Canfig: Validation-oriented configuration language

Tianyi Huang
University of Illinois at Urbana-Champaign
tianyih5@illinois.edu

Abstract

The Canfig language introduces a sophisticated approach to configuration management, designed to handle the complexities of modern software systems. Unlike traditional configuration languages, Canfig integrates advanced features such as Triggers, Structs, and Slices, each contributing to a more dynamic and flexible management environment. Triggers allow for the embedding of Python code directly within the configuration, enabling real-time, complex validations and automated adjustments in response to configuration changes. Structs empower developers to define custom data types, extending beyond basic types to accommodate specialized and complex configuration needs. Slices offer a method to dynamically manage and materialize views of configurations, allowing developers to specify and control the visibility and scope of configurations effectively. Together, these features position Canfig as a transformative tool in configuration management, facilitating precise control, enhanced flexibility, and increased reliability in deploying and maintaining system configurations.

1 Introduction

In the rapidly evolving landscape of software development, managing complex configurations has become a critical task, particularly as systems scale to enterprise levels. Mainstream configuration languages such as JSON, YAML, and Cue have been foundational in structuring and specifying settings, yet they often fall short in more complex scenarios. JSON and YAML, while popular due to their simplicity and human-readable format, lack capabilities for dynamic configuration and validation, leading to error-prone setups in large-scale environments. Cue, on the other hand, offers more sophisticated validation checks but remains limited in managing dependencies and dynamic conditions effectively.

These languages typically require additional tooling and scripts to handle complex validation rules or to dynamically adjust configurations based on runtime conditions, which can complicate deployment pipelines and increase the risk of human error. Furthermore, as configurations grow in size and complexity, these languages struggle with performance bottlenecks and lack mechanisms to handle interdependencies cleanly and efficiently.

Enter Canfig, a configuration language designed specifically to address these deficiencies. Canfig introduces advanced syntactic features like Triggers, Structs, and Slices, which not only enhance the flexibility and expressiveness of the language but also enable dynamic interactions and validations directly within the configuration files. This integrated approach allows Canfig to manage complex dependencies and conditions more effectively, making it ideally suited for large-scale, complex system configurations where traditional languages falter. By leveraging Python within its triggers, Canfig offers a unique capability for runtime validation and manipulation, thereby reducing overhead and minimizing the potential for errors. This introduction sets the stage for exploring how Canfig's innovative features can revolutionize configuration management in large-scale systems.

2 Background

Configuration management languages have traditionally offered basic validation checks to ensure parameters like number ranges and string patterns meet specified criteria. However, as configurations grow in complexity, these languages struggle to manage dependencies and dynamic conditions effectively. Canfig addresses these shortcomings by introducing a novel approach that includes:

Triggers: Triggers in Canfig are designed to respond to changes or events within the configuration environment, much like triggers in a traditional database system. These triggers allow developers to embed Python code directly, enabling not just simple validation checks but also complex operations and transformations. This integration provides a significant advantage by allowing runtime adjustments and dynamic responses based on the state of the configuration. For example, a trigger can automatically adjust related settings when a primary setting is changed, or it can perform data integrity checks against external criteria.

Struct: The STRUCT keyword in Canfig is pivotal for defining new data types, which are essential for handling diverse and complex configurations. Unlike traditional configuration languages that limit data types to predefined formats, Canfig's STRUCT allows users to define entirely custom data types tailored to their specific needs. These structures can contain various fields with types ranging from basic integers and strings to complex user-defined types. This capability greatly enhances the flexibility and expressiveness of the configuration language, ensuring that users can model their configurations as closely to their real-world counterparts as possible.

Slice: The SLICE keyword enables developers to dynamically manage subsets of configurations, similar to database views. It allows for tailoring configurations to specific environments or needs without duplicating the entire data

set. This adaptability makes it ideal for maintaining different states or versions of configurations, enhancing the dynamism and flexibility of system management.

3 Architecture

It is crucial to emphasize that Canfig functions as an interpreter rather than a compiler. This approach proves to be extremely convenient and beneficial, as it enables Canfig to utilize all SQL data types and features, such as CHECK constraints. This compatibility provides significant flexibility, which are essential for effective configuration management.

The architecture of the Canfig interpreter is organized into four distinct parts, each responsible for handling different aspects of the configuration processing:

Lexing: The lexing phase is managed using ocamllex, a tool that facilitates the lexical analysis by tokenizing the input configuration scripts. This initial step breaks down the raw data into manageable tokens, which are then used by the parser to construct a meaningful structure.

Parsing: The Canfig grammar, while straightforward, supports complex functionalities including user-defined types and structures similar to those found in SQL databases. The parser is implemented as a state machine, efficiently transforming tokens into a structured plan. This plan includes metadata, SQL-commands, and pre-SQL commands:

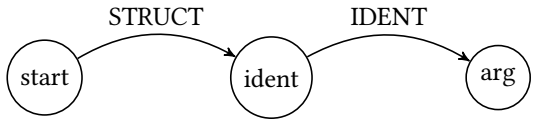
Evaluating: In the evaluation phase, the Canfig interpreter processes each struct definition into SQL-command plans. It also composes plans for each configuration field to facilitate efficient read/write operations. This step ensures that all components of the configuration are ready for deployment and interaction.

Registering: The final part of the architecture involves setting up triggers and slices as defined in the configuration. Triggers are registered to respond to specified events or changes, enhancing dynamic interaction with the configuration. Slices are set up to define specific views or subsets of the configuration, allowing tailored visibility and management depending on the operational requirements.

3.1 Parsing

Parsing in the Canfig interpreter is a crucial phase where tokens generated during the lexing process are transformed into structured plans. These plans dictate how configurations are interpreted and executed within the system. The parsing process in Canfig utilizes a state machine to manage and categorize tokens effectively, ensuring that the resulting plans are precise and ready for further evaluation.

