SPITCH

Spitch Lingware

User Guide

Enterprise Solutions

Driven by Voice

Publication Date: 30/11/2016
Document Name: Spitch Lingware Builder User Guide
Document Version: 1.0

Published by: Spitch Solutions AG

Table of Contents

1 Before You Start

1.1 Introduction

This user guide shows how to use Spitch Lingware Builder, an application built with the purpose to help you to train and deploy your Spoken Language Technology (SLT).

Spitch Lingware Builder uses Git repositories to manage version control; Git repositories are hosted in Bitbucket.

Bitbucket and Git repository documentation is outside the scope of this document.

1.2 Audience

This user guide is intended for developers, speech technology professionals and other IT professionals with basic knowledge in computer science and system administration.

1.3 Prerequisites

You need to be familiar with the following technologies:

- Scripting languages
- System administration
- Web services
- JSON file

1.4 Objectives

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

- Use Spitch Lingware Builder to create, configure a Lingware node.
- Login into a Lingware node.
- Create, access and deploy a bespoken Lingware application.
- Link your Lingware node to your Bitbucket account.
- Create and edit the JSON configuration file.
- Modify your data resource.
- Select the grammar for your Lingware application.

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 verifications 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 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.

2.2.2 What Is VeryFi?

VeryFi is the Voice Biometrics Platform and enables speaker verification and 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.

2.2.3 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.

2.2.4 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 (backend) to provide speech recognition.

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

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

2.3 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 Speaker Verification

3 Introduction to Spitch Lingware Builder

3.1 Overview

Spitch Lingware Builder allows you to train, tune and deploy your own Spoken language Technology (SLT).

The Spitch Lingware Builder relies on the Spitch Lingware Node.

The Lingware Node is an autonomous, fully provisioned virtual machine where speech-related applications run.

Spitch Lingware Builder is one of the applications running on the Lingware node.

Lingware Builder also uses a git repository to help you manage version control and Bitbucket, which hosts the git repository.

For more information on the Lingware node, see Error: Reference source not found

The Spitch Lingware builder diagram shows the interaction among the Lingware Builder key components.

