卒業研究報告書

題目

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

指導教員

井口信和教授

報告者

22-1-211-0187

平田麟太朗

近畿大学情報学部情報学科

2025年12月30日提出

概要

クラウドコンピューティングの普及に伴い，Infrastructure as Code は現代のシステム開発において不可欠なプラクティスとなっている．中でも HashiCorp 社が提供する Terraform は，宣言的な記述言語である HashiCorp Configuration Language を用いることで，マルチクラウド環境におけるリソース管理の事実上の標準としての地位を確立している．しかし，システムが大規模化・複雑化するにつれて，テキストベースのコードのみからリソース間の依存関係や全体構造を把握することは困難となり，開発者の認知負荷が増大するという課題がある．

この課題に対し，Terraform の構成を可視化するツールは多数存在するが，その多くは一方通行の可視化にとどまるか，あるいは編集が可能であってもコード内のコメントやフォーマットといったメタ情報を保持できないという問題がある．Infrastructure as Code においてコードは唯一の信頼源であり，ツールによる自動生成でコードの可読性が損なわれることは，保守性の観点から許容しがたい．

本研究では，HCL コードの構造を維持したまま視覚的な編集を可能にする OSS 環境「TerraGUI」を提案・開発する．本システムは，HCL の具象構文木を用いた独自の中間表現により，コードとグラフ表現のロスレスな相互変換を実現した．具体的には，typescript-parsec を用いて HCL をトークンレベルで解析し，コメントや空白を含む構文木を構築する．さらに，elkjs によるレイアウトアルゴリズムとリソース属性の解析を組み合わせることで，VPC やサブネットといった包含関係を考慮した意味的に階層化されたグラフを自動生成する．

本論文では，提案システムの設計思想，Next.js と React Flow を用いた実装の詳細，及び Amazon Web Services，Google Cloud Platform，Microsoft Azure といった主要クラウドプロバイダへの対応について述べる．また，実装したシステムを機能的に評価し，既存の Infrastructure as Code ワークフローを破壊することなく，視覚的な支援による生産性向上が達成できることを示す．

目次

1 序論

1.1 研究背景

近年，デジタルトランスフォーメーションの加速に伴い，企業におけるクラウドサービスの利用は拡大の一途を辿っている．Amazon Web Services，Google Cloud Platform，Microsoft Azure といったパブリッククラウドプロバイダは，コンピュート，ストレージ，ネットワークに加え，機械学習や IoT といった高度なマネージドサービスを提供しており，これらを組み合わせることで迅速なサービス開発が可能となっている．

こうした環境において，インフラストラクチャの構築・管理を手動で行うことは，操作ミスの誘発や再現性の欠如といったリスクを伴う．また，手動操作のログは残りにくく，監査や変更履歴の追跡が困難である．そのため，インフラの構成をコードとして記述し，バージョン管理システムで管理する Infrastructure as Code の手法が一般化した．Infrastructure as Code ツールの中でも Terraform [1] は，特定のクラウドベンダーに依存しないオープンソースのツールとして広く利用されている．Terraform は HashiCorp Configuration Language [2] と呼ばれる独自のドメイン固有言語を用い，リソースの状態を宣言的に定義する．

一方，Infrastructure as Code の実現手段は Terraform に限らない．AWS CloudFormation [3]，Azure Resource Manager テンプレート [4]，Google Cloud Deployment Manager [5] といったクラウド固有 DSL に加え，Pulumi [6] や CDK for Terraform [7] のような汎用言語ベースのアプローチが存在する．さらに，Ansible [8]，Puppet [9]，Chef [10]，SaltStack [11] といった構成管理系のツールも，Infrastructure as Code の実践において重要な位置を占める．このように Infrastructure as Code は多様な技術レイヤに広がっており，それぞれが異なる表現力や運用モデルを持つ．

