

HCL と意味的に階層化された UI の相互変換による クラウド構成の視覚的編集

平田 麟太郎† 井口 信和‡

† 近畿大学 情報学部情報学科 ‡ 情報学研究所

1 序論

Infrastructure as Code の普及により、クラウドインフラの構築や管理はコードによって宣言的に行われるようになった。特に HashiCorp 社が開発する Terraform [1] は事実上の標準ツールとして広く利用されている。しかし、独自の記述言語である HCL [2] はテキストベースであるため、システムが大規模化するにつれてリソース間の依存関係が複雑化し、開発者のメンタルモデルとコードの実態にかい離が生じやすい。また、大規模な設定ファイルから全体像を把握することは熟練者であっても困難であり、初学者にとっては学習コストが高い [3]。

この課題に対し、インフラ構成を可視化するツールは多数存在するが、その多くは閲覧専用であるか、編集機能を備えていても商用プロプライエタリな製品に限られている。さらに、既存ツールの重大な欠点として、HCL コードとグラフ表現の変換過程における情報の欠落が挙げられる。一般的に、Terraform の構成情報は JSON 形式の中間表現を経由して解析されるため、開発者が記述したコメントや空白といった、可読性や意図を伝えるためのメタ情報が失われてしまう。これは、コードを唯一の信頼源とする IaC の原則において、ツール導入の大きな障壁となる。

本研究では、Web ブラウザ上で動作するノードベース UI により、クラウド構成を視覚的に編集可能な OSS 「TerraGUI」を提案する。提案システムは、HCL の具象構文木を扱うことで、コードの体裁やコメントを保持したままグラフとの双方向同期を行うロスレス相互変換を実現した。また、リソースの意味的なまとまりを自動的にグルーピングする階層化機能により、認知負荷を低減する。

2 関連研究

Terraform の可視化ツールとして、Terraform Visual [4] や Inframap [5] などが存在する。これらは主に Terraform の State ファイルや JSON 出力を入力とする。この手法は解析が容易である反面、静的な可視化にとどまり、GUI 上での編集結果をコードに書き戻す機能を持たない。

一方、Brainboard [6] のような商用サービスは GUI での編集をサポートしている。しかし、これらのツールはインポート時にコードを独自の内部表現に変換し、保存時に再生成するアプローチを採ることが多い。この過程で、元の HCL ファイルに含まれていたコメントや独自のフォーマットが失われる場合がある。既存のコードベースを持つ

プロジェクトに導入した場合、一部の変更だけでファイル全体のフォーマットが変更され、バージョン管理システム上で大量の差分が発生する。これは、コードレビューにおいて論理的な変更点の判別を困難にし、チーム開発を阻害する。

3 研究内容

本研究では、以下の 2 点を満たす OSS 環境の構築を行った。

1. **ロスレスな双方向同期:** HCL コードをパースする際、抽象構文木 (AST) ではなく、コメントや空白情報を含む具象構文木 (CST) として扱う。GUI での変更は差分としてのみコードに適用され、元の記述を破壊しない。これにより IaC の原則を崩さずに GUI の利便性を享受できる。
2. **意味的な階層化:** リソースのフラットな羅列ではなく、VPC やサブネットといった包含関係や、IAM ロールとポリシーのような論理的なグループを UI 上で階層化して表示し、編集時の視認性を高める。

3.1 システム構成

TerraGUI は、Next.js 及び React Flow を用いた Web アプリケーションとして実装されている。ユーザーはブラウザ上でノードとして表現されるリソースと、エッジとして表現される依存関係进行操作し、バックグラウンドで動作するサーバーがリアルタイムに HCL コードを更新する。この全体構成とデータフローを図 1 に示す。

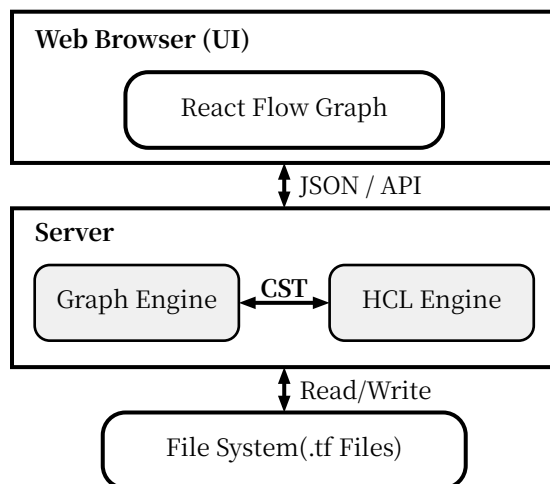


図 1: TerraGUI のシステム概要とデータフロー

3.2 HCL とグラフのロスレス相互変換

本システムは、新規性の中核となる以下2つのエンジンにより、HCL テキストとグラフデータ構造の相互変換を実現している。従来のツールが Terraform の State ファイルや JSON 出力を正としていたのに対し、TerraGUI はソースコードである .tf ファイル自体を解析対象とする。編集前後におけるコメント保持の挙動を図2に示す。