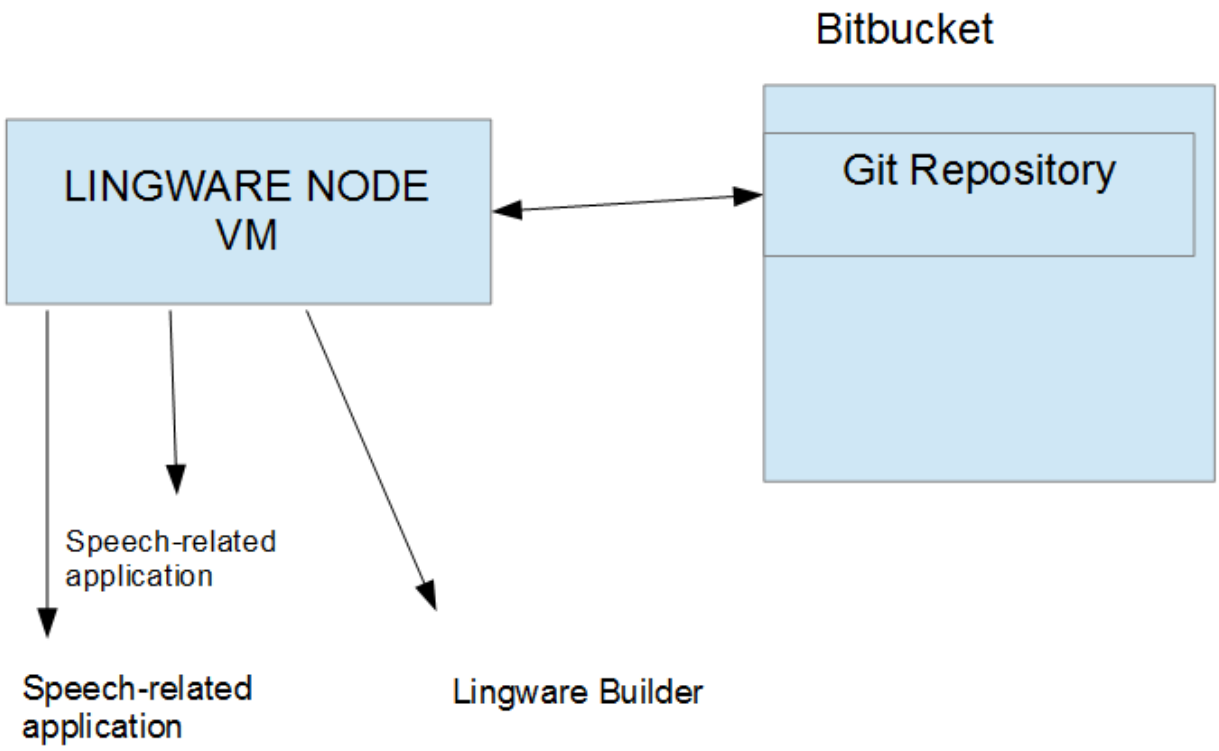

Illustration 1 Architecture

3.2 Prerequisites

To create your Lingware application, you require:

- Browser

  Choose from:

  - Google Chrome
  - Firefox
  - Opera

- Bitbucket

  Bitbucket is an online service that hosts the git repository.

  For more information, see:

  https://bitbucket.org/

- Git repository

  The git repository is a software revision control system that tracks development changes.

  For more information, see:

  https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

4 Using Spitch Lingware Builder

This section provides a reference on how to use Spitch Lingware Builder.

To deploy your ASR and SV technologies, you need to use a Lingware node, and Lingware builder allows you to configure, test and access the application that you have created.

You create a node through the AWS console.

To access the AWS console, refer to the AWS documentation:

https://aws.amazon.com/console/

When you have ensured that AWS has created the node, you can then configure it and test it through Spitch Lingware builder.

The steps that are required to build a node are as follows:

- Create a Lingware node

  How to Create a Lingware Node
- Configure the Lingware node

  How to Configure a Lingware Node
- Login in the new Lingware node

  How to Log into a Lingware Node
- Test the first Lingware build

  How to Test the First Lingware Build
- Package and deploy your application build

  How to Package and Deploy Your Lingware Application Build

For increased versatility and power, you can build your own LVCSR and GRXML using your own data in more than 10 languages. For more information, see The Lingware Configuration Objects

4.1 What Is a Lingware Node?

A Spitch Lingware Node is a fully provisioned, autonomous virtual machine-based, which can train, compile, tune, and deploy your Spoken Language Technology (SLT) applications on demand.

You launch the Lingware node from the AWS console.

The Spitch Lingware node uses a git repository to manage revision control and trigger automatic builds in remote.

The git repository is hosted and accessed through Bitbucket.

Spitch Lingware Nodes are based on the Strider -CD continuous development project.

For more information on Strider-CD, see:

https://strider-cd.github.io/

4.2 Spitch Lingware Builder Flow

The Lingware Node Flow diagram provides a high-level overview of the steps that you need to undertake to create your Lingware application.

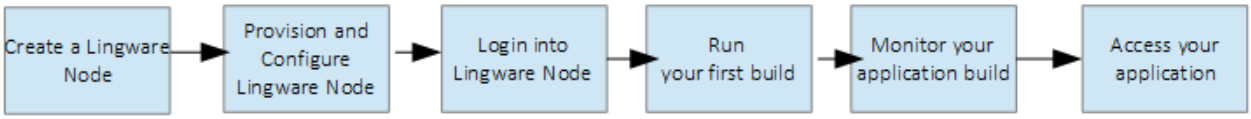

Illustration 2: The Lingware Node

Create a Lingware Node

The first step in deploying your application is creating a Lingware node that can be used for all your building, tuning, packaging and deployment work.

You can do this from your AWS console.

Provision and Configure the Lingware node

Select an instance type for your node that has sufficient RAM and compute power to build your target application. Configure it with the spitch-lingware-node security group.

Login into the Lingware Node

You can access the Lingware node via the external IP and login with one of the default user accounts.

Run your first build

When you have selected the grxml branch, you can trigger your first build.

Monitor your application build

The build page streams the console output from the build to your screen in real-time.

Access your application

You can now access your application using the Spitch ASR client and using the port and IP information contained in the JSON file.