1.2 現状の課題

Infrastructure as Code の導入により，インフラ構築の自動化や再現性の確保は達成された．しかし，テキストベースの管理には依然として以下のような課題が残る．

まず，全体像把握の困難さが挙げられる．大規模なシステムでは数千行に及ぶ HCL コードが記述され，リソース間の依存関係は複雑に絡み合う．コード上の参照記述を追うだけでは，システム全体のトポロジーを脳内で構築することは，熟練したエンジニアであっても困難である [12]．

次に，学習コストの高さがある．クラウドプロバイダが提供するリソースの種類は膨大であり，それぞれに固有の設定項目が存在する．初学者が正しい HCL の構文と各リソースの必須パラメータを同時に理解し，適切な構成を記述するには多くの時間を要する．エディタの補完機能も存在するが，視覚的な補助なしにアーキテクチャを設計することは難しい．

また，既存ツールの限界も課題である．これらの課題を解決するために，Brainboard [13] や Terraform Visual [14]，Inframap [15]，Blast Radius [16]，terraform graph [17] などの可視化ツールが存在する．しかし，これらは既存のコードを読み込んで図示するだけの一方通行であったり，GUI で編集すると元のコードのコメントやフォーマットが破壊される不可逆的変換といった問題を抱えている．特に後者は，コードレビューの妨げとなり，既存の開発フローとの親和性を著しく低下させる．

さらに，品質・セキュリティ観点の複雑化がある．Infrastructure as Code は通常のソフトウェアと同様に欠陥やコードスメルを内包し得ることが示されており，欠陥分類やアンチパターン，スメル検出に関する研究が蓄積されている [18–23]．さらに，セキュリティスメルや秘密情報の扱いに関する実証研究もあり，設定の誤りがシステム全体の脆弱性につながることが報告されている [24–27]．

加えて，運用とレビューの断絶も問題となる．Infrastructure as Code の保守にはレビューが不可欠であるが，レビュー時に全体構造を把握できないことが品質低下につながるとの報告がある [28]．また，Terraform Registry 上のモジュール更新や依存関係の追跡は複雑であり，メンテナンス負荷が増大する [29,30]．

1.3 研究の目的

本研究の目的は，Infrastructure as Code の利点であるコードによる管理を損なうことなく，視覚的な操作による直感的な理解と編集を両立する環境を構築することである．具体的には，以下の要件を満たす OSS ツール「TerraGUI」を開発する．

第一に，ロスレスな相互変換である．HCL コードとグラフ UI の間で，コメントや空白を含む全ての情報を保持したまま双方向に同期する．これにより，GUI での変更が最小限の差分としてコードに反映される．第二に，意味的な可視化である．単なるリソースの羅列ではなく，ネットワーク階層や VPC，Subnet といった論理グループに基づいた見やすいグラフ構造を自動生成する．第三に，既存ワークフローとの統合である．独自の保存形式を導入せず，標準的な HCL テキストをそのまま保持することで，既存の Git ベースの開発フローへ円滑に組み込めるようにする．第四に，スキーマ駆動の編集体験である．Terraform プロバイダのスキーマ情報 [31] を利用し，リソース属性の入力支援と妥当性の高い編集体験を提供する．

1.4 本研究の貢献

本研究では，HCL の具象構文木を用いたロスレス編集機構を設計・実装し，コメントや整形を保持したままの差分更新を可能にした [32,33]．また，リソース属性から意味的な包含関係を推論するグラフ構築アルゴリズムを設計し，視認性の高い階層化グラフを自動生成した [34]．さらに，Terraform プロバイダスキーマを用いた動的プロパティエディタと，公式アイコンセットの自動マッピング機構を実装した [31]．加えて，HCL テキストを単一の真実源とし，graph_json を派生データとして扱うローカルファースト設計を採用することで，編集体験と整合性を確保した．

