->
    <ssl>
        <tlsVersion>1.2</tlsVersion>
        <privateKeyFile>asr-proxy-rsa.key</privateKeyFile>
        <certificateFile>asr-proxy-rsa.crt</certificateFile>
        <caPath></caPath>
        <sslBacklog>128</sslBacklog>
        <!--
         if sslBacklog > 128, you must update the system settings :
         sysctl -w net.core.somaxconn=N where N >= sslBacklog
     -->
        <cipherList>ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH</cipherList>
        <!-- verificationMode :
             NONE    -> The server will not send a client certificate request to the
client.
             RELAXED -> The server sends a client certificate request to the client.
                        The certificate returned (if any) is checked.
                        If the verification process fails, the TLS/SSL handshake is
immediately terminated.
             STRICT ->  If the client did not return a certificate, the TLS/SSL
handshake is immediately terminated.
             ONCE   ->  Only request a client certificate on the initial TLS/SSL
handshake.
```

```
                        Do not ask for a client certificate again in case of a
renegotiation.
        -->
        <verificationMode>NONE</verificationMode>
    </ssl>
    <maxConnectionsQueued>512</maxConnectionsQueued>
    <logLevel>debug</logLevel>
    <streamReadBlockSize>512</streamReadBlockSize>
    <maxThreads>512</maxThreads>
    <!-- comment out mongo for uuid authentication - DEPRECATED
    <mongo>
        <host>sandbox-developer.spitch.ch</host>
        <port>27029</port>
        <dbname>spitch-developer-portal</dbname>
        <user>spitchAG</user>
        <password>spitchAG</password>
        <method>MONGODB-CR</method>
    </mongo>
    -->

  <portal>
      <host>sandbox-developer.spitch.ch</host>

      <!-- comment out oauth2 to enable oauth2 authorization
      <oauth2>
          <validateEndpoint>/validate</validateEndpoint>
      </oauth2>
      -->
      <!-- comment out recording to enable recording
      <recording>
          <recordingEndpoint>/getSignedRequest</recordingEndpoint>
      </recording>
      -->
    </portal>
<portal>
            <host>sandbox-developer.</host>
```