4.3 How to Create a Lingware Node

1. In the AWS console, select the Actions tab.
2. Select AMIs.
3. In the Actions tab, click on the lingware-node-based button, as shown in Illustration 3 Creating a Lingware Node

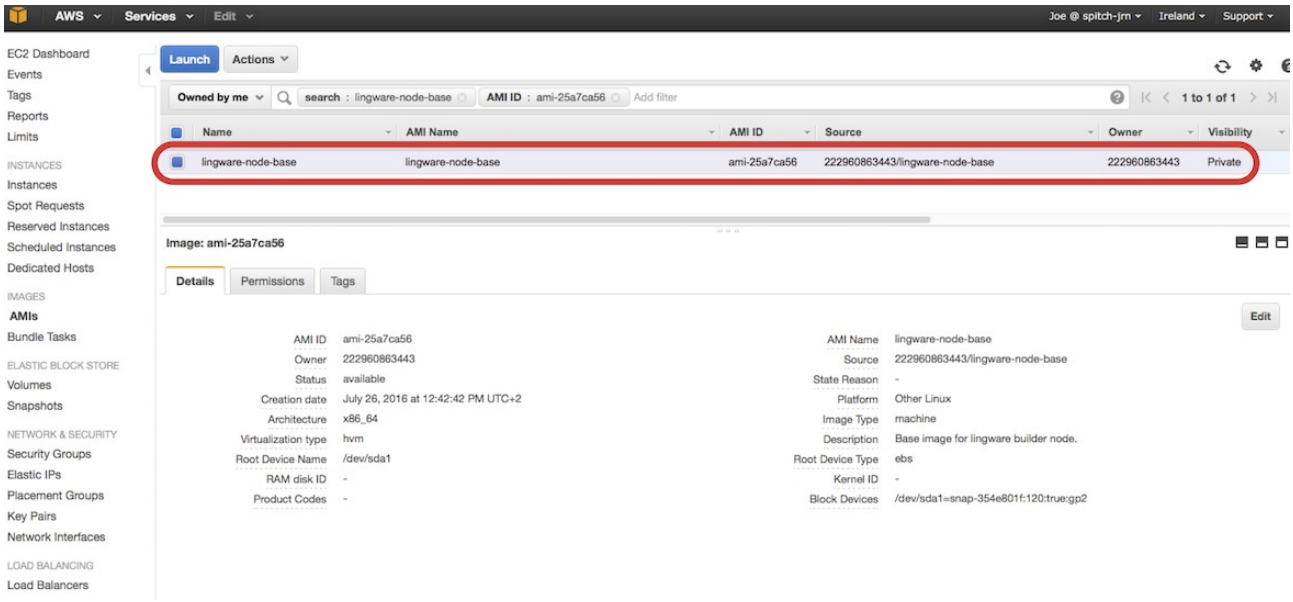

Illustration 3 Creating a Lingware Node

4.4 How to Configure a Lingware Node

1. In the ASW console, choose AMI.
2. Click on Choose Instance type to select an instance type.

   The instance must have sufficient RAM and compute power to build the target application.

   The selected instance type,be must configured with the `spitch-lingware-node` security group.
3. Click on Configure Instance.
4. Click on Add storage.
5. Click on Configure security group.

   AWS automatically assigns a new external IP address to the new node, and the server will select this IP address.

Note

Spitch recommends to provide a tune block in the configuration file, the node automatically tunes your application for speed and accuracy.

4.5 How to Log into a Lingware Node

Note

Before logging into the new node, ensure that the AWS console confirms that the node has booted and passed all checks

1. Access the Spitch Lingware Builder.
2. Login using one of the default accounts.

   To access the new node, use the external IP and one of the defaults user accounts.

   After you have logged in the Spitch Lingware Dashboard is displayed as shown in Illustration 4 The Spitch Lingware Builder Dashboard

