**Azure AI Services Continuous Integration (CI) and Continuous Deployment (CD)**

You can use Azure DevOps and GitHub Actions to manage your deployments. We have two examples of CI/CD integrations to train and deploy custom models for Speech and the Language Understanding (LUIS) service.

* **Target user(s):** Developers, data scientists, and data engineers
* **Benefits:** Allows you to continuously adjust, update, and deploy applications and models programmatically. There's significant benefit when regularly using your data to improve and update models for Speech, Vision, Language, and Decision
* **UI tools:** N/A - Code only
* **Subscription(s):** Azure account + Azure AI services resource + GitHub account

**CI/CD for custom speech**

Implement automated training, testing, and release management to enable continuous improvement of custom speech models as you apply updates to **training and testing data**. Through effective implementation of CI/CD workflows, you can ensure that the endpoint for the best-performing custom speech model is always available.

[**Continuous integration**](https://learn.microsoft.com/devops/develop/what-is-continuous-integration)**(CI)** is the engineering practice of frequently committing updates in a shared repository and performing an automated build on it. CI workflows for custom speech train a new model from its data sources and perform automated testing on the new model to ensure that it performs better than the previous model.

[**Continuous delivery**](https://learn.microsoft.com/devops/deliver/what-is-continuous-delivery)**(CD**) takes models from the CI process and creates an endpoint for each improved custom speech model. CD makes endpoints easily available to be integrated into solutions.

Custom CI/CD solutions are possible, but for a robust, pre-built solution, use the [Speech DevOps template repository](https://github.com/Azure-Samples/Speech-Service-DevOps-Template), which executes CI/CD workflows using GitHub Actions.

**CI/CD workflows for custom speech**

The purpose of these workflows is to ensure that each custom speech model has better recognition accuracy than the previous build. If the updates to the testing and/or training data improve the accuracy, these workflows create a new custom speech endpoint.

Git servers such as GitHub and Azure DevOps can run automated workflows when specific Git events happen, such as merges or pull requests. For example, a CI workflow can be triggered when updates to testing data are pushed to the *main* branch. Different Git Servers have different tooling but allow scripting command-line interface (CLI) commands so that they can execute on a build server.

Along the way, the workflows should name and store data, tests, test files, models, and endpoints such that they can be traced back to the commit or version they came from. It's also helpful to name these assets so that it's easy to see which were created after updating testing data versus training data.

**CI workflow for testing data updates**

The principal purpose of the CI/CD workflows is to build a new model using the training data, and to test that model using the testing data to establish whether the [**Word Error Rate (WER)**](https://docs.azure.cn/en-us/ai-services/speech-service/how-to-custom-speech-evaluate-data#evaluate-word-error-rate-wer) improved compared to the previous best-performing model (the "benchmark model"). If the new model performs better, it becomes the new benchmark model against which future models are compared.

The CI workflow for testing data updates should retest the current benchmark model with the updated test data to calculate the revised WER. This ensures that when the WER of a new model is compared to the WER of the benchmark, both models were tested against the same test data and you're comparing like with like.

This workflow should trigger on updates to testing data and:

* Test the benchmark model against the updated testing data.
* Store the test output, which contains the WER of the benchmark model, using the updated data.
* The WER from these tests will become the new benchmark WER that future models must beat.
* The CD workflow doesn't execute for updates to testing data.

**CI workflow for training data updates**

Updates to training data signify updates to the custom model.

This workflow should trigger on updates to training data and:

* Train a new model with the updated training data.
* Test the new model against the testing data.
* Store the test output, which contains the WER.
* Compare the WER from the new model to the WER from the benchmark model.
* If the WER doesn't improve, stop the workflow.
* If the WER improves, execute the CD workflow to create a custom speech endpoint.

**CD workflow**

After an update to the training data improves a model's recognition, the CD workflow should automatically execute to create a new endpoint for that model and make that endpoint available such that it can be used in a solution.

**Release management**

Most teams require a manual review and approval process for deployment to a production environment. For a production deployment, you might want to make sure it happens when key people on the development team are available for support, or during low-traffic periods.

**Tools for custom speech workflows**

Use the following tools for CI/CD automation workflows for custom speech:

* [Azure CLI](https://learn.microsoft.com/cli/azure/) to create an Azure service principal authentication, query Azure subscriptions, and store test results in Azure Blob.
* [Azure AI Speech CLI](https://docs.azure.cn/en-us/ai-services/speech-service/spx-overview) to interact with the Speech service from the command line or an automated workflow.

**DevOps solution for custom speech using GitHub Actions**

For an already-implemented DevOps solution for custom speech, go to the [Speech DevOps template repo](https://github.com/Azure-Samples/Speech-Service-DevOps-Template). Create a copy of the template and begin development of custom models with a robust DevOps system that includes testing, training, and versioning using GitHub Actions. The repository provides sample testing and training data to aid in setup and explain the workflow. After initial setup, replace the sample data with your project data.