```
        <!-- comment out oauth2 to enable oauth2 authorization
        <oauth2>
              <validateEndpoint>/validate</
        </oauth2>
        ->
        <!-- comment out recording to enable recording
        <recording>
            <recordingEndpoint>/
        </recording>
        -->
        </portal>
        <storage>
           <credentials>
               <user>spitch</user>
              <password>spitch</password>
           </credentials>
        <audio>
                <!-- floating IP of the storage Rest gw handling Audio records -->
               <uri>https://52.58.118.134:8080</uri>
          </audio>
          <voiceprints>
          <!-- floating IP of the storage Rest gw handling Voiceprint records -->
              <uri>https://52.58.118.134:8080</uri>
          </voiceprints>
    </storage>
</config>
```

4.5.2 SnmpStats Monitor

The SNMP Stats Monitor daemon is based on net-snmp lib and, as such it is configured with the same configuration file format as the standard snmpd daemon.

The configuration file is stored in*:*

`/opt/spitch/stat-monitor/conf/agent.conf.`

You can configure parameters such as access, community and traps that you want to be sent to your management station through the agent.conf

By default, SnmpStatsMonitor is set to localhost access on port 161:

`agentaddress udp:localhost:161`

Note

You may change the *localhost* access name.

4.5.3 Storage Cluster

As a minimum, the deployment of a storage cluster can be operated on two nodes.

Typically, only two nodes for DRBD are required to operate.

For very high throughput, you can create multiple Storage cluster and a Load-balance.

For storage, you use as a maximum two clusters.

If you require to absorb additional throughput, you can scale your nodes vertically adding CPU and RAM, or you can scale the storage device horizontally through the increase of SSD (Solid State Device) for each node.

Spitch can provide you with custom scripts and assist you on this.

You use at a maximum two clusters:

- One for voice prints
- One for audio, and upgrade your SSD.

For very high throughput, you can:

- Scale the nodes vertically
- Scale the storage device horizontally and vertically

Spitch provides all the scripts for the recommended setup.

Minimum Storage Cluster Requirements

The following resources are required to run a Storage HA cluster :

- 1 Master/Slave DRBD resource that backs up one dedicated disk partition for Audio Storage.

  Active and Backup must have similar partition size and File System type.
- 1 Master/Slave DRBD resource that backs up one dedicated disk partition for Voice Prints Storage (same Disks constraints than for Audio)
- 1 FileSystem resource that runs on a DRBD Audio master node.
- 1 FileSystem resource that runs on a DRBD Voice-prints master node.
- 1 Active Custom NGINX HTTP server for audio with a floating IP

  The active REST gateway can be restarted on different nodes with its floating IP.
- 1 Active Custom NGINX for voice-prints with a floating IP

  The active NGINX must always be co-located with the DRBD master resource to avoid the need of using NFS .

5 Installation Prerequisites

To successfully install the Spitch HA framework ans Spitch services, your system must meet the following requirements:

- Hardware and memory

  For the Speech engine installation

- Hardware and memory

  For the REST Gateway

- CPU
- Memory

  Specific requirements are highlighted for the CodyFi speech engine and VeryFi speech engine.

Note

To guarantee performance, Spitch recommends to meet or exceed the system requirements specifications.

5.1 Hardware Requirements

- 64 bit processor
- Red Hat Enterprise Linux/CentOS 7 Linux distro, kernel 2.7 or above
- 16 Gb minimum memory for the Speech Engine

  The Speech Engine contains the resource-intensive Lingware.

  When evaluating the memory requirements, take into account:

  - Number of nodes
  - Maximum number of speech engines per node

    Multiply the memory for the number of speech engines on the node.

5.1.1 CodyFi Speech Engine

We recommend allocating one modern vCPU per CodyFi port when using LVCSR models.

Since real time speech recognition is tuned to take approximately 50-75% of a vCPU on average (higher/lower CPU usage depend on utterance complexity).

When using closed grammars, a single core can serve multiple, simultaneous requests.

The exact number depends on specific applications.

5.1.2 VeryFi Speech Engine

We recommend allocating one modern vCPU per each 10 VeryFi ports, since real time speaker verification takes approximately 6-12% of a vCPU.

5.1.3 SnmpAgent and Dispatcher

- 64 bit processor
- 1 core for small throughput
- 512 Mb

5.1.4 HTTP Proxy

In terms of memory, HTTP Proxy uses around 180k per connection.
Virtual memory grows by 8meg per threaded connection.

When the pool is exhausted, the connections are queued and the memory is expected to stall.

Because the speech engine operates asynchronous processing, it is preferable that the proxy server is configured to use a number of threads that is equal to the maximum number of expected concurrent connections.

5.1.4.1 SSL Processing

Most of the CPU memory usage is for SSL processing.

SSL processing is CPU resource-intensive, depending on factors such as:

- Server key size (asymmetric key)
- Key exchange protocol (symmetric key)
- Bulk cipher (used for encrypt data with symmetric key)
- Quantity of data transferred through the SSL connection

Multiple cores multiply the number of Proxy threads that the CPU can handle , as well as the usage of SSL caching.

USE CASE

This use-case is based on the following specifications:

- Server key is 2048 (standard)
- The key exchange is legacy (no forward privacy)
- Bulk cipher high (256 AES)
- Audio file between 50k to 1m
- Caching is not always possible

  It is assumed that for local deployments, the number of clients is fixed, and able to use persistence http session (KA) and to resume SSL sessions (cache session id + symmetric key).

  As a rule of thumb and depending on data file size, using SSL cache improves by 2 to 5 non cached SSL, a safe value for audio PCM file is x2 on an average core (2.3 GHz, no specific AES instruction).

  The Proxy can handle around 60 new requests per sec (no caching). With SSL reuse, the Proxy should handle at least 100 threads.

New users not using SSL cache,
Based on these specifications, the CPU can handle around 60 requests per second and per medium modern core.

Existing users re-using SSL sessions (SSL cache)

Based on these specifications, the CPU can handle at least 120 requests per second per medium modern core.

FULL-CLUSTER
For a full cluster you may have a dedicated node for the Proxy, and high-end servers, for example, with eight cores. As an alternative, if you want to reuse low-end devices, you can aggregate small clusters until you reach the required throughput.

5.1.5 Storage REST

Storage REST is a third-party software.

STORAGE-REST clients such as SV engines and Proxies use SSL caching.

Nginx architecture is better optimized for event-driven requests, so performance, in terms of request per seconds, is enhanced.

You can find performance benchmarks in the NGINX Performance link

https://www.nginx.com/wp-content/uploads/2014/07/NGINX-SSL-Performance.pdf

Existing users re-using SSL sessions (SSL cache)

Based on the use case as shown in SSL Processing specifications, the CPU can handle around 800 requests per second per medium modern core (audio) and 2000 requests per second for voice-prints.

5.1.6 DRBD

Kernel based and optimized.

For more information, see:

http://blogs.linbit.com/p/983/intel-dc-3700-sata-ssd/

5.2 Memory Requirements

5.2.1 CodyFi Speech Engine

Each CodyFi server may support one or more grammars.

RAM must be allocated for each grammar.

The LVCSRs grammar size is in theory unbounded, but we typically prune them to require a maximum of 2GB for each grammar.

In memory constrained environments, they could be pruned further, although sacrificing accuracy.

Closed grammars typically require around 100MB each.

Size is mostly dependent on the acoustic models that you use.

Since memory is shared between all ports running each grammar, these figures are approximately independent on the number of ports used. In fact, despite an increase in memory usage for each new port, the main driver for memory usage is still the model size.

Because of subtle side-effects of parallelization on memory usage, more memory is consumed for each new port. However, the main driver of memory usage is still the model size.

Memory usage for other resources of the cluster, such as HTTPS proxy, MRCP server, storage sub-system and monitoring sub-system,is negligible compared to the CodyFI and the VeryFI engines.

5.2.2 VeryFi Speech Engine

Each VeryFi server may support one or more genders for one or more languages.

For each gender of each language, memory usage is approximately 2GB.

Since the memory is shared between the ports, this figure is roughly independent on the number of ports used.

Memory usage for other components of the cluster, such as HTTPS proxy, storage sub-system and monitoring sub-system,is negligible compared to the CodyFi and the VeryFi engines.

6 Installing the Spitch HA Clusters

6.1 Installation Procedure for CodyFi and VeryFi clusters

To install and run the Speech Clusters you need to

1. Install and configure Spitch Software packages on some or all nodes of the cluster (mandatory)
2. Install and configure Pacemaker Software packages on all nodes of the cluster (mandatory)
3. Deploy resources in the Cluster (mandatory)
4. Configure Utilization Strategy (optional)
5. Configuring Fencing (recommended)

6.1.1 Spitch SW installation and Configuration

You shall have been provided with 2 bundles (tgz files) :

- spitch_apps_bundle.<distribution>.tgz : contains a set of packages (debian files for UBUNTU distribution, rpm files for CENTOS/RHEL ones). The following packages shall be bundled:
  - spitch-proxy (proxy app)
  - spitch-mrcp (mrcp app)
  - spitch-dispatcher (dispatcher app)
  - spitch-monitor (monitor app, mib and libs)
  - spitch-codyFi (CodyFi app)
  - spitch-veryFi (VeryFi app)
  - spitch-fence (fencing scripts)
  - spitch-ha-misc (set of tool scripts used by cluster)
  - spitch-ha-resourceagents (custom cluster resource agents OCF scripts)
- spitch-<lingwares>.deb|rpm: contains a set of codyFi/veryFi lingwares adequate for your environment.

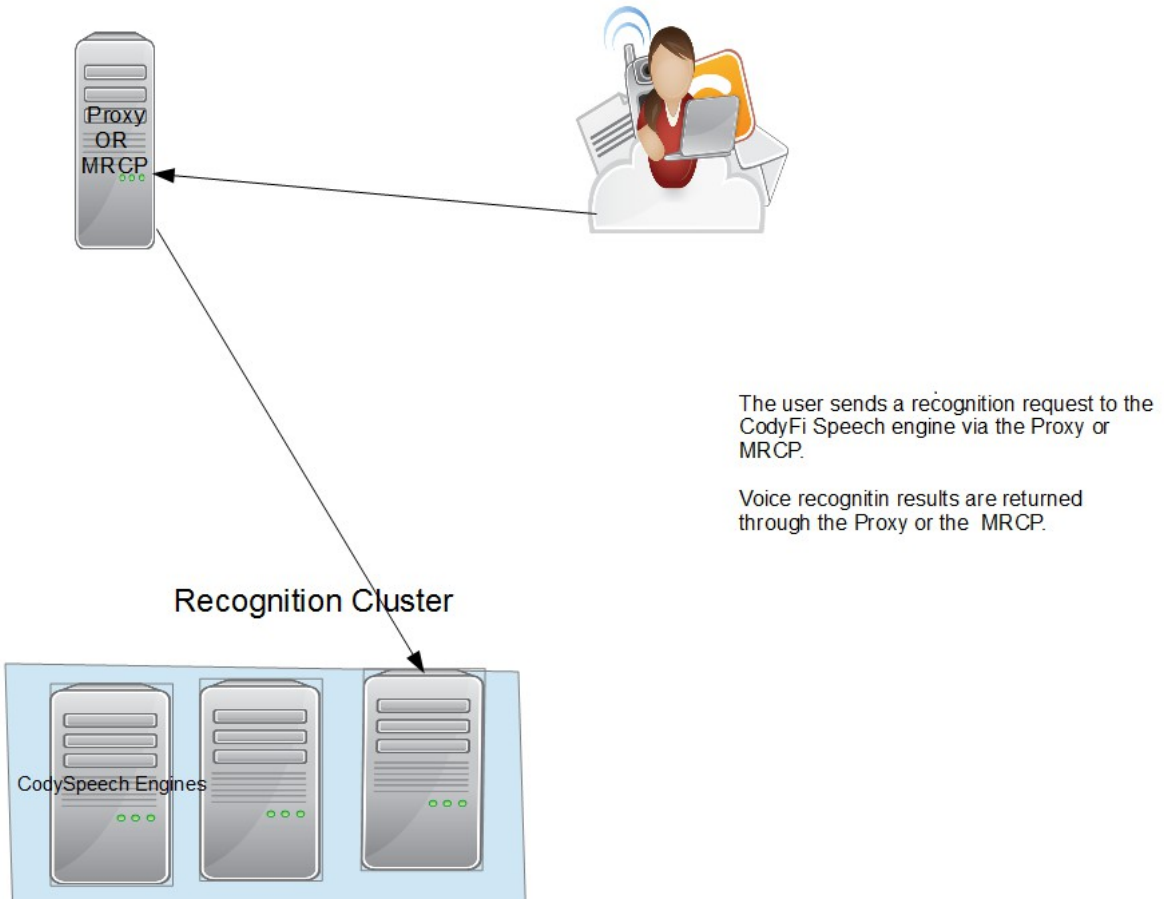

Illustration 14: Recognition Cluster

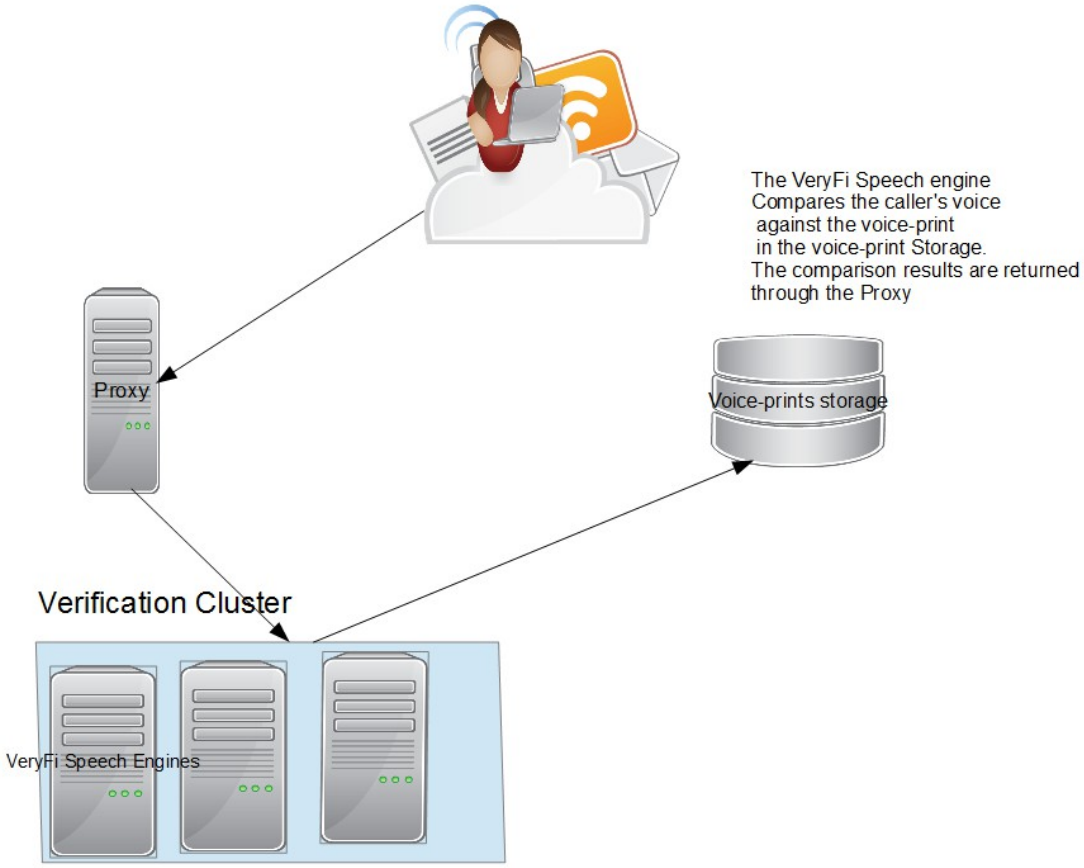

Illustration 15: Verification Cluster storage

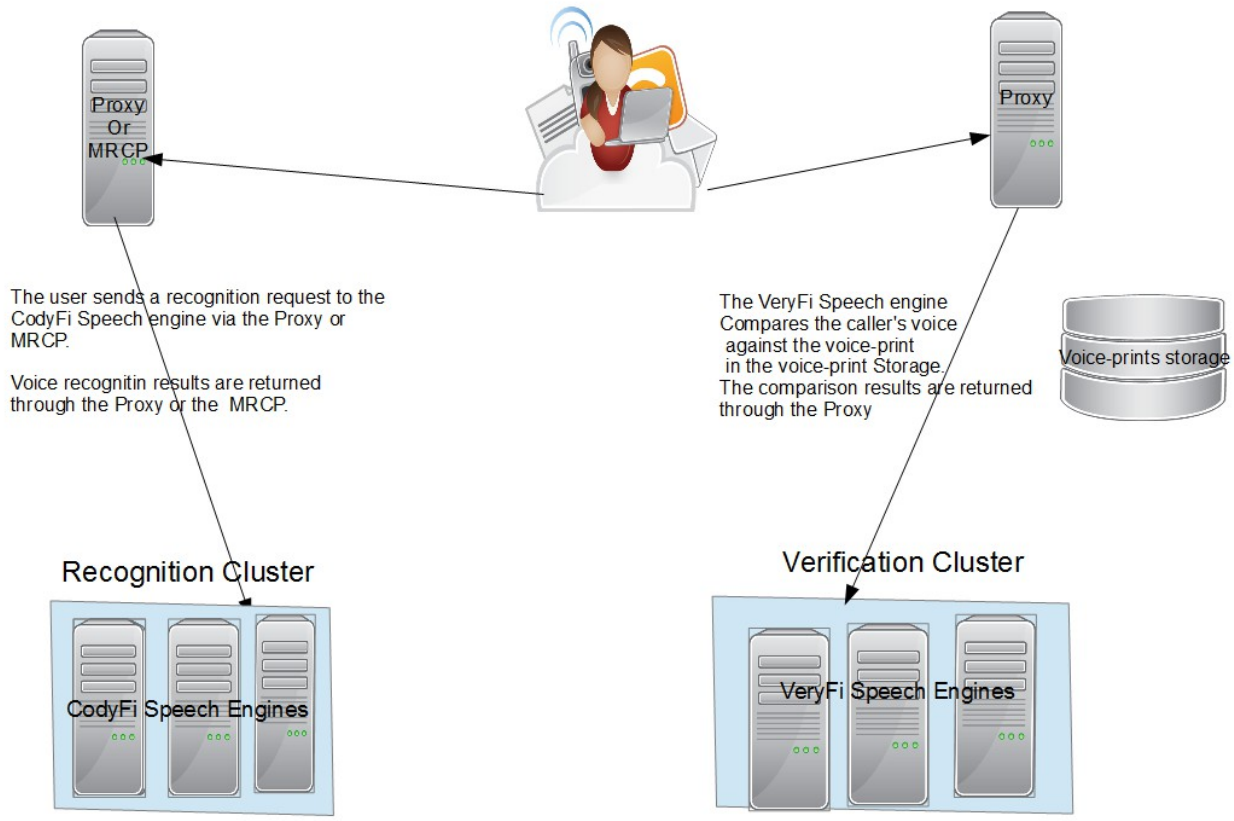

Illustration 16: Recognition and Verification Clusters

To install those packages, do the following:

→ copy apps bundle & lingwares package on ALL nodes

You must copy spitch_apps_bundle.<distribution>.tgz to all nodes in your cluster. You must copy spitch-lingwares.deb|rpm to all nodes that can run speech engines (codyFi or veryFi). To do so you can use what is available on your systems (scp, ftp).

→ untar bundle on ALL nodes