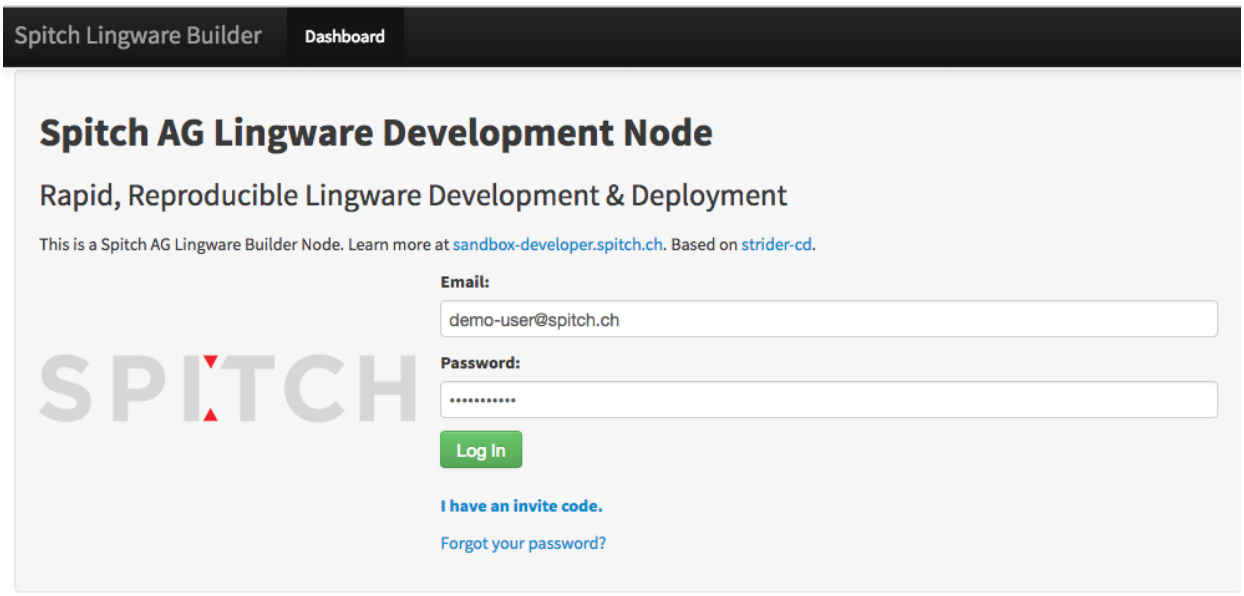

Illustration 4 The Spitch Lingware Builder Dashboard

4.6 How to Test the First Lingware Build

1. In the Spitch Lingware Builder, click on Projects to add the lingware-app-eg project.
   Use the default project to run your first build.
2. Click on Add.
3. Click on Spitch.
4. Click on Configure.
   The Configuration pane is displayed.
5. In the Project menu, click on Branches.
6. Add the `grxml` branch.
7. Select the grxml branch from the drop-down menu.
8. Click on Test to trigger the first build.

The build page streams the console output from the build to your screen in real-time.

When the build completes successfully, the node will start serving your application.

