Rapport

www.infosupport.com

# Survey Report

## Azure Pipelines DX Survey

Serggio Pizzella

28 october 2024

Draft

Solid Innovator

**Info Support B.V.**

**Kruisboog 42**
**3905 TG Veenendaal**
**(NL)**
**www.infosupport.com**

**K.v.K 3013 5370**
**BTW NL8062.30.277B01**
**IBAN NL92 RABO 0305 9528 89**
**BIC RABONL2U**

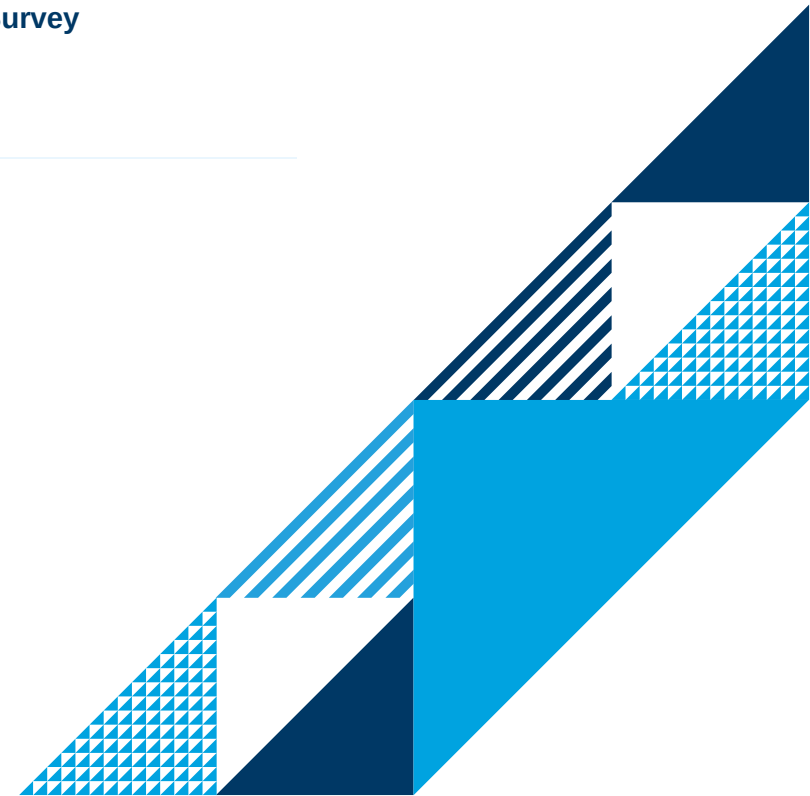
**Hoofdkantoor**
**Tel. +31(0)318 - 55 20 20**
**info.nl@infosupport.com**

**Kenniscentrum**
**Tel. +31(0)318 - 50 11 19**
**training.nl@infosupport.com**

# Survey Report

## Azure Pipelines DX Survey

| Details | |
|---|---|
| Titel | **Survey Report** |
| Project | **Azure Pipelines DX Survey** |
| Version | **1.0** |
| Status | **Draft** |
| Date | **28 october 2024** |



## History

| Version | Status | Date | Author | Changes |
|---|---|---|---|---|
| 1.0 | **Draft** | **10/10/2024** | **Serggio Pizzella** | **Creation, Introduction and Findings.** |
| 1.0 | **Draft** | **11/10/2024** | **Serggio Pizzella** | **Analysis and Conclusion.** |
| 2.0 | **Final** | **28/10/2024** | **Serggio Pizzella** | **Explain how azure pipelines work. Add reference to the survey.** |

## Distribution

| Version | Status | Date | To |
|---|---|---|---|
| 1.0 | **Draft** | **11/10/2024** | **Supervisor, first and second accessor. Trough Teams and Canvas respectively.** |
| 2.0 | **Final** | **28/10/2024** | **Supervisor and first accessor. Published on portfolio site.** |

# Contents

# 1. Introduction

Azure Pipelines are an essential tool for continuous integration and deployment at Info Support, but developers have expressed frustration with its usage. A common challenge is that the only way to validate a pipeline is by running it, this requires committing and pushing changes to see if they are correct, even for simple mistakes like typing errors in variable names. This leads to a slow and cumbersome process for catching minor issues, which ultimately impacts developer productivity.