1. **HCL Engine:** HCL と具象構文木 (CST) をロスレスに相互変換するモジュールである。HCL パーサを用いてファイルを解析し、リソースブロック、属性、式を抽出する。この際、各トークンの位置情報とともにコメント (# や //) や空行もデータ構造内に保持する。これにより、真の意味で情報のロスレス性を保証する。
2. **Graph Engine:** CST と React Flow グラフの相互変換を担う。CST からグラフへの変換では、ELK.js を用いた自動レイアウトを行い、各ノードのデータプロパティとして CST を埋め込む。グラフから CST への変換 (保存) 時は、この埋め込まれた CST に対して編集差分を適用し、HCL を再構成する。これにより、レイアウト情報を持たない HCL に対しても、記述者の意図 (コメントや空白) を保持したままの編集を可能にしている。

Before Editing

```
resource "aws" "web" {
  // Important (Keep)
  ami = "old-123"
}
```

After Editing

```
resource "aws" "web" {
  // Important (Keep)
  ami = "new-456"
}
```

↓ Update ami to new-456

図 2: 編集前後におけるコメント保持の挙動

3.3 意味的な階層化

IaC コードでは全てのリソースが並列に記述されることが多いが、TerraGUI ではリソースタイプや vpc_id などの属性を解析し、視覚的な包含関係を構築する。例えば、AWS プロバイダにおいて aws_subnet リソースが vpc_id 属性を持つ場合、UI 上ではそのサブネットノードを VPC ノードの内部に配置する。また、セキュリティグループや IAM といった論理的な設定群は仮想グループノードとしてまとめることで、グラフの複雑性を抑制している。

3.4 編集の適用

TerraGUI は単なるエディタにとどまらず、編集結果を実際のクラウド環境へ適用する機能も備えている。編集完了後、ユーザーが適用操作を行うと、システムは以下の手順を実行する。

1. **HCL 生成:** Graph Engine 及び HCL Engine が最新のグラフ状態から HCL コードを生成する。
2. **実行計画の作成:** データベースに暗号化して保存されたプロバイダの認証情報 (AWS Access Key 等) を利用し、バックグラウンドで terraform plan コマンドを実

行する。出力された実行計画は UI 上に表示され、ユーザーは変更内容が意図通りかを確認できる。

3. **インフラの更新:** ユーザーの承認後、terraform apply コマンドが実行され、実環境への変更が適用される。

このプロセスにより、GUI 上の編集、コードへの反映、そして実環境への適用という一連のフローが完結し、常に HCL コードを正とした整合性のあるインフラ管理が実現される。

3.5 リソースアイコンの自動生成

数千種類に及ぶクラウドリソースの可視化において、手動でのアイコン紐付けは困難である。本システムでは、AWS, Azure, GCP の公式アイコンセットを自動収集し、Terraform のリソース名とファイル名のトークン類似度に基づいて動的にマッピングする機構を実装した。これにより、プロバイダの更新に追従した持続的なアイコン管理を実現している。

3.6 実装と機能的評価

現在、GitHub 上の kerthical/terrargui で OSS として公開しており、Docker コンテナを用いて容易にローカル環境で起動可能である。本システムの品質保証として、HCL Engine や Graph Engine のロスレス性を検証する単体テストから、実際のブラウザ操作を模倣したシナリオテストまでを実施し、機能の正当性を確認した。特にインポート機能においては、複雑な構成を持つ既存の HCL ファイルを読み込み、内部データ構造を経て再度ファイル出力した際に、コメントや空白を含むテキストが完全に復元されることを重点的に検証している。これらのテストを通じ、TerraGUI が既存の IaC 資産を破壊することなく、安全に視覚的な編集を提供できることを確認した。

4 結論

本稿では、HCL のロスレス編集と意味的階層化を実現する OSS 「TerraGUI」を提案した。提案手法により、既存の IaC 資産のコメント等の価値を損なうことなく、直感的な GUI 編集が可能となった。今後は共同編集機能の追加や、AI を用いた構成提案機能の統合を進める予定である。

参考文献

- [1] HashiCorp, Terraform, <https://www.terraform.io/>.
- [2] HashiCorp, HCL (HashiCorp Configuration Language), <https://github.com/hashicorp/hcl>.
- [3] Pierre-Jean Quéval, Nicole Elisabeth Hörner, Evangelos Ntontos, Uwe Zdun, On the understandability of coupling-related practices in infrastructure-as-code based deployments, Information and Software Technology, 185, pp. 107761 (2025), <https://www.sciencedirect.com/science/article/pii/S0950584925001004>.
- [4] Hieven, Terraform Visual, <https://github.com/hieven/terraform-visual>.
- [5] Cycloid, Inframap, <https://github.com/cycloidio/inframap>.
- [6] Brainboard, <https://www.brainboard.co/>.