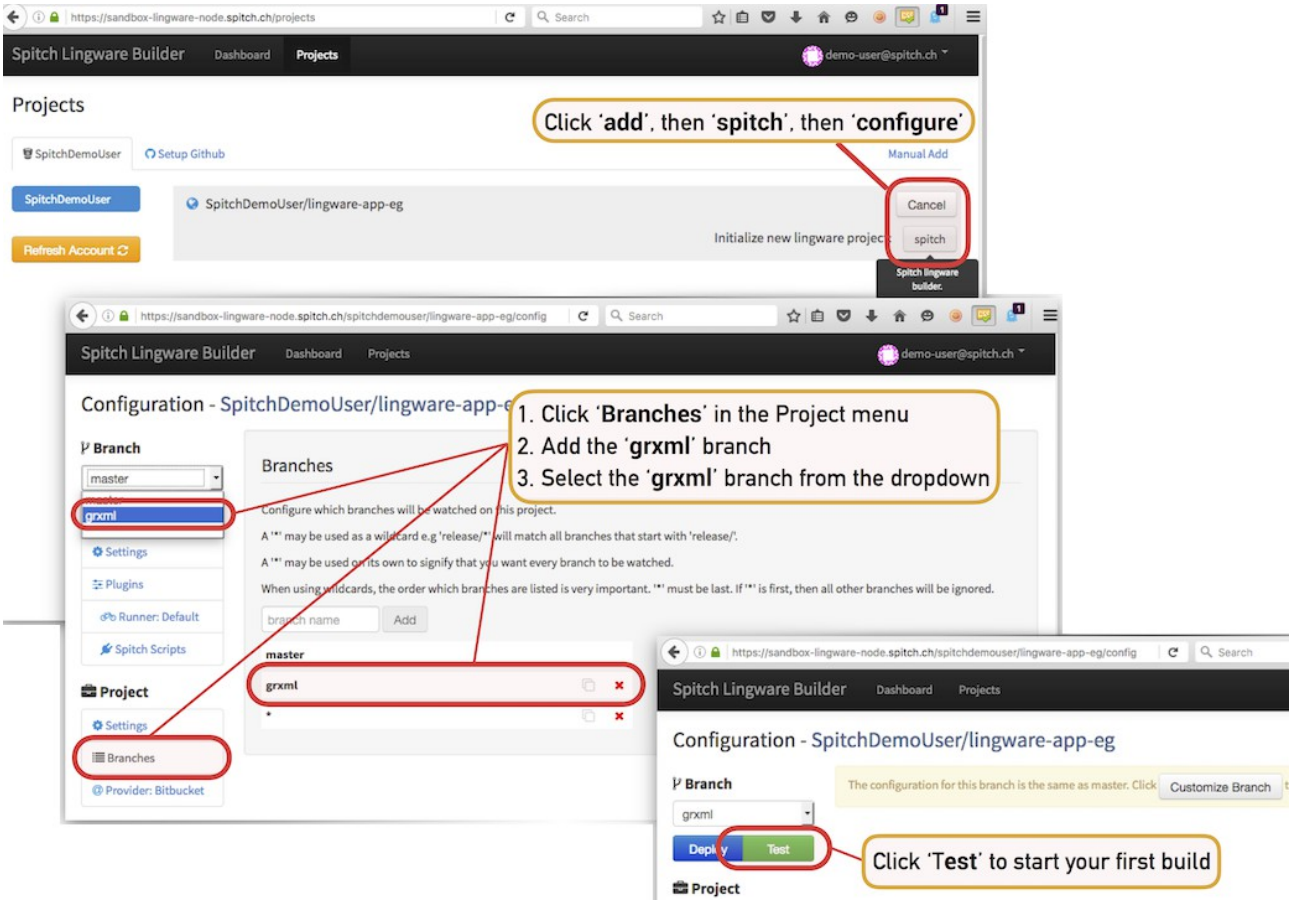

Illustration 5 Running the first builld

4.7 How to Package and Deploy Your Lingware Application Build

1. Click on Deploy to package and deploy your application to Amazon Simple Storage Services (S3).
   Clicking on Deploy runs again the build and the test phase.

5 Creating Your Lingware Application

Each Lingware application must be defined within the context of a git repository so to allow you to:

- Accommodate changes to the resources used to train your model.
- Extend the text set to optimize the speaker's recognition system.
- Change your strategy.

  For example, you might prioritize response speed over accuracy or viceversa.

All the changes that you may want to adopt are implemented in the configuration file and managed through the git repository.

The configuration file is stored in the git repository, which is hosted by Bitbucket.

Because Bitbucket hosts the git repository, you can link the Spitch Lingware node to your Bitbucket account through the Lingware application.

When you link Bitbucket with the Lingware node, the history of the changes that are made to the configuration file are stored in Bitbucket.

The high-level steps required to create your Lingware application are as follows:

- Link your Lingware Node with your Bitbucket account.

  For more information, see How to Link the Spitch Lingware Node to Your Bitbucket Account
- Create the JSON configuration file.

  For more information, see How to Create the Configuration File
- Commit and push the changes to your repository.

  How to Commit and Push the Application to the Repository
- Add the new application to the Lingware node.

  How to Package and Deploy Your Lingware Application Build

5.1 How to Link the Spitch Lingware Node to Your Bitbucket Account

1. In your Lingware node account page, click on Bitbucket.
2. Click on Add Account to start the account linking process.

   The Confirm Access to your account pop-up window is displayed.
3. Click on Grant access to provide your Lingware node with access to your Bitbucket account and all its linked repositories.

   The Lingware node account page is displayed.

   Note

   You may need to unlink all Bitbucket accounts with live projects before you delete them.
4. Click on Refresh Account to display your linked Bitbucket account in your Project page.

Note

Each time that you build a new application, you must create a repository and commit the repository to your Bitbucket account.

5.2 How to Add a New Application Repository to an Existing Account

1. In your Lingware node account page, check that you have linked your Bitbucket account with your Spitch Lingware node.

   Your Bitbucket account should be displayed in the Project page.

2. Follow the Bitbucket procedure to create a repository.

   For more information, see https://bitbucket.org/product

3. In Bitbucket, click on Create repository.

   To initialize your repository, follow the I am starting from scratch direction.

5.3 How to Create the Configuration File

1. In your new git repository, create a file called lingware.json.
2. Add content to the file as follows:

   Always ensure to select the appropriate grammar for the application that you want to create.

```
{
  "topic": "cooking-app",
  "language": "en-US",
  "srate": "8000",
  "port": "10010",
  "description": "Some simple cooking-related voice commands.",
  "acoustic-model": {
    "s3path": "s3://spitch-data/Processed/en-US/AcousticModels/clean-speech_en-US_8KHz"
  },
  "grxml": {
    "s3path": "s3://spitch-data/Processed/en-US/GRXML/demo-grammar.grxml"
  }
}
```

This file provides a basic description for the voice commands interface of the application called “cooking-app”.