State Machine Design: The state machine at the heart of the Canfig parser uses a series of states and transitions to process tokens and assemble them into meaningful constructs, such as SQL commands, metadata, or triggers. The transitions between states are triggered by the appearance of specific tokens, guiding the parser on how to assemble the final configuration plans. The graph below shows how Canfig Parser parse the two consecutive token: `STRUCT SAMPLE`



3.2 Evaluating

After the parsing phase, the Canfig interpreter generates a .candy file, which encapsulates the configuration's metadata, pre-SQL commands, and ready-to-run SQL commands. The evaluating phase focuses on processing these elements to ensure proper execution and integration into the database system. The primary objectives of the evaluating phase are two fold:

Conversion of Pre-SQL Commands to SQL Commands: This involves transforming all pre-SQL commands into executable SQL commands. There are specific transformations based on the type of command:

1. **User-Defined Struct:** For user-defined structs, the evaluator first generates a corresponding database table. Subsequently, it links the config fields to this new table through foreign keys, establishing relational integrity and facilitating complex queries.
2. **List Types:** In the case of list types, such as A LIST[B], the evaluator creates an intermediary table (named A_B) to manage the many-to-many relationship between A and B. It also sets the type of A to be NULL to accommodate the relational structure.

Execution of SQL Commands: Once all pre-SQL commands are converted into executable SQL commands, these commands are executed in the correct order to ensure data consistency and integrity. This step involves direct interaction with the database to apply the configurations.

3.2.1 Plan Objects and Operations

During evaluation, the interpreter also generates a Plan object for each field in the configuration. Each Plan object consists of two main components:

Writing and Reading Plans: These plans contain lists of atomic operations necessary to modify or retrieve data from the database. They are designed to ensure efficient and secure data handling.

Interfaces – Bind, Execute, and View:

- **Bind:** Prepares data for insertion or update by generating the necessary command sequences based on the input values.
- **Execute:** Applies the changes to the database, effectively committing the data modifications. This process simulates a Turing machine, sequentially processing and applying each command.

- **View:** Retrieves and displays data from the database, allowing for verification and review of the configuration settings.

4 Registering

The final phase in the Canfig interpreter architecture is the registration of triggers and slices, which are critical for enabling dynamic and customizable interactions within the configuration system. Although this phase has not yet been implemented, the proposed design and functionality are outlined below, providing insight into how Canfig aims to enhance configuration management through dynamic behavior and tailored visibility.

4.1 Triggers

Triggers in Canfig are designed to respond to specific events or changes within the configuration, thereby introducing a level of automation and responsiveness that is vital for maintaining system integrity and performance. The design idea for triggers involves:

Event-Driven Callbacks: Each plan associated with a configuration field will include a callback function. This function is triggered whenever a read operation is attempted on the data associated with that field.

Registration of Callbacks: During the registering phase, these callbacks are registered along with the triggers defined in the configuration. The triggers specify the conditions under which the callbacks should be executed, thus ensuring that the system reacts appropriately to changes or specific conditions within the configuration.

4.2 Slices

Slices provide a mechanism to define specific views or subsets of the configuration, which can be tailored to meet different operational requirements. This allows administrators and users to interact with only the relevant parts of the configuration, based on the current context or user role. The approach to implementing slices includes:

Conversion to Python Sets: The slice syntax specified in the configuration is converted into Python sets. This conversion facilitates the management and manipulation of the configuration subsets.

Hosting by the Frontend: Once converted, these sets are hosted by the frontend of the system. The frontend uses these sets to display or manage the slices, providing users with the tailored views as defined by the slices. This ensures that users can interact with the configuration in a controlled and segmented manner, enhancing usability and manageability.

The registration of triggers and slices is a sophisticated feature that Canfig plans to integrate, enabling not just customized configuration views but also ensuring that configurations are dynamically responsive to the system's operational states. This design not only facilitates better configuration management but also enhances the system's adaptability and responsiveness to changes.

This theoretical framework lays the groundwork for future implementation and offers a glimpse into how Canfig could revolutionize configuration management through advanced features and dynamic interactions.

5 Related Work

The concepts and methodologies in my project find precedence in significant prior research in the field of configuration management. A particularly relevant study is by Tang et al., which discusses holistic configuration management practices at Facebook [1]. Their work provides a comprehensive overview of managing complex configurations at scale, highlighting innovative approaches to enhance system reliability and efficiency. Our project builds upon these foundational ideas, extending them with the latest technological advancements in configuration languages and systems management.

6 Conclusion

The Canfig language has been meticulously designed with the intention of revolutionizing configuration management through its advanced features such as Triggers, Structs, and Slices. These elements collectively offer a robust framework for handling the intricacies of modern software systems with unprecedented flexibility and dynamism. However, the true efficacy and practicality of Canfig remain to be tested in real-world scenarios. As the project moves forward, the next critical phase will involve comprehensive testing across various environments to validate its functionality and performance. These tests will be pivotal in determining how well Canfig can integrate with existing systems and manage complex configurations under real operational conditions. Ultimately, the success of Canfig in practical applications will define its potential to become an essential tool in the toolkit of developers and system administrators aiming to streamline and enhance configuration management processes.

7 Metadata

The presentation of the project can be found at:

https://youtu.be/rwtOcufVZ9s

The code/data of the project can be found at:

https://github.com/TwinIsland/Canfig

References

[1] Tang, C., Kooburat, T., Venkatachalam, P., Chander, A., Wen, Z., Narayanan, A., Dowell, P., and Karl, R. Holistic configuration management at Facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP'15)* (Oct. 2015).