```
tar -C /tmp -xzvf spitch_apps_bundle.<distribution>.tgz
```

→ install all packages on relevant nodes

To install a debian package (Ubuntu), you use

```
sudo dpkg -i <package_filename>
```

To install a rpm package (CentOS/RHEL) you use

```
sudo rpm -i <package_filename>
```

Note that some package have dependencies. When you install the package you shall be notified of missing dependencies, so that you can install them via dpkg -i or rpm -i. (Alternatively you can first check dependencies using dpkg -I <package_filename> or rpm rpm -qpR <package_filename>)

You can either install all packages on all nodes, or you can install only relevant packages on selected nodes (I.e install mrcp app only on the nodes that will run a mrcp). However all non app packages must be installed on all nodes (spitch-ha-misc and spitch-ha-resourceagents).

→ create/update configuration on MASTER node

This step is deployment-specific. The user configures all files locally on the master node (MASTER node is any node of your choice that will be referenced as MASTER), and pushes the configuration to all nodes.

CodyFi/VeryFi Files configuration.

This step is only needed if you want to change the default ports of your speech engines.

For all lingwares in `/opt/spitch/asr/backend/lingware,` change the xml with appropriate settings.

For more information on how to configure the engine, see:

Deployment Recommendations

MRCP Files Configuration

Configure the unimrcp files through the `/opt/spitch/mrcp/conf/mrcpserver.xml` file.

For more details on MRCP configuration, see MRCP Server

Note

You are only required to configure the MRCP, if your install uses a MRCP server.

Proxy Configuration

Configure the proxy files through the /opt/spitch/proxy/conf/proxy.xml
For more details on how to configure the Proxy server, see section Proxy Server.

Monitor Configuration

Configure the proxy files through the /opt/spitch/stat-monitor/conf/agent.conf.
For more details, see section on SnmpStats Monitor .

Note

You are only required to configure the Proxy files, if your setup uses a Proxy server.

Once you have make the needed changes to all your apps, you must push the changes to all other nodes in the cluster.
To do that you can make a tar of the xml configuration file and scp them to other nodes.

```
tar cvfz spitch_conf.tgz $(find /opt/spitch/asr/lingware -name
"*.xml") $(find /opt/spitch/speaker_verification/apps -name
"*.xml") /opt/spitch/proxy/conf/ /opt/spitch/mrcp/conf
/opt/spitch/stat-monitor/conf
```

Then copy and unpack this tarball to all nodes (using scp or whatever you like and is supported)

→ configure log-rotate on ALL nodes

Log-rotate allows you to compress all the log files.

To enable and configure this option, follow the steps as shown:

```
cat /etc/logrotate.conf
/var/opt/spitch/asr/backend/log/*.log {
 missingok
 notifempty
 compress
 size 10M
 daily
 copytruncate
}
/var/opt/spitch/speaker_verification/apps/log/*.log {
 missingok
 notifempty
 compress
 size 10M
 daily
 copytruncate
}
/var/opt/spitch/dispatcher/log/*.log {
 missingok
 notifempty
 compress
 size 10M
 daily
 copytruncate
}
/var/opt/spitch/proxy/log/*.log {
 missingok
 notifempty
 compress
 size 10M
 daily
 copytruncate
}
/var/opt/spitch/stat-monitor/log/*.log {
 missingok
 notifempty
 compress
 size 10M
 daily
 copytruncate
}
```

The log-rotate configuration is the last step in the Spitch software installation.

Spitch software is installed and configured on all nodes. However, you won't be able to run the Dispatcher until you install and set up the Pacemaker cluster framework.

6.1.2 Speech Cluster Configuration & Installation

To add one or more clusters, you install Pacemaker.

Installing Pacemaker entails the following steps:

- Install Pacemaker on all nodes
- Set up and start a cluster on the Master node
- Create HA resources on Master mode
- Create deployment-specific resources on the Master node

  The deployment-specific resources are:

  - MRCP

    OR

  - Proxy

6.1.2.1 Pacemaker framework setup

6.1.2.1.1 CentOS / RHEL distributions

→ setup centos repo on ALL nodes (RHEL only)

```
cat /etc/yum.repos.d/centos.repo
[centos-7-base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?
release=7&arch=$basearch&repo=os
baseurl=http://mirror.centos.org/centos/7/os/$basearch/
gpgcheck=0
enabled=1
```

→ install pacemaker on ALL nodes

If you are on Centos6/RHEL6

```
sudo yum install pacemaker pcs resource-agents fence-agents
sudo yum install cman
sudo service pcsd start
sudo chkconfig pcsd on
```

If you are on CentOs 7

```
sudo yum install pacemaker pcs resource-agents fence-agents
```

```
sudo systemctl start pcsd.service
sudo systemctl enable pcsd.service
```

sudo systemctl stop firewalld.service

If you are on RHEL 7

```
sudo yum install pacemaker pcs resource-agents fence-agents
sudo systemctl start pcsd.service
sudo systemctl enable pcsd.service
```

Then you can configure cluster password

```
sudo passwd hacluster
```

→ setup & start cluster on MASTER node

```
sudo pcs cluster auth -u hacluster -p your_password hostname1 ..
<hostnameN>
sudo pcs cluster setup --enable --transport udpu --name mycluster
hostname1 .. <hostnameN>
sudo pcs cluster start --all
```

NOTE

It is recommended to use the FQDN instead of short hostname if you plan to use fencing on CentOS/RHEL 6.

NOTE

The --transport udpu configure corosync to use unicast. If your network support multicast it is recommanded to use --transport udp instead (multicast).

If you don't plan to use fencing you must disable STONITH:

```
sudo pcs property set stonith-enabled=false
```

Note

The resources WON'T START if you don’t create at least 1 fence device while the property is enabled.

If you only have 2 nodes, set the ignore-quorum property:

```
sudo pcs property set no-quorum-policy=ignore
```

Note

The ignore-quorum property is a Pacemaker setting that allows you to run a two-nodes system in the event of node failure. To have a quorum of nodes, more than half of the total number of nodes must be online. In a two-nodes scenario, in the event of node failure, the quorum cannot be met. You keep the remaining node online, setting the no-quorum -policy to ignore.

At this stage your cluster is up and running, but no resources are started (check that all nodes appears online doing:

```
sudo pcs status
```

Example output

```
Cluster name: mycluster
Last updated: Tue Nov 29 16:16:29 2016      Last change: Fri Nov
25 09:14:31 2016 by root via crm_resource on ip-172-31-23-53.eu-
west-1.compute.internal
Stack: corosync
Current DC: ip-172-31-30-211.eu-west-1.compute.internal (version
1.1.13-10.el7_2.2-44eb2dd) - partition with quorum
4 nodes and 15 resources configured

Online: [ ip-172-31-23-53.eu-west-1.compute.internal ip-172-31-28-
174.eu-west-1.compute.internal ip-172-31-30-211.eu-west-
1.compute.internal ]
OFFLINE: [ ip-172-31-31-24.eu-west-1.compute.internal ]

Full list of resources:

 Resource Group: dispatcher-grp
     aws-eip   (ocf::spitch:eip):  Started ip-172-31-23-53.eu-west-
1.compute.internal
     dispatcher     (ocf::heartbeat:anything):    Started ip-172-
31-23-53.eu-west-1.compute.internal
     asrproxy  (ocf::heartbeat:anything):    Started ip-172-31-23-
53.eu-west-1.compute.internal
 PerformanceTelecom.en-GB.8k  (ocf::heartbeat:anything):    Started
ip-172-31-28-174.eu-west-1.compute.internal
 bss-card.ru-RU.8k  (ocf::heartbeat:anything):    Started ip-172-
31-30-211.eu-west-1.compute.internal
 bss-mortgage.ru-RU.8k   (ocf::heartbeat:anything):    Started ip-
172-31-30-211.eu-west-1.compute.internal
 mobile-bank.en-US.8k    (ocf::heartbeat:anything):    Started ip-
172-31-30-211.eu-west-1.compute.internal
 mobile-bank.ru-RU.8k    (ocf::heartbeat:anything):    Started ip-
172-31-30-211.eu-west-1.compute.internal
 uzh.de-CHZH.8k     (ocf::heartbeat:anything):    Started ip-172-
31-30-211.eu-west-1.compute.internal
 voicebank.en-US.8k (ocf::heartbeat:anything):    Started ip-172-
31-30-211.eu-west-1.compute.internal
 qualtrak-i.en-GB.8k     (ocf::heartbeat:anything):    Started ip-
172-31-30-211.eu-west-1.compute.internal
 aero-flightdays.ru-RU.8k     (ocf::heartbeat:anything):    Started
ip-172-31-30-211.eu-west-1.compute.internal
 aero-flightdays.ru-RU.16k    (ocf::heartbeat:anything):    Started
ip-172-31-30-211.eu-west-1.compute.internal
 claritas.en-US.8k  (ocf::heartbeat:anything):    Started ip-172-
```

```
31-30-211.eu-west-1.compute.internal
 claritas.en-US.16k (ocf::heartbeat:anything):    Started ip-172-
31-30-211.eu-west-1.compute.internal

PCSD Status:
  ip-172-31-23-53.eu-west-1.compute.internal: Online
  ip-172-31-30-211.eu-west-1.compute.internal: Online
  ip-172-31-28-174.eu-west-1.compute.internal: Online
  ip-172-31-31-24.eu-west-1.compute.internal: Offline

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

6.1.2.1.2 Ubuntu distributions

→ install pacemaker on ALL nodes

```
sudo apt-get install corosync pacemaker fence-agents
```

→ setup cluster on ALL nodes

```
sudo cat /etc/corosync/corosync.conf
totem {
    version: 2
    secauth: off
    cluster_name: mycluster

    interface {
            ringnumber: 0
            bindnetaddr: network_id
            mcastport: 5405 #remove if mcast is not supported
            ttl: 1
        }

    transport: udpu
}

nodelist {
    node {
        ring0_addr: node1-IP
        name: hostname1
        nodeid: 1
    }

    node {
```

```
        ring0_addr: node2-IP
        name: hostname2
        nodeid: 2
    }
}

quorum {
    provider: corosync_votequorum
    two_node: 1 #only when using 2 nodes cluster
}
```

[here we have 2 nodes in unicast mode, but it can be extended to N by extending the nodelist{}. Careful that if you do so, the quorum.two_node is not needed]

Note that it is better to use multicast if your network supports it, because when using Unicast the cluster needs to be restarted if you want to add new node (see "adding new node later"). A multicast configuration is identical to the above corosync.conf file dump, changing the udpu into udp, and removing completely the *nodelist {} and quorum {}* block

→ start cluster on ALL nodes

```
sudo cat /etc/default/corosync
# start corosync at boot [yes|no]
START=yes
```

```
sudo service corosync start
sudo service pacemaker start
```

If you don't use fencing:

```
sudo crm configure property stonith-enabled=false
```

If you only have 2 nodes set ignore -quorum property

```
sudo crm configure property no-quorum-policy=ignore
```

At this stage your cluster is up and running, but no resources are started
(check that all nodes appears online doing sudo crm_mon)

6.1.2.2 Deploying Cluster Resources

Depending on your organization, you might use a Proxy server or a MRCP server.

You need to create the HA resources on the Master node for either the MRCP server or the Proxy server.

You also need to create your dispatcher resourcer, and all veryFI and codyFI speech engines resources you want to run in your cluster.

To ease the creation of all those resources in different distribution, spitch provides a deploy script that you may use from one master node:

```
./opt/spitch/ha-devops/install/deploy_speech_cluster.sh
```

The script scans your /opt/spitch directories and deduce the cluster resources from it, so it is critical that /opt/spitch is accurate (be sure you have completed Spitch Software installation).

Before using this script you must open the file and change default values to suit your needs:

```
vi ./opt/spitch/ha-devops/install/deploy_speech_cluster.sh

# NETWORK
PROXY_FLOATING_IP=
PROXY_FLOATING_CIDR_NETMASK=
MRCP_FLOATING_IP=
MRCP_FLOATING_CIDR_NETMASK=

# MONITORING INTERVALS
DEFAULT_MONITOR_SEC=5
MRCP_MONITOR_SEC=
ENGINE_MONITOR_SEC=
PROXY_MONITOR_SEC=
MONITOR_MONITOR_SEC=
DISPATCHER_MONITOR_SEC=

# RESOURCES STICKINESS
# definition: requirement for a resource to not be relocated to
# other node while running. INFINITY means it will NEVER be
# relocated while running. Other values > 0 means resource MAY be
# relocated while running if cluster finds out that it will
# improve overall performance.
DEFAULT_RESOURCE_STICKINESS=500
MRCP_STICKNESS=
PROXY_STICKNESS=
ENGINE_STICKNESS=

# strategy
# possible values : utilization, balanced, minimal
# see pacemaker doc for more explanations
PLACEMENT_STRATEGY=balanced

# TOPOLOGY
# for now you can attached hosts to resources
MRCP_HOSTS=
PROXY_HOSTS=
ENGINE_AVOIDS_ACCESS_HOSTS="-500"
# engines may not run on mrcp/proxy nodes.
# Put -INFINITY if the MAY  is a MUST
ENGINE_AVOID_ACCESS_RESOURCES="-500"
# engines may not runon nodes where MRCP or PROXY is running.
# Use -INFINITY if the MAY is a MUST

# SNMP traps listener IP
```

```
TRAP_LISTENER=
```

After you run the script,

Check that no error has been reported and that ALL resources are STARTED by typing `sudo crm status` (ubuntu) or `sudo pcs status` (rhel/centos)

The script is limited in flexibility but shall provide quiet a generic setup. If you need more flexibility (in term of topology for example), you may want to learn pcs (CentrOS/RHEL) or crm (Ubuntu) CLI tools in order to create a fine tuned setup

6.1.2.3 Configuring Utilization Strategy

Pacemaker offers the possibility to configure a utilization strategy that allows you to define arbitrary utilization criteria for each resource and node.

For example, you configure the utilization strategy based on memory usage, and extend it to CPU and other parameters.