Note

The configuration file shown in the example uses the GRXML grammar.

5.4 How to Commit and Push the Application to the Repository

1. In the git repository, enter the following to commit the application:

```
$ emacs lingware.json
$ git add .
$ git commit
[master (root-commit) 00ab555] First commit for cooking app!
 1 file changed, 13 insertions(+)
 create mode 100644 lingware.json
$ git push origin master
Password for 'https://SpitchAwesomeUser@bitbucket.org':
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 457 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://SpitchAdmin@bitbucket.org/SpitchAdmin/cooking-app.git
  * [new branch]      master -> master
$
```

2. Return to the Spitch Lingware node.
3. In the Project page, click on Refresh Account.

   Your new application is displayed.
4. Click on Add to add the new application to the Lingware node.

   Wait until your application is displayed in the Project page.
5. Click on Test.

   When you click on Test, you effectively build and deploy your application.

The application is now accessible using the parameters as specified in the lingware.json configuration file.

5.5 Selecting the Grammar for Your Application

When you select the grammar for creating your language application, ensure that the grammar that you choose matches the content of the audio files that it is intended for.

5.5.1 Mind the Headers!

If accessing the new application, you see an unexpected output, check the output headers against the results to verify that you have selected the appropriate grammar.

In the example, the data-binary header is set to `@yo-spotify-play-top-fifty.wav,` whilst the results list parameters such as Employment status, Civil status and so on.

This mismatch indicates that the incorrect grammar for the intended application has been selected.

When this happens you need to create a new resource and rebuild the application.

OUTPUT EXAMPLE

```
$ curl -s \
> --data-binary @yo-spotify-play-top-fifty.wav \
> -H "Transfer-Encoding:chunked" \
> -H "Content-Type: audio/x-pcm; rate=8000" \
> -H "Host:cooking-app.en-US.8k" \
> "http://sandbox-lingware-node.spitch.ch:10010" \
> | xmllint --format -
 <?xml version="1.0" encoding="UTF-8"?>
<result>
   <interpretation grammar="session:request1@form-level.store" confidence="1">
      <input mode="speech">CIVIL_STATUS:divorced:divorced
EMPLOYMENT_STATUS:unemployed:unemployed</input>
      <instance>
         <dummy confidence="1.0"/>
         <SWI_literal>CIVIL_STATUS:divorced:divorced
EMPLOYMENT_STATUS:unemployed:unemployed</SWI_literal>
         <SWI_spoken>CIVIL_STATUS:divorced:divorced
EMPLOYMENT_STATUS:unemployed:unemployed</SWI_spoken>
         <SWI_meaning/>
```

5.5.2 How to Modify the Data Resource (GRXML, Corpus or NGram)

1. Create a data resource for the cooking-app.grxml, in this instance a GRXML resource, and add it to S3.

   At the prompt, use the command:
   ```
   $ s3cmd put cooking-app.grxml s3://spitch-data/Processed/en-US/GRXML/
   ```

   To view the example application go to:

   https://s3-eu-west-1.amazonaws.com/spitch-data/Processed/en-US/GRXML/cooking-app.grxml

2. Update your JSON configuration file so that it points at the new grxml application (cooking-app).

```
    {
  "topic": "cooking-app",
  "language": "en-US",
  "srate": "8000",
  "port": "10010",
  "description": "Some simple cooking-related voice commands.",
  "acoustic-model": {
    "s3path": "s3://spitch-data/Processed/en-US/AcousticModels/clean-speech_en-US_8KHz"
  },
  "grxml": {
    "s3path": "s3://spitch-data/Processed/en-US/GRXML/cooking-app.grxml"
  }
}
```

3. Add, commit and push the changes as shown in How to Commit and Push the Application to the Repository .

Return to the Spitch Lingware node.

4. In the Project page, select the application for which the resource was created, in this instance cooking-app.
5. Click on Test in the configuration page to rebuild and redeploy the application.

6 The Lingware Configuration Objects

6.1 The Lingware JSON Objects

Each Spitch Lingware Application is specified via the JSO+N format object, and it is stored in a git repository that can be accessed through a Spitch Lingware node.