The goal is to improve the experience by creating a diagnostics tool that can catch and report such errors early, directly in the editor. However, due to the vast range of potential errors and limited time, it's crucial to focus efforts on addressing the most common issues first. To achieve this, a survey was conducted to identify the most frequent pain points developers encounter while working with Azure Pipelines.

## 1.1 Objective

The survey's primary objective was to gather insights from developers about the most problematic features in Azure Pipelines. The goal was to rank these features based on how often they cause issues, helping to prioritize which areas the new diagnostics tool should address first.

## 1.2 Survey Design

The survey was distributed to developers within Info Support, aiming to understand the frequency of pipeline usage, their self-assessed proficiency, and the specific features that cause issues. Although the design was relatively straightforward, focusing mainly on feature-related frustrations, it became apparent that including questions about the time taken to resolve issues would have provided more nuanced insights. Nonetheless, the responses offer a valuable perspective on the challenges developers face. Over one week, 29 developers participated in the survey.

## 1.3 Target Audience

The target audience for the diagnostic tool is any developer working with Azure Pipelines. However, the immediate focus is on addressing the needs of developers at Info Support, based on their specific experiences with the platform, therefore the survey will target them.

### 1.4 How Azure Pipelines Work

Pipelines are configured using YAML files, which describe the sequence of jobs and stages that run tasks in a specific order. Stages group related jobs together, such as building, testing, and deploying, while jobs represent the individual steps of the pipeline.

Throughout the pipeline, developers rely on variables to store and reuse values across different stages, jobs or tasks. Variables can be predefined by the system, defined by the user, or provided through templates, which allow for code reuse across multiple pipelines. These templates can be customized using template parameters, providing flexibility to modify the behavior of the template without duplicating code.

Azure Pipelines also supports expressions that are evaluated during pipeline compilation, enabling the developer to programmatically define the pipeline they want to execute, based on predefined conditions, such as the branch the pipeline will run on.

Through its use of stages, variables, templates, and expressions, Azure Pipelines enables the automation of complex workflows.

### 1.5 The Survey

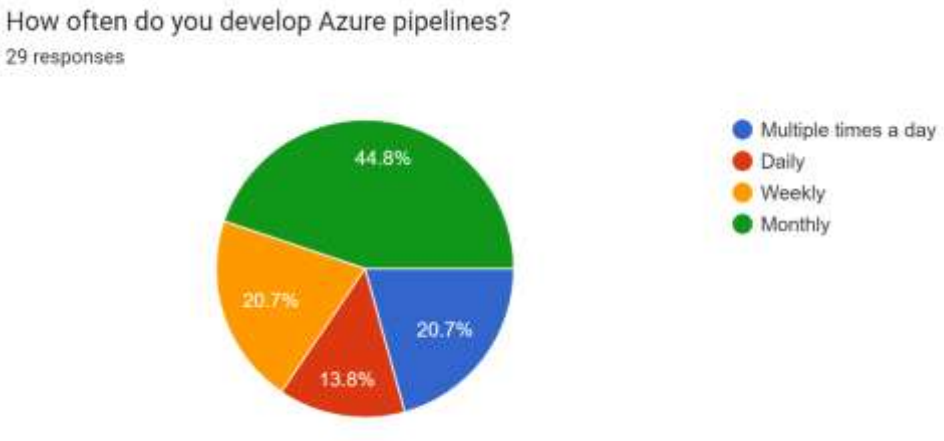
The full survey with all the questions can be found at https://forms.gle/cRhRyNR49DwFz8qV6.

# 2. Survey Findings

## 2.1 Participant Demographics

A total of 29 developers from Info Support participated in the survey. Their frequency of Azure Pipelines usage varied as follows:

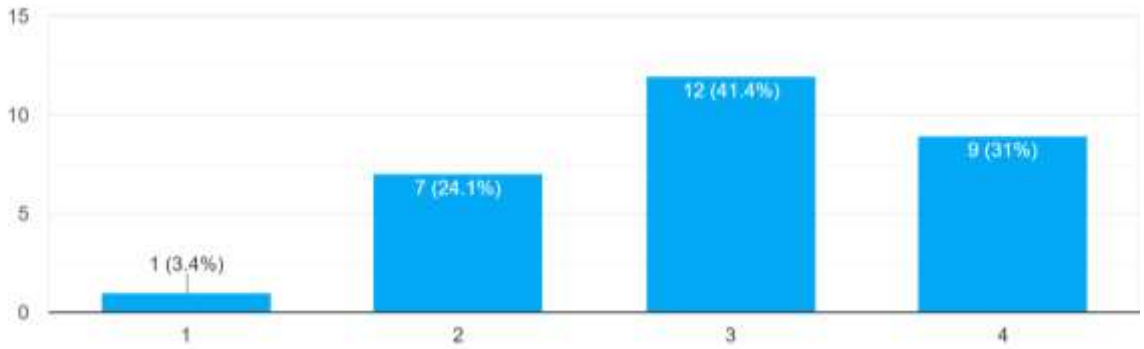
- **Monthly**: 13 participants
- **Weekly**: 6 participants
- **Daily**: 4 participants
- **Multiple times a day**: 6 participants




When asked to rate their proficiency with Azure Pipelines on a scale from 1 to 4, where 4 indicates the highest level of proficiency, the results were as follows:

- Proficiency level 4: 9 participants
- Proficiency level 3: 12 participants
- Proficiency level 2: 7 participants
- Proficiency level 1: 1 participant

Upon further investigation, it was noted that making the scale from 1 to 4 was not the best approach as this did not allow for a middle response option. This can be problematic as, forcing respondents to take a side may introduce unwanted variance or bias to the data. (Webster, 2021)



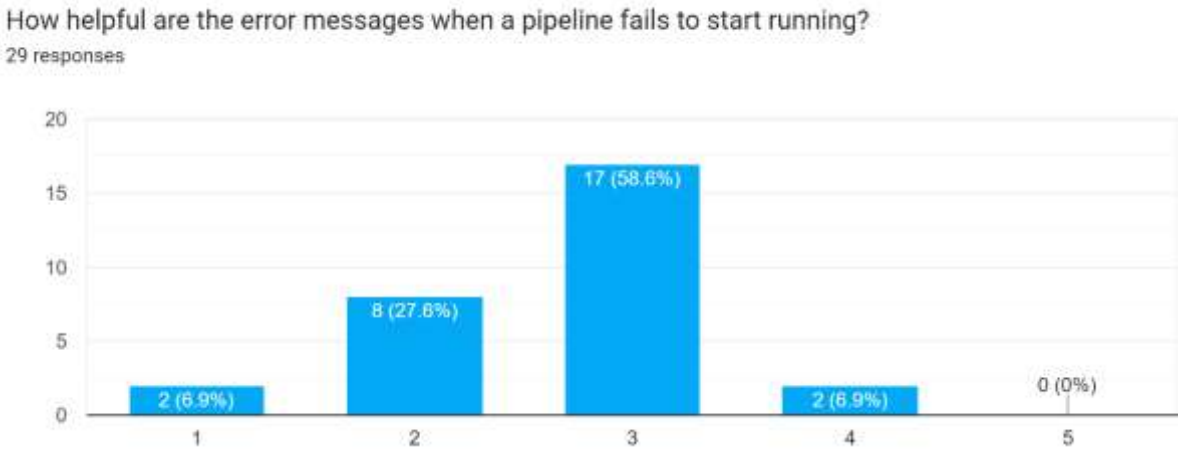
## 2.2 Feature related issues

The survey aimed to identify the features of Azure Pipelines that cause the most development issues. Participants were asked to rate how frequently different features were responsible for issues, with responses ranging from ‘None’ to ‘Many’. Out of the 39 features evaluated, the following key insights were observed:

- **Compile time expressions** were reported as the most problematic feature, with the majority of developers rating them as causing ‘Vast’ or ‘Many’ issues. Specifically, 8 participants rated them ‘Vast’ and 8 as ‘Many’, with only 7 participants rating it as ‘Few’ or less.
- **Template parameters and Variables** were other frequently highlighted features, with 14 participants indicating that they are responsible for ‘Vast’ or ‘Many’ issues.
- On the other hand, features such as **Continue on Error** were less commonly rated as problematic, with all participants rating it ‘Few’ or less, with 19 of them saying ‘None’ or ‘Some’.
- Lastly, **Pipeline decorators** are of note as participants seem to either not use the feature or not have any problems with it. With 16 participants rating the feature as ‘N/A’ and 11 rating them as causing ‘None’ issues.

## 2.3 Error messages

Participants were asked to rate the helpfulness of Azure Pipelines’ error messages, using a scale of 1 (does not lead to a solution) to 5 (directly points to a solution). The responses indicate a general level of dissatisfaction, with vast majority of participants rating them a **3** or below.