This example is based on the assumption that you have two nodes:

- Node 1 and Node 2
- Node-1 has twice the amount of memory than node-2 (i.e. 32 and 64 MB)

You also have:

- Two CodyFi lingware_A and lingware_B
  - lingware_A loads the lingware model requiring 64 MB
  - lingware_B requires 8 MB

First set node utilization. You must edit the *cib* and insert utilization settings manually

```
sudo cibadmin --query > $HOME/local.xml
cp $HOME/local.xml $HOME/local.xml.backup
vi $HOME/local.xml
```

and add (for each nodes, add this *<utilization></>* block inside a *<node></>* block)

```
<utilization id=”nodes-1-utilization”>
     <nvpair id=”nodes-1-utilization-memory” name=”memory”
value=”64”/>
</utilization>
```

(changing the 64 with the amount of memory the node supports)

Then add codyFi resource utilization.
add (for each resources) under the <primitive id> tag

```
<utilization id=”lingware_A-utilization”>
     <nvpair id=”lingware_A-utilization-memory” name=”memory”
value=”64”/>
</utilization>
```

(where 64 is the lingware_A expected memory usage)

Then apply the change to the cluster

```
sudo cibadmin --replace --xml-file $HOME/local.xml
```

(If you made mistakes you can reload the $HOME/local.xml.backup file into the CIB with the same command)

6.1.2.4 Fencing

Fencing architecture is hardware agnostic but fencing devices and agents are not. We provide some guidelines here for Virtual Box Guest machines because non virtualized setup are too specific. Of course we can assist You if you want to install hardware fencing. There are stable fencing agents for most virtualization technologies (vmware, xen, vbox and so on), and many different agents for non virtualized environment (UPS, PDU, Blade etc). Note that there is a direct consequence on introducing the use of fencing in your cluster setup : quorum won't work in 2 nodes setup, and ignoring quorum is not an option as it could causes stonith death-match (if a split brain occurs – for some reasons they cannot see each other) then both nodes will try to kill the other). To solve this is you can have at least 3 nodes in your cluster (for clusters where only 2 nodes are needed, the third one would just be used to vote -no resources will ever run on it so please create location constraints accordingly). If you cannot have 3 nodes, you must be sure the option two_node is set to 1 in corosync.conf :

```
quorum {
    provider: corosync_votequorum
    two_node: 1
}
```

By default pcs (RHEL/CentOS) will set it to 1 if you setup your cluster with 2 nodes.

→ install Fencing on ALL nodes

Intall spitch-fence package on all node:

```
sudo dpkg/rpm -i spitch-fence
```

At this stage fence_vbox shall be installed in /usr/sbin/fence_vbox.

→ setup cluster for vbox fencing (on MASTER node)

You can run the script

```
./opt/spitch/ha-devops/install/deploy_fence_vbox.sh <id_file>
<hypervisor_ip_addr> <hypervisor_login> <passphrase>
```

NOTE : in this setup we use private keys for authentication and the passwd is the passphrase that protects your private key file locally. This is not the remote pass for logged in user. Be also sure remote user can use VboxManage.

6.1.2.5 Adding and Removing nodes

6.1.2.5.1 CentOS / RHEL distributions

→ Adding a Node to the cluster

To add a node dynamically to your running cluster you must first follow the section 6.1.1 and then perform all "→" actions tagged with "on ALL nodes" of section 6.1.2 (centos setup (if on rhel), pacemaker install) on the NEW node.

Don't forget to also push the *spitch_conf.tgz* you have made on the MASTER to the new NODE. Then perform those commands on the MASTER node to add the node to the existing cluster:

```
sudo pcs cluster auth -h hacluser -p your_password hostnameM
sudo pcs cluster node add hostnameM --start –enable
```

Note

If you are on RHEL / CentOS 6 you MUST RESTART your cluster if it has been configured in Unicast (for multicast setup this is not needed) . To restart do : `sudo pcs cluster stop –all` then `sudo pcs cluster start –all`

Check that your new node is now ONLINE (from MASTER) (along with existing ones)

```
sudo pcs status
```

→ Removing a Node from the cluster

```
sudo pcs cluster node remove <hostname>
```

(change <hostname> with node hostname to remove)

NOTE If you are on RHEL / CentOS 6 and cluster is configured to use Unicast you MUST RESTART your cluster `sudo pcs cluster stop –all` `sudo pcs cluster start --all`

Check that the node does not appear anymore.

```
sudo pcs status
```

6.1.2.5.2 Ubuntu

→ Adding a Node to the cluster

First of all install and configure Spitch Software on the new node (follow 6.1.1). Don't forget to also unpack the configuration tgz (that contains your configurations changes) so that the new node is in sync.

Then install pacemaker & corosync following 6.1.2.1.1.

Now we distinguish 2 cases, if you have have corosync configured for multicast, you just have to copy /etc/corosync/corosync.conf onto your new node, and corosync shall detect it automatically after you start pacemaker/corosync on the new node.

If you are using Unicast you must open `/etc/corosync/corosync.conf` on the master node (at your choice), add the new node in the nodelist{} block of the corosync.conf file, push the changes to all nodes (including the new one), and restart the cluster via :

```
sudo service pacemaker stop
sudo service corosync stop
sudo service corosync start
sudo service pacemaker start
```

Check that your new node is now ONLINE (from MASTER) (along with existing ones)

```
sudo crm status
```

→ Remove Node (remove a node in the cluster)

```
sudo crm node delete <node>
```

NOTE If you are using unicast, it is safer to manually update the `/etc/corosync/corosync.conf` and sync it to all left node, then restart pacemaker/corosync on all nodes.

Check that the node does not appear anymore

```
sudo crm status
```

6.2 Storage Cluster

We illustrate here how to configure a 2 nodes storage cluster setup, running two independent logical "HA Storage system", one for Audio, and one for Voice-prints. (Customer may prefer to run 1 HA Storage system per cluster).

In order to install the storage in your deployment, you must:

- Install the Spitch software for the storage
- Install and configure third-party software (DRBD, OpenRESTY)
- Install and Setup Pacemaker
- Deploy Storage resources

6.2.1 Storage Spitch Software install & Configuration

You shall have been provided with 1 bundle (tgz files) :

- spitch_apps_bundle.<distribution>.tgz : contains a set of packages (debian files for UBUNTU distributions, rpm files for CENTOS/RHEL ones). The following package shall be bundled:
  - spitch-storage
  - spitch-ha-misc (set of tool scripts used by cluster)
  - spitch-ha-resourceagents (custom cluster resource agents OCF scripts)

To install those packages and lingwares, do the following:

→ copy bundle on ALL nodes

scp spitch_apps_bundle.<distributions>.tgz to both master and backup nodes

→ untar bundle on ALL nodes

tar -C /tmp -xzvf spitch_apps_bundle.<distributions>.tgz

→install packages on ALL Nodes

For debian packages do :

```
sudo dpkg -i spitch-storage.deb
sudo dpkg -i spitch-ha-misc.deb
sudo dpkg -i spitch-ha-resourceagents.deb
```

For rpm packages:

```
sudo rpm -i spitch-storage.deb
sudo rpm -i spitch-ha-misc.deb
sudo rpm -i spitch-ha-resourceagents.deb
```

→ create/update REST gateways configuration on all node

The spitch-storage package contains preconfigured REST gateways.

REST Gateways are configured by default to :

• Use ssl, at port 8080 for Audio, 8081 for Voiceprints

• Use self signed certifcate and key located at /opt/spitch/storage//conf/storage.crt|key

• Accept only request from user whose basic authorization user is 'spitch' and password 'spitch'. A hash of the password is stored at /opt/spitch/storage/rest-gw/nginx/conf/htpasswd

• logs are stored in /usr/local/openresty/nginx/logs/

• audio and voiceprints files partitions are mounted respectively on /mnt/audio for audio and

/mnt/voiceprints

Configuration files for those gateways are located in /opt/spitch/storage/rest-gw/nginx/conf/ (nginx.audio.conf and nginx.voiceprints.conf), so if you need to change one of the default settings or other advanced settings please go ahead.

→ Configure logging on ALL nodes

cat /etc/logrotate.d/nginx
/usr/local/openresty/nginx/logs/*audio.log {
daily
missingok
rotate 52
compress
delaycompress
notifempty
create 640 root adm
sharedscripts
postrotate
[! -f /usr/local/openresty/nginx/logs/nginx_audio.pid] ||
kill -USR1 `cat
/usr/local/openresty/nginx/logs/nginx_audio.pid `
endscript
}

/usr/local/openresty/nginx/logs/*voiceprints.log {
daily
missingok
rotate 52
compress
delaycompress
notifempty
create 640 root adm
sharedscripts
postrotate
[! -f /usr/local/openresty/nginx/logs/nginx_voiceprints.pid] ||
kill -USR1 `cat
/usr/local/openresty/nginx/logs/nginx_voiceprints.pid `

```
endscript
}
```

6.2.2 Storage Third Party Software install

6.2.2.1 CentOS / CentOS Distributions

First follow section Pacemaker Install for Redhat 6/7 & CentOS 6/7

Once pacemaker is installed:

→ install DRBD on Both nodes

sudo yum install -y kmod-drbd84 drbd84-utils

→ install OpenRESTY on both nodes

For CentOS create a /etc/yum.repos.d/OpenResty.repo file whose content is :

```
cat /etc/yum.repos.d/OpenResty.repo
[openresty]
name=Official OpenResty Repository
baseurl=https://copr-
be.cloud.fedoraproject.org/results/openresty/openresty/epel-
$releasever-$basearch/
skip_if_unavailable=True
gpgcheck=1
gpgkey=https://copr-
be.cloud.fedoraproject.org/results/openresty/openresty/pubkey.gpg
enabled=1
enabled_metadata=1
sudo yum install openresty
/etc/yum.repos.d/OpenResty.repo
```

For RHEL create a /etc/yum.repos.d/OpenResty.repo file whose content is :

```
cat /etc/yum.repos.d/OpenResty.repo
[openresty]
name=Official OpenResty Repository
baseurl=https://copr-
be.cloud.fedoraproject.org/results/openresty/openresty/epel-
RELEASE-$basearch/
skip_if_unavailable=True
gpgcheck=1
gpgkey=https://copr-
be.cloud.fedoraproject.org/results/openresty/openresty/pubkey.gpg
```

```
enabled=1
enabled_metadata=1
```

(You need to replace RELEASE in the file content above with your RHEL system's major version number, like 5, 6, or 7.)