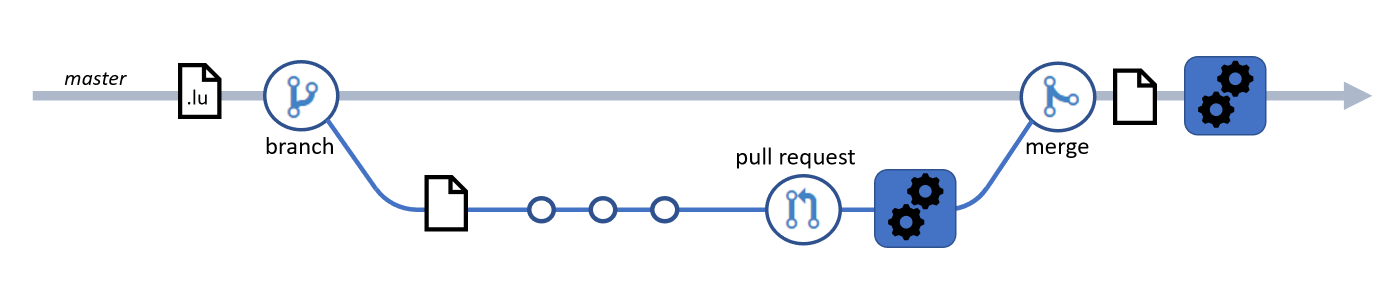
The [Speech DevOps template repo](https://github.com/Azure-Samples/Speech-Service-DevOps-Template) provides the infrastructure and detailed guidance to:

* Copy the template repository to your GitHub account, then create Azure resources and a [service principal](https://docs.azure.cn/en-us/active-directory/develop/app-objects-and-service-principals#service-principal-object) for the GitHub Actions CI/CD workflows.
* Walk through the "[dev inner loop](https://learn.microsoft.com/dotnet/architecture/containerized-lifecycle/design-develop-containerized-apps/docker-apps-inner-loop-workflow)." Update training and testing data from a feature branch, test the changes with a temporary development model, and raise a pull request to propose and review the changes.
* When training data is updated in a pull request to *main*, train models with the GitHub Actions CI workflow.
* Perform automated accuracy testing to establish a model's [Word Error Rate (WER)](https://docs.azure.cn/en-us/ai-services/speech-service/how-to-custom-speech-evaluate-data#evaluate-word-error-rate-wer). Store the test results in Azure Blob.
* Execute the CD workflow to create an endpoint when the WER improves.

**Continuous Integration and Continuous Delivery workflows for LUIS DevOps**

Software engineers who are developing a Language Understanding (LUIS) app can apply DevOps practices around [source control](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-sourcecontrol), [automated builds](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-automation), [testing](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-testing), and [release management](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-automation#release-management). This article describes concepts for implementing automated builds for LUIS.

**Build automation workflows for LUIS**

****

In your source code management (SCM) system, configure automated build pipelines to run at the following events:

1. PR workflow triggered when a [pull request](https://help.github.com/github/collaborating-with-issues-and-pull-requests/about-pull-requests) (PR) is raised. This workflow validates the contents of the PR *before* the updates get merged into the main branch.
2. CI/CD workflow triggered when updates are pushed to the main branch, for example upon merging the changes from a PR. This workflow ensures the quality of all updates to the main branch.

The CI/CD workflow combines two complementary development processes:

* [Continuous Integration](https://learn.microsoft.com/en-us/devops/develop/what-is-continuous-integration) (CI) is the engineering practice of frequently committing code in a shared repository and performing an automated build on it. Paired with an automated [testing](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-testing) approach, continuous integration allows us to verify that for each update, the LUDown source is still valid and can be imported into a LUIS app, but also that it passes a group of tests that verify the trained app can recognize the intents and entities required for your solution.
* [Continuous Delivery](https://learn.microsoft.com/en-us/devops/deliver/what-is-continuous-delivery) (CD) takes the Continuous Integration concept further to automatically deploy the application to an environment where you can do more in-depth testing. CD enables us to learn early about any unforeseen issues that arise from our changes as quickly as possible, and also to learn about gaps in our test coverage.

The goal of continuous integration and continuous delivery is to ensure that "main is always shippable,". For a LUIS app, this means that we could, if we needed to, take any version from the main branch LUIS app and ship it on production.

**Tools for building automation workflows for LUIS**

There are different build automation technologies available to create build automation workflows. All of them require that you can script steps using a command-line interface (CLI) or REST calls so that they can execute on a build server.

Use the following tools for building automation workflows for LUIS:

* [Bot Framework Tools LUIS CLI](https://github.com/microsoft/botbuilder-tools/tree/master/packages/LUIS) to work with LUIS apps and versions, train, test, and publish them within the LUIS service.
* [Azure CLI](https://learn.microsoft.com/en-us/cli/azure/) to query Azure subscriptions, fetch LUIS authoring and prediction keys, and to create an Azure [service principal](https://learn.microsoft.com/en-us/cli/azure/ad/sp) used for automation authentication.
* [NLU.DevOps](https://github.com/microsoft/NLU.DevOps) tool for [testing a LUIS app](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-testing) and to analyze test results.