The JSON object is managed via the git repository to change one or more values for the parameters.

The Lingware JSON objects are:

- NGram
- Text corpora
- GRXML

6.2 Configuring the JSON Objects

Each Lingware application must be configured using a JSON object where the properties of the target application are defined.

When you build an application, this must be stored in a repository with the name `lingware.json`.

Your application can be configured to use one of the following JSON objects:

- NGram

  For an example on how to use the NGram object, see The NGram JSON Object - Example

- Corpora

  For an example on how to use the corpora object, see The Text Corpora JSON Object - Example

- GRXML

  For an example on how to use the GRXML object, see The GRXML JSON Object - ExampleThe GRXML JSON Object - Example

Note

You can edit the JSON configuration file either in Bitbucket or locally.

6.3 The Lingware Application Configuration Parameters

Spitch Lingware Application is defined through a set of parameters as shown in Table 1: Lingware Configuration Parameters.

Parameters that you may want to change include:

Order:

A parameter that provides a value for determining the probability for a given word to occur.

Lambda:

A parameter that provides a value for determining the interpolation.

The `tune` parameter is optional. As shown in Table 2: The Tune Object Configuration Parameters, Spitch recommends that you configure this parameter since it allows you to specify:

- The parameter ranges to search
- A test/tuning set
- An objective to target the application tuning

When you have made the required changes, you commit the changes and push them to the repository.

Bitbucket is automatically updated with the changes.

Table 1: Lingware Configuration Parameters

Parameter	Type	Description	Required
Topic	string	The target name for the application. This will be used to specify the model in requests, and form the prefix for the packaging identifier prefix	Yes
Language	String	The language-locale shorthand that describes the target language and locale of the application. Currently the list of supported language-locale combinations is: en-US, de-CHZH, de-DE,en-GB, es-ES,fr-FR,it-IT, and ru-RU.	Yes
Srate	string	The target sample rate of the application. Note: This must match the sample rate of the value supplied for the acoustic-model parameter.	Yes
Port	string	The target port for the application. Affects configuration, deployment and local instance.	Yes
Description	string	An optional description of the application.	No
Acoustic-model	object	The acoustic-model specification. This is a sub-object, see the associated specification below. Expert parameters are also available. **Value** **Description** s3path The s3 path to the acoustic model resource.	

<table>
<tr>
<td>grxml|corpora|
language-
models</td>
<td>object</td>
<td>The type of text data that the application targets.
The corpora and language-models parameters may be combined.
The grxml parameter should only be used in isolation.
grxml [object]
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>s3path</td><td>The s3 path to the GRXML grammar definition.
This is a text format file, ending in .grxml.</td></tr>
</table>
If an application is designed from raw text resources, each text resource must be specified in the following format:
corpora [list] (may be combined with 'language-models')
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>s3path</td><td>The s3 path to the CLEAN text resource.
If you are not 100% sure what 'clean' means, ask in the R&D group.
It should be compressed in gzip format and end with
the extension .gz.</td></tr>
<tr><td>lambda</td><td>The interpolation weight for the model, if any.
Should be a float with a value between zero and
one (0.0≤lambda≤ 1.0).</td></tr>
<tr><td>order</td><td>The target maximum n-gram order for this corpus.
Must be an integer1≤order≤5.
CAUTION:
A setting order > 3 may require significant RAM and/or storage during the model training and bu
phases, as well as decoding.</td></tr>
</table>
The language-models parameter can be set with either 'lstm-models'
Not yet implemented! or 'ngram-models'.
The section below assumes that this is set to ngram-models, and each element in the list is a valid, ARPA format ngram model in gzip format.
The language-models: ngram-models,[object] may be combined with 'corpora'.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>s3path</td><td>The s3 path to the valid, ARPA format ngram model.
This should be a text format file, compressed in gzip format, and ending with the .arpa.gz file extension</td></tr>
<tr><td>lambda</td><td>The interpolation weight for the model, if any. Should be a float with a value between zero and one
(0.0≤lambda≤ 1.0).</td></tr>
<tr><td>order</td><td>The target maximum n-gram order for this ngram model.
Must be an integer1≤order≤actual-order.
Note that this is only relevant in the case of interpolation for multiple models.</td></tr>
</table>
</td>
<td>Yes</td>
</tr>
</table>

The tune object configuration parameter is optional. Spitch recommends to configure the tune object.