Then perform

```
sudo yum install openresty
```

to complete the installation of openresty

6.2.2.2 Ubuntu distributions

First follow section Pacemaker Install for Ubuntu

Once pacemaker is installed:

→ install DRBD on Both nodes

```
sudo apt-get install drbd8-utils
```

This is the only thing to do as spitch-storage.deb comes with a compiled version of openresty so if you followed 6.2.1 you shall have openresty installed.

6.2.3 HardDisk and LVM configuration

→ partition HardDisk and setup LVM on both nodes

We assume here we have same disk capacity on both nodes. We also assume the hard disk device

name is /dev/xvdf, and partitions created are named /dev/xvdf1 and /dev/xvdf2

→ Create 2 partitions (not strictly needed, we can put everything on one)

```
sudo parted -a optimal /dev/xvdf mkpart primary 0% 70%
sudo parted -a optimal /dev/xvdf mkpart primary 70% 100%
```

→ Create 2 physical volume, one for each partition

```
sudo pvcreate /dev/xvdf1
sudo pvcreate /dev/xvdf2
```

→ create 2 Audio and Voiceprints volume groups

Each volume group contains the previouysly created PV

```
sudo vgcreate audio_pool /dev/xvdf1
sudo vgcreate voiceprints_pool /dev/xvdf2
```

Note
We could only create one group using both PVs. I guess here that using 2 groups give u a bit of more flexibility in term of using specific PV instead of a shared pool

→ create Audio and Voiceprints LogicalVolume

Audio Logical Volume belong to the audio pool, and Voiceprints Volume to voiceprints pool.

Note
Dont use 100% of group; leave a margin so u can extend etc . In the current example the disk is 1G

sudo lvcreate --name audio_volume --size 650M audio_pool
sudo lvcreate --name voiceprints_volume --size 250M voiceprints_pool

6.2.4 DRBD and Filesystem configuration

If you followed previous steps we should be able to use those LVM logical volume block device for DRBD to operate.

→ create Audio and Voiceprints DRBD resources on ALL nodes

You shall have proper contents for the drbd res files, except for the hostnames and ips. You can change them to match your setup:

```
cat /etc/drbd.d/audiodata.res
resource audiodata {
 protocol C;
 startup {
   wfc-timeout 15;
  degr-wfc-timeout 60;
  }
  net {
    cram-hmac-alg sha1;
    shared-secret "secret";
    after-sb-0pri discard-younger-primary;
    after-sb-1pri discard-secondary;
    after-sb-2pri disconnect;
  }
  on hostname1_fqdn {
    device /dev/drbd0;
    disk /dev/mapper/audio_pool-audio_volume;
    address ip1:7788;
    meta-disk internal;
  }
```

on hostname2_fqdn {

```
    device /dev/drbd0;
    disk /dev/mapper/audio_pool-audio_volume;
    address ip2:7788;
    meta-disk internal;
  }
}
```

(where hostname1_fqdn, ip1 and hostname2_fqdn, ip2 must be replaced by your settings)

```
cat /etc/drbd.d/voiceprintsdata.res
resource voiceprintsdata {
protocol C;
  startup {
    wfc-timeout 15;
    degr-wfc-timeout 60;
  }
  net {
    cram-hmac-alg sha1;
    shared-secret "secret";
    after-sb-0pri discard-younger-primary;
    after-sb-1pri discard-secondary;
    after-sb-2pri disconnect;
  }
  on hostname1_fqdn {
    device /dev/drbd1;
    disk /dev/mapper/voiceprints_pool-voiceprints_volume;
    address ip1:7789;
    meta-disk internal;
  }
  on hostname2_fqdn {
    device /dev/drbd1;
    disk /dev/mapper/voiceprints_pool-voiceprints_volume;
    address ip2:7789;
    meta-disk internal;
  }
}
```

→ bring DRBD resource up, master node

do it for each resource (audio, then voiceprints. here we show voiceprints only

```
sudo drbdadm create-md voiceprintsdata
sudo drbdadm up voiceprintsdata
```

→ bring DRBD resource up, slave node

```
sudo drbdadm create-md voiceprintsdata
sudo drbdadm up voiceprintsdata
```

→ force DRBD primary, master node

```
sudo drbdadm primary --force voiceprintsdata
```

check status

```
cat /proc/drbd
1: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r-----
ns:258004 nr:0 dw:0 dr:258916 al:8 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1
wo:f oos:0
```

You shall see connected and UpToDate, along with Primary/Secondary state

→ Create Filesystem, master

Depending on which Filesystem you choose you may not need to install xfsprogs. In our case example we need it because our fs is xfs.

On centOS/RHEL :

```
sudo yum install xfsprogs
```

On Ubuntu:

```
sudo apt-get install xfsprogs
```

then just create the filesystem:

```
sudo mkfs.xfs /dev/drbd1
```

→ create mounted point for filesystem

```
sudo mkdir /mnt/voiceprints
```

6.2.5 Cluster Resources deployment

You must create DRBD, Filesystem, LVM and REST Gateway resources. You can use the script /opt/spitch/ha-devops/install/deploy_storage.sh instead of manually create all resources.

Before doing so, open the script file and change the default values to match your setup (essentially set the floating Ips correctly)

```
vi /opt/spitch/ha-devops/install/deploy_storage.sh

#NETWORK
AUDIO_FLOATING_IP=
AUDIO_FLOATING_CIDR_NETMASK=
VOICEPRINTS_FLOATING_IP=
VOICEPRINTS_FLOATING_CIDR_NETMASK=

#MONITORING INTERVALS
```

```
DEFAULT_MONITOR_SEC=5
VOICEPRINTS_MONITOR_SEC=
AUDIO_MONITOR_SEC=
```

WARNING: this script makes use of REST Gateway and DRBD configuration files, so be sure you have completed 6.2.1.

7 Application Programming Interface (API)

The Spitch cluster APIs give a layer of coordination between the server and the client.

Spitch provides APIs for speech recognition and speaker verification.

The speech recognition process can be initiated via an HTTP interface or a MRCP interface.

Spitch provides APIs for both the HTTP interface and the MRCP interface.

When using the HTTP interface, the client decides when to start the recognition session and when to terminate the recognition session.

When using the MRCP interface, the MRCP server decides when to start and when to terminate the recognition process.

Speaker Verification can only be accessed via an HTTP interface.

7.1 Speech Recognition HTTP Interface – CodyFi

The speech recognition process can be initiated via an HTTP interface.

Spitch APIs support HTTP POST requests to forward audio to the selected Spitch ASR Engine (server) in chunked mode.

In the HTTP POST request, the HTTP headers specify the recognition parameters grammar and audio format.

The third-party application must be able to:

- Do POST requests.
- Use HTTP transfer encoding to send audio data to the Spitch ASR Engine.

The Spitch CodyFi API employs the headers, as shown in Table 3: HTTP Request Headers

Table 3: HTTP Request Headers

HTTP Header Name	Header Value Definition
Host	This header specifies the grammar name. **E.g. (Host: grYesNo.en-US)**
Content-Type	This header specifies the audio format. The possible values are: • audio/x-pcm; rate=8000 • audio/x-pcm; rate=16000 • audio/ogg; codecs=opus **E.g (Content-Type: audio/x-pcm; rate=8000)**
Transfer-encoding	This tag must contain the type of transfer value. The possible values are: • Chunked **E.g. (Transfer-encoding: Chunked)**
Content-MD5 (Optional)	Base64-encoded binary MD5 sum of the audio payload content. If Content-MD5 is present, the proxy performs the MD5 checksum calculation of the content. The checksum is compared it with the supplied MD5 checksum to ensure the intended data was transferred correctly. **Note** This mode of operation is only applicable to recordings (offline transfers), when the audio content is known before transmission, and not for online transfers where the con tent is sent as it is generated. **E.g. Content-MD5 : Q2hlY2sgSW50ZWdyaXR5IQ==**.

Table 4: HTTPs Proxy Return Messages

Message	Descripton
BAD REQUEST	You receive this message in case of: • A wrong bit rate for pcm stream. • IO error while receiving Audio. • Stereo not supported (Audio must be mono-channel). • No audio received (the body of the post request is empty).
NOT FOUND	You receive this message if: • No suitable Speech Engine was found.

7.1.1 How to Test Spitch HTTP POST Request

To test the POST Request, use the CURL command tool.