**The PR workflow**

As mentioned, you configure this workflow to run when a developer raises a PR to propose changes to be merged from a feature branch into the main branch. Its purpose is to verify the quality of the changes in the PR before they're merged to the main branch.

This workflow should:

* Create a temporary LUIS app by importing the .lu source in the PR.
* Train and publish the LUIS app version.
* Run all the [unit tests](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-testing) against it.
* Pass the workflow if all the tests pass, otherwise fail it.
* Clean up and delete the temporary app.

If supported by your SCM, configure branch protection rules so that this workflow must complete successfully before the PR can be completed.

**The main branch CI/CD workflow**

Configure this workflow to run after the updates in the PR have been merged into the main branch. Its purpose is to keep the quality bar for your main branch high by testing the updates. If the updates meet the quality bar, this workflow deploys the new LUIS app version to an environment where you can do more in-depth testing.

This workflow should:

* Build a new version in your primary LUIS app (the app you maintain for the main branch) using the updated source code.
* Train and publish the LUIS app version.
* Run all the [unit tests](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-testing).
* Optionally run [batch tests](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-testing#how-to-do-unit-testing-and-batch-testing) to measure the quality and accuracy of the LUIS app version and compare it to some baseline.
* If the tests complete successfully:
  + Tag the source in the repo.
  + Run the Continuous Delivery (CD) job to deploy the LUIS app version to environments for further testing.

**Continuous delivery (CD)**

The CD job in a CI/CD workflow runs conditionally on success of the build and automated unit tests. Its job is to automatically deploy the LUIS application to an environment where you can do more testing.

There's no one recommended solution on how best to deploy your LUIS app, and you must implement the process that is appropriate for your project. The [LUIS DevOps template](https://github.com/Azure-Samples/LUIS-DevOps-Template) repo implements a simple solution for this which is to [publish the new LUIS app version](https://learn.microsoft.com/en-us/azure/ai-services/luis/how-to/publish) to the *production* publishing slot. This is fine for a simple setup. However, if you need to support a number of different production environments at the same time, such as *development*, *staging* and *UAT*, then the limit of two named publishing slots per app will prove insufficient.

Other options for deploying an app version include:

* Leave the app version published to the direct version endpoint and implement a process to configure downstream production environments with the direct version endpoint as required.
* Maintain different LUIS apps for each production environments and write automation steps to import the .lu into a new version in the LUIS app for the target production environment, to train, and publish it.
* Export the tested LUIS app version into a [LUIS docker container](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-container-howto?tabs=v3) and deploy the LUIS container to Azure [Container instances](https://learn.microsoft.com/en-us/azure/container-instances/).

**Release management**

* Generally, we recommend that you do continuous delivery only to your non-production environments, such as to development and staging. Most teams require a manual review and approval process for deployment to a production environment. For a production deployment, you might want to make sure it happens when key people on the development team are available for support, or during low-traffic periods.

**Apply DevOps to LUIS app development using GitHub Actions**

Go to the [LUIS DevOps template repo](https://github.com/Azure-Samples/LUIS-DevOps-Template) for a complete solution that implements DevOps and software engineering best practices for LUIS. You can use this template repo to create your own repository with built-in support for CI/CD workflows and practices that enable [source control](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-sourcecontrol), automated builds, [testing](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-testing), and release management with LUIS for your own project.

The [LUIS DevOps template repo](https://github.com/Azure-Samples/LUIS-DevOps-Template) walks through how to:

* Clone the template repo - Copy the template to your own GitHub repository.
* Configure LUIS resources - Create the [LUIS authoring and prediction resources in Azure](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-how-to-azure-subscription) that will be used by the continuous integration workflows.
* Configure the CI/CD workflows - Configure parameters for the CI/CD workflows and store them in [GitHub Secrets](https://help.github.com/actions/configuring-and-managing-workflows/creating-and-storing-encrypted-secrets).
* Walks through the ["dev inner loop"](https://learn.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/design-develop-containerized-apps/docker-apps-inner-loop-workflow) - The developer makes updates to a sample LUIS app while working in a development branch, tests the updates and then raises a pull request to propose changes and to seek review approval.
* Execute CI/CD workflows - Execute [continuous integration workflows to build and test a LUIS app](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-automation#build-automation-workflows-for-luis) using GitHub Actions.
* Perform automated testing - Perform [automated batch testing for a LUIS app](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-testing) to evaluate the quality of the app.
* Deploy the LUIS app - Execute a [continuous delivery (CD) job](https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-automation#continuous-delivery-cd) to publish the LUIS app.
* Use the repo with your own project - Explains how to use the repo with your own LUIS application.

**Source control and branch strategies for LUIS**

<https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-sourcecontrol>

**Testing for LUIS DevOps**

<https://learn.microsoft.com/en-us/azure/ai-services/luis/luis-concept-devops-testing>