This suggests that while the error messages provide some level of assistance, there is considerable room for improvement, particularly in delivering more detailed or specific information that helps developers quickly identify and resolve issues.

## 2.4 Most Frustrating Aspects

When asked about the most frustrating aspects of getting their pipelines to run, participants frequently mentioned long waiting times to reach the section of the pipeline that contains issues as a primary concern. This delay is especially frustrating when small issues like syntax errors cause a pipeline to fail, requiring developers to restart the process. These issues stem from expressions which are evaluated at runtime, but whose correctness is not checked before running, such as ensuring a referenced variable exists and is properly spelled, both of which could be checked without needing to evaluate the expression. Common complaints included:

- "There's a lot of waiting time involved. I can't test it locally. I don't have a good syntax checker."
- "Deployment speed, can't run locally."
- ”Waiting for a syntax mistake”
- "Not being able to run it quickly local on my own machine. I need to commit some changes, start the pipeline, wait for it to be picked up of the queue, run the 10 steps that go well all to test the 11th step which fails. Makes the development process tedious and longer than needed.”
- “You tend to spend a lot of time on things that take less than 10 seconds in other development fields. Syntax and spelling for instance, knowing which variables are available. etc.”

These responses highlight the need for faster feedback loops and local pipeline validation to reduce the time spent waiting for the task to be picked up and executed.

# 3. Analysis

## 3.1 Methodology

The goal of this analysis was to produce a ranked list of the most problematic features in Azure Pipelines, taking into account several different metrics. To achieve this, a **weighted decision matrix** was used, a potent method in product management applied in multi-criteria decision-making. This approach allows us to consider various factors and weight them according to their relative importance. (Sooni, 2023)

In this case, the key criteria selected were:

- **Rated Severity**: This factor reflects how problematic each feature is, based on the participants' ratings. It provides a direct measure of how many issues the feature is responsible for.
- **Proficiency**: The proficiency factor takes into account how experienced the developers are with Azure Pipelines. More proficient users might encounter different or more nuanced issues, so it is important to incorporate their feedback into the prioritization.
- **Frequency**: The frequency factor ensures that the opinions of developers who use Azure Pipelines more frequently are given more weight, as they are more likely to encounter issues repeatedly.

A decision matrix is particularly useful when different factors must be weighed differently. By assigning weights to each of these criteria, we can account for the fact that some metrics (e.g., severity of the issue) may be more important than others (e.g., how often a developer makes pipelines).

While a separate study could be conducted to determine the optimal weights, various weights were tested in this analysis, each time prioritizing a different metric over the others. Regardless of the specific weights applied, the relative ranking of issues remained consistent. This consistency suggests that developers experience issues at a similar rate, regardless of their proficiency or frequency of use, making the final ranking robust and applicable across different user profiles. The only way to significantly alter the rankings was by not taking the severity of the issues into account, which would completely defeat the purpose of the ranking.

### 3.2 Final Formula for Ranking

The final ranking of the issues was determined using the weighted scoring formula, that stems from the decision matrix, to determine the score all participant assigned to a feature, and taking the average across all participants. The formula used for an individual participants score for a specific feature is as follows:

$$\text{Score} = \text{Weight } 1 \times \frac{\text{Rated Severity}}{\text{Max Severity}} + \text{Weight } 2 \times \frac{\text{Proficiency}}{\text{Max Proficiency}} + \text{Weight } 3 \times \frac{\text{Frequency}}{\text{Max Frequency}}$$

#### 3.2.1 Standardization of Metrics:

To ensure that no single factor disproportionately influences the final result due to differences in the range of values, each factor is divided by its maximum value. This normalization ensures that the scores for each factor fall between 0 and 1. By doing so, the weights assigned to each factor can fully control the influence of that metric on the final score, rather than the scale or range of the original data.

For instance, the frequency at which a feature is responsible for errors had a range of 5 different options, whereas the proficiency was on a scale of 1 to 4.

### 3.3 Final Results from the Ranking:

Based on the weighted formula and the responses collected from the survey, the following ranking of issues was generated:

1. Compile time expressions.
2. Variables at a stage level.
3. Predefined Variables.
4. Templating parameters.
5. Depends on Conditions at the stage level.
6. Conditions at the stage level.
7. Conditions at the job level.
8. Triggers.
9. User-defined variables.
10. Templates used for variable reuse.