1. Open the CURL tool.
2. From the command line, test the POST request using the following command:

```
curl vvv -databinary @<PathtoAudioFileRawData> H "Transfer-Encoding: chunked" H
"ContentType: <audiofileformat>” H "Host:<grammar-name>” “<SpitchASRServer
Address>”
```

For more information on how to install and use the CURL tool, click on:

https://curl.haxx.se/

7.1.2 How to Improve Output Readability

1. Open the CURL tool.
2. From the CURL command line, pipe the result to xmllint to indent the XML lines of the output, using the following command:

```
curl -vvv --data-binary @<Path-to-Audio-File-Raw-Data> -H "Transfer-Encoding:
chunked" -H "Content-Type: <audio-file-format>” -H "Host: <grammar-name>”
“<Spitch-ASR-Server-Address>” | xmllint –format -
```

7.1.3 The HTTP POST Request

You can send a specific request for a specific grammar, for example `grYesNo.en-US,` and for a specific audio codec such as “linear PCM at 8Khz” to the specific HTTP Proxy running on 172.30.30.5 , using the following curl command:

```
curl -vvv --data-binary @<Path-to-Audio-File-Raw-Data> -H "Transfer-Encoding:
chunked" -H  "Content-Type: audio/x-pcm; rate=8000 ” -H "Host: grYesNo.en-US”
“https://172.30.30.5”| xmllint –format -
```

The output of the curl command will result in:

Host: grYesNo.en-US

User-Agent: curl/7.43.0

Accept: */*

Transfer-Encoding: chunked

Content-Type: audio/x-pcm; rate=8000

Expect: 100-continue

Once the command is processed, a recognition result in XML format is returned, see The Speech Recognition Results

7.1.4 The Speech Recognition Results

The ASR Spitch engine returns results using the Natural Language Semantic Markup Language (NLSML), as specified by the W3C.

The speech recognition results depend on:

- The type of grammar.

  LVCSR (Open Grammars) or Closed Grammars
- Whether SignyFi is enabled.

7.1.4.1 Speech Recognition Output XML Elements

The voice recognition output uses XML elements as shown in Table Speech Output Elements.

Table 5: Speech Output XML Elements

7.1.4.2 SWI_ssmMeanings and SWI_ssmConfidences Output Scenarios in SignyFI

The Speech Output XML Elements table shows two elements that concern the Statistical Semantic Model (SSM):

- SWI_ssmMeanings
- SWI_ssmConfidences

These elements only contain meaningful values, if recognition is performed using an LVCSR grammar with SSM, that is, with SignyFi enabled.

SWI_ssmMeanings and SWI_ssmConfidences elements appear in the output, but they will be empty.

If recognition was performed for:

- An LVCSR grammar without SSM the SWI_meaning, SWI_ssmMeanings and SWI_ssmConfidences elements appear in the output, but they will be empty.
- A closed grammar where the GRXML path selected by the decoder contains an SWI_meaning=<meaning> tag, the <meaning> string appears in the SWI_meaning element, and the SWI_ssmMeanings and SWI_ssmConfidences will contain a "fake" 1 element list with this <meaning>, and a "1.0" confidence.
- A closed grammar where the GRXML path selected by the decoder does NOT contain an SWI_meaning tag, the SWI_meaning, SWI_ssmMeanings and SWI_ssmConfidences elements will appear in the output, but they will be empty.

7.1.5 Speech Recognition Output Examples

This section provides you with various output examples depending on the scenarios and the type of grammar used.

To be noted that the Spitch ASR Engine returns all output results on a single line.

To improve output readability, you pipe it to the xmllint command line tool, as shown in How to Improve Output Readability

7.1.5.1 Speech Recognition Output Example – LVCSR

Example of a recognition result output for LVCSR grammar.

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
	<interpretation grammar="session:0x0033b363" confidence="1">
		<input mode="speech">want to know about services</input>
		<instance>
			<dummy confidence="1.0">app-services</dummy>
			<SWI_literal>want to know about services</SWI_literal>
			<SWI_spoken>want to know about services</SWI_spoken>
			<SWI_meaning>app-services</SWI_meaning>
			<SWI_ssmMeanings><dummy>app-services::app-
detalisation</dummy></SWI_ssmMeanings>

		<SWI_ssmConfidences><dummy>1.70785::0.917104</dummy></SWI_ssmConfidences>
	</interpretation>
</result>
```

7.1.5.2 Speech Recognition Output Example – Closed Grammar

Example of a recognition output for a closed grammar.

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
	<interpretation grammar="session:0x0033b363" confidence="1">
		<input mode="speech">Yes</input>
		<instance>
			<dummy confidence="1.0">Yes</dummy>
			<SWI_literal>Yes</SWI_literal>
			<SWI_spoken>Yes</SWI_spoken>
			<SWI_meaning>Yes</SWI_meaning>
			<SWI_ssmMeanings><dummy>Yes</dummy></SWI_ssmMeanings>
			<SWI_ssmConfidences><dummy>1.0</dummy></SWI_ssmConfidences>
		</instance>
	</interpretation>
</result>
```

7.1.5.3 Speech Recognition Output Example – No-Match Result

Voice recognition can lead to a “no-match” output result when:

- Using a closed grammar and the Spitch ASR Engine cannot find a result in the user’s input.
- Using an LVCSR grammar and there is no recognizable audio in the user’s input.

Example of a recognition output for a no-match result.

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
        <interpretation>
                <input mode="speech">
                        <nomatch />
                </input>
                </instance>
        </interpretation>
</result>
```

7.2 Speech Recognition MRCP Interface – CodyFi

The speech recognition process is initiated via a MRCP server.

Typically, a PBX system sends a request to the MRCP server on the Spitch framework.

Spitch APIs support SIP, MRCP and RTP requests to forward audio to the selected Spitch ASR Engine (server) in chunked mode.

Both the MRCPv2 messages and the XML responses conform the RFC 6787.

For more information about the RFC standards and messages, see:

https://tools.ietf.org/html/rfc6787

and

MRCP Server section.

Table 6: Supported MRCPv2 Messages and Headers

Incoming Messages	Headers
SET-PARAMS	• No-Input-Timeout • Recognition-Timeout • Speech-Incomplete-Timeout • Save-Waveform
DEFINE-GRAMMAR	• Content-ID
RECOGNIZE	• Start-Input-Timers • No-Input-timeout • Speech-Incomplete-Timeout • Save-Waveform
START-INPUT-TIMERS	
STOP	
Outgoing Messages	**Headers**
START-OF-INPUT	
RECOGNITION-COMPLETE	• Content-Type application/x-nlsml; charset=\"utf-8\" • Completion-Cause ◦ 000 Success ◦ 001 No Match ◦ 002 No Input ◦ 008 Error • Waveform-URI http://spitch.ch/asr-recordings/

Note

GET-PARAMS is not supported.

Supported MRCPv2 Message Bodies

The Spitch MRCP server does not support XML defined grammars.

All the grammars are pre-compiled and made available via the built-in syntax.

MRCPv2 Message	Message Bodies
DEFINE-GRAMMAR	• Built-in <grammar spec>
RECOGNIZE	• Built-in <grammar spec> • Session:<ID> ID refers to the content-id of a grammar previously defined with the DEFINE-GRAMMAR

7.3 Storage REST API

NGINX can access the file system, and it can be configured to support HTTP basic authentication over an SSL through the `auth_basic_user_file.`

7.3.1 Uploading an Audio File

Use a HTTP PUT request to upload an audio file along with its meta-data.

Metadata must have the format:

`x-spitch-meta-yourmetadatatag`

Accepted file types are .opus, .ogg, .wav, .pcm and .raw.

7.3.1.1 PUT Request Example

PUT file.wav HTTP/1.1

```
Host: floating_ip or DNS host name
Authorization: Basic base64pass
x-spitch-meta-param1: value1
x-spitch-meta-param2: value2
Content-Type: audio/wav
Content-Length: 22

file_content
```

Response code:

HTTP 1/1 200 OK

Note

If file_content is empty and a local file was already created, only the metadata is updated,else an empty local file is created and meta-data attached to it.

7.3.2 Downloading an Audio File

Use a HTTP GET request to download an audio file along with its meta-data.

7.3.2.1 GET Request Example

GET file.wav HTTP/1.1

```
Host: floating_ip or DNS host name
Authorization: Basic base64pass
Host: wavs.spitch.ch
```

Response code:

HTTP 200 OK

```
x-spitch-meta-param1: value1
x-spitch-meta-param2: value2
Content-Type: audio/wav
Content-Length: 22
```

file_content

Note

By default, when you specify the file.wav in the GET request, the body always contains wav audio data.

If file.wav is not specified in the GET request, the response body will contain a text-formatted list of all files on the storage device.

7.3.3 Deleting an Audio File and its Meta-data

Use a HTTP DELETE request to delete an audio file and its meta-data.

7.3.3.1 DELETE Request Example

DELETE file.wav HTTP/1.1