Table 2: The Tune Object Configuration Parameters

Value	Description
s3path	Valid s3 path to the test set. See below for the required test set specification.
force	Boolean value, always force re-download of the resources. Typically set to `true`.
params	Sets of parameter ranges for `beam` (search beam), asf (acoustic scale factor) wip (word insertion penalty). Note that the search space, and number of iterations corresponds to the value of `#beam-vals * #asf-vals * #wip-vals`. The more you set, the longer your tuning experiments will take. See the above examples or ask the R&D team if you have questions about this.
objective	Indicates what should be optimized by the tuning process. The value of `optimize` may be set to any of wer (recommended),ser, or rtf. The `max-rtf` may also be set. This should be a float >0.0. This will be used as an auxiliary parameter during tuning. For example, to select the best WER with an RTF less than 0.5 times real time.

6.3.1 The NGram JSON Object - Example

The following example illustrates a valid JSON object that could be used to configure, build, tune and deploy a new Lingware application.

This application uses a pre-trained NGram model.

Note that the resources are external to the configuration and must actually exist.

```
{
   "topic": "demo-app",
   "language": "en-US",
   "srate": "8000",
   "port": "10001",
   "description": "Demo: [very] limited domain LVCSR suitable for clean microphone speech
application.",
   "acoustic-model": {
      "s3path": "s3://spitch-data/Processed/en-US/AcousticModels/clean-speech_en-US_8KHz"
   },
   "language-models": {
      "ngram-models": [
         {
            "s3path": "s3://spitch-data/Processed/en-US/LanguageModels/demo-lm.o3.arpa.gz",
            "lambda": 1.0,
            "order": 3
         }
      ]
   },
   "tune": {
      "s3path": "s3://spitch-data/Processed/en-US/TestSets/demo-test.tgz",
      "force": true,
      "params": {
         "beam": [10],
         "asf": [0.045,0.05],
         "wip": [0.0]
      },
      "objective": {
         "optimize": "wer",
         "max-rtf" : 0.5
      }
   }
}
```

6.3.2 The Text Corpora JSON Object - Example

The following example illustrates a valid JSON object that could be used to configure, build, tune and deploy a new Lingware application.

This application uses a two corpora and specifies interpolation weights.

```
{
  "topic": "demo-corpora-app",
  "language": "en-US",
  "srate": "8000",
  "port": "10002",
  "description": "Demo: [very] limited domain LVCSR resources suitable for a telephony
application.",
  "acoustic-model": {
    "s3path": "s3://spitch-data/Processed/en-US/AcousticModels/clean-speech_en-US_8KHz"
  },
  "corpora": [
    {
      "s3path": "s3://spitch-data/Processed/en-US/Corpora/demo-corpus-1.corpus.gz",
      "lambda": 0.8,
      "order": 3
    },
    {
      "s3path": "s3://spitch-data/Processed/en-US/Corpora/demo-corpus-2.corpus.gz",
      "lambda": 0.2,
      "order": 2
    }
  ],
  "tune": {
    "s3path": "s3://spitch-data/Processed/en-US/TestSets/demo-test.tgz",
    "force": true,
    "params": {
      "beam": [10],
      "asf": [0.045,0.05],
      "wip": [0.0]
    },
    "objective": {
      "optimize": "wer",
      "max-rtf" : 0.5
    }
  }
}
```

6.3.3 The GRXML JSON Object - Example

The following example illustrates a valid JSON object that could be used to configure, build, tune and deploy a new Lingware application.

This application specifies an expert or closed grammar, which is in turn defined as GRXML.

```
{
  "topic": "demo-grxml-app",
  "language": "en-US",
  "srate": "8000",
  "port": "10003",
  "description": "Demo: simple GRXML applcication suitable for a telephony application.",
  "acoustic-model": {
    "s3path": "s3://spitch-data/Processed/en-US/AcousticModels/clean-speech_en-US_8KHz"
  },
  "grxml": {
    "s3path": "s3://spitch-data/Processed/en-US/GRXML/demo-grammar.grxml"
  },
  "tune": {
    "s3path": "s3://spitch-data/Processed/en-US/TestSets/demo-test.tgz",
    "force": true,
    "params": {
      "beam": [10],
      "asf": [0.045,0.05],
      "wip": [0.0]
    },
    "objective": {
      "optimize": "wer",
      "max-rtf" : 0.5
    }
  }
}
```

7 Alphabetical 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