# 4. Recommendations

Based on the final rankings of the most problematic features in Azure Pipelines, the following recommendations are proposed for the focus of the diagnostic tool:

1. **Focus on Compile Time Expressions**:
    - Compile time expressions were identified as the most problematic feature, and therefore should be the highest priority for providing diagnostics. The tool should focus on detecting common issues within these expressions, such as incorrect syntax, variable misuse, and invalid functions, to give developers immediate feedback in the editor.
2. **Diagnostics for Variables and Their Scope:**
    - Variables are a frequent source of issues, whether they are predefined by Azure, user-defined, or variables from templates. It is important for the tool to provide diagnostics that help developers verify the presence, availability, and correct scope of these variables (at stage, job, or pipeline levels). By ensuring that variables are properly recognized within their respective scopes, the tool can prevent common errors related to variable usage.
3. **Support for Template Parameters:**
    - Template parameters were also highly ranked as a source of frustration. The tool should provide diagnostics for the proper definition and usage of parameters in templating. This will help catch issues early, especially in pipelines that reuse templates extensively. As from anecdotal evidence, seems to be the case.
4. **Conditions at Job and Stage Levels:**
    - Conditions applied at both the job and stage levels often cause confusion, and participants highlighted their importance. Like with compile time expressions, the tool should aim to detect common issues, such as incorrect syntax or invalid functions.
5. **Improving Error Messages:**
    - Error messages were rated as moderately unhelpful, there is a clear opportunity to improve them. Developers expressed frustration with the lack of clarity in current error messages, particularly when pipelines fail to start. The diagnostic tool should aim to provide detailed, actionable error messages that pinpoint specific problems, such as misconfigured variables or invalid expressions. These enhanced error messages will help developers resolve issues faster, reducing the trial-and-error process currently experienced.

By focusing on these areas, the tool can significantly improve the developer experience by catching common issues early and reducing the time spent troubleshooting pipeline errors.

## 5. Conclusion

The analysis of the survey data has provided valuable insights into the most common and impactful issues developers face when working with Azure Pipelines. By using a decision matrix and applying a weighted scoring formula, a clear ranking of the problematic features was established. The results showed that certain features; such as compile time expressions, variables (predefined, user-defined, and template-based), and conditions; consistently cause issues for developers, regardless of their experience or frequency of use.

The diagnostic tool that will be developed should prioritize providing feedback on compile time expressions, variable presence and scope, template parameters, and condition statements. By addressing these areas, the tool can greatly enhance the Azure Pipelines development experience, reducing the time developers spend troubleshooting errors and improving overall productivity.

While the formula allowed for a balanced consideration of various factors, further refinement of the tool's capabilities can be informed by ongoing feedback and testing. This project lays the groundwork for targeted diagnostics that can adapt to the specific needs of developers working with Azure Pipelines.

In summary, the recommendations presented in this report reflect a research-driven approach to identifying and addressing key pain points in Azure Pipelines usage. The implementation of these recommendations will significantly improve the developer experience and simplify the pipeline creation process.

## 6. References

Clark, H. (2023, October 31). *What Is the Weighted Scoring Model and How Does it Work?* Retrieved from theproductmanager: https://theproductmanager.com/topics/weighted-scoring-model/

M., T. (2021, 07 03). *The Weighted Scoring Model – Project Management Tools & Techniques.* Retrieved from stakeholdermap: https://www.stakeholdermap.com/project-management/weighted-scoring-model.html

qualtrics. (n.d.). *Survey data analysis and best practices for reporting.* Retrieved from qualtrics: https://www.qualtrics.com/experience-management/research/analysis-reporting/

qualtrics. (n.d.). *Survey statistical analysis methods.* Retrieved from qualtrics: https://www.qualtrics.com/experience-management/research/survey-analysis-types/

SurveyPlanet. (2024, March 8). *How to make a survey report: A guide to analyzing and detailing insights.* Retrieved from blog.surveyplanet.com: https://blog.surveyplanet.com/how-to-make-a-survey-report-a-guide-to-analyzing-and-detailing-insights

Webster, W. (2021, January 15). *How to design rating scale questions.* Retrieved from Qualtrics: https://www.qualtrics.com/blog/three-tips-for-effectively-using-scale-point-questions/