```
Authorization: Basic base64pass
Host: floating_ip or DNS host name
```

HTTP 200 OK

7.4 Speaker Verification HTTP Interface – VeryFi

The speaker verification process is conducted via an HTTP interface.

Spitch APIs support HTTP POST requests to forward audio files to the selected Spitch SV Engine (server) in chunked mode.

7.4.1 The HTTP POST Request

The SV speech engine receives HTTP POST requests from the HTTPS Proxy to:

- Create (enrol) voice-prints
- Update voice-prints
- Verify voice-prints
- Verify a given segment against a given voice-print.

The SV engine provides an XML response to the client request. The response contains the operation results or the verification score.

In the HTTP POST request, the HTTP headers specify the verification parameters of the audio format.

In the same way as for the ASR Engine API, the SV Engine cluster:

- Loads a configuration file in a XML format.
- Processes HTTP POST requests.
- Returns an XML response.

The third-party client must be able to:

- Do POST requests.
- Use HTTP chunked transfer encoding to send audio data to the Spitch SV Engine.

The Spitch VeryFi API employs the headers as shown in Table 7: Verification HTTP Request Headers

Table 7: Verification HTTP Request Headers

HTTP Header Name	Header Value Definition
Host	Verification.* This header includes the application name and it is used to route the request to the appropriate SV engine, for example **`sv_engine1.en-US`**
Enroll	The Enroll header possible values are: • false • true • update • delete The default value of this header is <false>.
Gender	The Gender header possible values are: • male • female The default value of this header is <male>
Speaker-Id	This Header contains the speaker id.
Content-Type	This header specifies the audio format. **E.g (Content-Type: audio/x-pcm; rate= <8000|16000>)**
Transfer-encoding	This tag must contain the type of transfer value. The possible values are: • Chunked **E.g. (Transfer-encoding: Chunked)**

What Is Enrolment?

Enrolment is the process during which a speaker voice is added to the voiceprints database.

Enrolment is a prerequisite for the speaker verification process since it allows the Spitch SV system to compare the speaker voice against his or her voice print.

7.4.2 The Enrolment Response Format

Based on the value of the Enroll header, four types of response are allowed:

- Enroll header = true

  The back-end performs an enrolment for the user specified in the header “Speaker-Id”.

- Enroll header = false

  The back-end evaluates the speaker voice against his or her voice print, based on the value of the Speaker-ID header.

- Enroll header = update

  The speaker voice, whose id is specified in the Speaker-ID header, is updated.

- Enroll header = delete

  The speaker voice print is removed from the voice-prints storage.

Table 8: Enrolment Header Responses

Enroll Header Value	Scenario	SYSTEM Response
TRUE	• Less than 60 seconds of non-silence data, i.e. continuous speech, is collected.	• NOT ENOUGH DATA
	• More than 60 seconds of non-silence data, i.e. continuous speech, is collected.	• ENROLLED WITH SUCCESS OR ENROLLED FAILED In case of an internal error.
	• Enroll request after a user has already been enrolled.	• HTTP 400 Bad Request : “Bad parameters”
FALSE	• The speaker voice has been enrolled. The back-end evaluates the audio data against the stored voice print and returns a normalized score . Scores higher than 0.5 mean that the caller was verified. The higher the score, the higher the confidence.	• A score between 0 and 1.
	• The speaker voice has not been enrolled. An evaluation request results in an error.	• HTTP 400 Bad Request : “Bad parameters”
UPDATE	• The speaker voice is enrolled. New data is appended to the previous enrolment data, and a new voice print is created.	• VOICEPRINT UPDATED WITH SUCCESS OR UPDATE FAILED In case of an internal error.
	• The speaker voice is not enrolled.	• HTTP 400 Bad Request : “Bad parameters”
DELETE	• If the speaker voice is enrolled, the system deletes the voice-print.	• VOICEPRINT DELETED
	• If the user is not enrolled, the system returns an error	• HTTP 400 BAD REQUEST: 'Bad parameters'

7.4.3 Speaker Verification Output

The output is dependant on the operation performed (enrolment, voice-print update, voice-print deletion, or speaker verification), and it is based on the Request's Enroll headers.

As shown in the Table Enrolment Headers Responses, four values are possible:

- true
- false
- update
- delete

This section provides examples for the enroll/verify/voice-print update/voice-print delete operations, and include for each operation the relevant curl command and the corresponding XML output.

Note

The examples assume that:

- An HTTP Proxy is running on **172.30.30.5**
- A speech engine named "sv_backen1.en-US" runs in the Spitch cluster on port 3857.
- All operations are performed for speaker-id "joe", with "male" gender.

7.4.3.1 Enroll (Header Enroll:True)

CURL COMMAND:

```
curl -vvv -s --data-binary @/tmp/audio_1.raw -H "Transfer-Encoding: chunked" -H
"Content-Type: audio/x-pcm; rate=8000" -H "Speaker-Id: joe" -H "Enroll: true"-H
"Gender: male" H "Host: sv_backend1.en-US-
"http://172.30.30.5/"
```

Possible Responses with Enrol Header Set to True:

1. If the aggregate duration of all the received audio data in this and previous enroll commands is less 60 seconds, the response "NOT ENOUGH DATA" is returned.

   Note

   The duration count of the audio excludes moments of silence, which means that there has to be 60 seconds of utterances for the verification process to occur.

   ```
   <?xml version="1.0"?>
   <result>Total seconds for speaker: 13.94
   NOT ENOUGH DATA
   </result>
   ```

2. If aggregate duration exceeds 60 seconds for the fist time, the user is enrolled, that is the voiceprint is created, and the response “Enrolled with success!” is returned.

```
<?xml version="1.0"?>
<result>Total seconds for speaker: 69.29
Enrolled with success!
</result>
```

3. If the user is already enrolled, an error response is returned showing:
   - The HTTP status code “400 Bad request”
   - The body comprising the text “Bad Parameters”.

7.4.3.2 Verify (Header Enroll: False)

CURL COMMAND:

```
curl -vvv -s --data-binary @/tmp/audio_2.raw pml6Cqu1 -H "Transfer-Encoding: chunked"
-H "Content-Type: audio/x-pcm; rate=8000" -H "Speaker-Id: joe" -H "Enroll: false" -H
"Gender: male" -H "Host: sv-backend1.en-US" "https://172.30.30.5/"
```

Possible Responses with Enroll Header Set to False:

If the user is correctly enrolled, a verification score between 0 and 1 is calculated and returned.

- A score greater than 0.5 indicates that the audio data is likely to correspond to the enrolled user, and that the speaker should be deemed verified.

```
<?xml version="1.0"?>
<result>0.9286</result>
```

- A score lower then 0.5 indicates that the speaker is not verified.

7.4.3.3 Update (Header Enroll: Update)

CURL COMMAND:

```
curl -vvv -s --data-binary @/tmp/audio_3.raw -H "Transfer-Encoding: chunked" -H
"Content-Type: audio/x-pcm; rate=8000" -H "Speaker-Id: joe" -H "Enroll: update" -H
"Gender: male" -H "Host: sv-backend1.en-US" "https://172.30.30.5/"
```

Possible Responses with Enroll Header Set to Update

If the user is correctly enrolled, the audio data used for the voice-print creation is aggregated to the new audio data sent in the request, and a new voice-print is created using all the aggregated data.

<?xml version="1.0"?><result>Total seconds for speaker: 121.27
Voiceprint updated with success!
</result>

If user is not enrolled, an error response is returned:

"Bad parameters"

7.4.3.4 Delete (Header Enroll: Delete)

CURL COMMAND:

```
curl -vvv -s --data "" -H "Speaker-Id: joe" -H "Enroll: delete" -H "Gender: male" -H
"Host: sv-backend1.en-US" "https://172.30.30.5/"
```

Possible Responses with Enroll Header Set to Dlelete

If the user is enrolled, the voice-print is deleted.

<?xml version="1.0"?><result>Total seconds for speaker: 121.27
Voiceprint deleted!
</result>

If user is not enrolled, an error response is returned:

"Bad parameters"

7.5 Setting up SnmpStatsMonitor

You set up the full-cluster monitoring via the Monitoring API for the SNMP protocol.

For more information on monitoring, see SnmpStats Monitor

In addition, the MIB files offer a complete description on how to retrieve and interpret the Spitch statistics and the Pacemaker notifications. This description is found in:

- SPITCH-MIB file

  and

- PCMK-MIB.txt, file

  The events that the cluster will notify are defined in:

  https://github.com/ClusterLabs/pacemaker/blob/master/extra/PCMK-MIB.txt

SNMP Cluster monitoring is activated through a Cluster Stats Monitor clone agent and a ClusterMon clone.

7.5.1 How to Create an SnmpStatsMonitor Instance

To instantiate the SnmpStatsMonitor agent on all the nodes of the cluster, you create a Pacemaker Clone resource.

Each Spitch process that runs on a node sends its statistics to the local Cluster Monitor Snmp Agent process.

7.5.2 How to Access the SnmpStatsMonitor

You access SnmpStatsMonitor via an SNMP front-end client from where you can retrieve standard MIB-2 stats or Spitch statistics.

The statistics that you can view are those implemented in the MIB file.

8 Analytical Index

Alphabetical Index

Contact Information

Contact Information:

Zürich Office

Kreuzstrasse 54,
8008 Zürich,
Switzerland
+ 41 445 42 82 66

London Office

3 More London Riverside,
London SE1 2RE,
United Kingdom
+ 44 20 3283 4343

Milan Office

Via Torino 2,
20123 Milano,
Italy
+ 39 02 725 467 70

Spitch website: http://www.spitch.ch/contact/

Mail: support@spitch.ch