1.5 論文の構成

本論文は全 10 章で構成される． 第 1 章では，研究の背景，課題，目的について述べた． 第 2 章では，関連技術を概説する． 第 3 章では，関連研究と既存ツールを比較する． 第 4 章では，提案システム「TerraGUI」の概要と設計思想について述べる． 第 5 章では，システムの実装詳細，特に HCL 解析エンジンとグラフエンジンの仕組みについて詳述する．第 6 章では，ユーザインターフェースとワークフローを紹介する． 第 7 章では，データ管理と運用設計を述べる． 第 8 章で評価し，第 9 章で考察する． 第 10 章で結論と今後の展望を述べる．

2 関連技術

2.1 Infrastructure as Code と Terraform

Infrastructure as Code は，サーバー，ネットワーク，データベースなどのインフラ構成を，スクリプトや定義ファイルとして記述し，ソフトウェア開発のベストプラクティスをインフラ運用に適用する手法である．

Terraform は，HashiCorp 社が開発する Infrastructure as Code ツールであり，以下の特徴を持つ．まず，宣言的記述として，手順ではなく，あるべき状態を記述する．次に，リソースグラフとして，リソース間の依存関係をグラフとして管理し，適切な順序で作成・更新する [17]．また，State ファイルとして，実環境の状態を JSON 形式のファイルに記録し，コードとの差分を検出する．

Terraform Registry は，プロバイダとモジュールを共有するエコシステムの基盤であり，構成部品の再利用性を高めている [29]．また，Terraform Language Server は HCL の静的解析や補完に利用される [35].

2.1.1 Terraform 言語仕様と実行モデル

Terraform Language は, 宣言的なリソースブロックと式評価を組み合わせることで, 構成の再利用と差分更新を可能にする．ブロック内の属性値は式として評価され，for 式や条件式，nullish 演算子，関数呼び出しなどの表現力を持つ [36]．また，dynamic ブロックにより反復的なネスト構造を記述でき，count，for_each，depends_on，provider，lifecycle といったメタ引数は，繰り返しや依存関係，ライフサイクル制御を担う [37,38].

構成の再利用性はモジュール単位で整理される．モジュールは入力変数と出力を持つ再利用単位であり，リポジトリ全体の規模拡大を前提とした設計指針となる [39]．状態管理は state ファイルによって実現され，実環境のリソースとコードの対応関係を保持する [40]．ワークスペースは複数環境の state を分離し，開発・検証・本番の切替を支援する [41].

実行モデルとしては，plan が差分を計算し，apply が実行する二段階プロセスである [42,43]．validate は構文・型・参照を検証し，fmt は表記ゆれを整形する [44,45]．さらに Terraform は JSON 構文を公式にサポートしており，外部ツールとの連携や自動生成の出力先として利用できる [46,47].

2.1.2 Terraform 周辺ツールと運用

Terraform の周辺には，運用や品質保証を支援するツール群が存在する．Terragrunt は複数環境の管理を支援し，TFLint や tfsec，Checkov，Terrascan はコード規約やセキュリティを検証する [48–52]．Infracost はコストを見積もり，OPA や Conftest はポリシーをコードとして適用する [53–55]．これらのツールは Infrastructure as Code の保守性向上に寄与するが，グラフ UI との統合は限定的である．

2.1.3 Infrastructure as Code 運用プラットフォームと実行基盤

Terraform の運用は，CI/CD と権限管理を統合したプラットフォームと密接に関連する．HCP Terraform（Terraform Cloud）はリモート実行, ポリシー評価, 状態管理を担う [56]. また, Terraform の互換実装として OpenTofu が登場し, ライセンスやコミュニティ主導の開発体制が議論の対象となっている [57].

実務では, Pull Request 駆動で Plan/Apply を行う Atlantis, 複数クラウド/組織を横断したポリシーと実行を提供する Spacelift や Scalr などが用いられる [58–60]．さらに，Terraform 以外のオーケストレーション基盤として Cloudify があり，多様な DSL の統合を支援する [61]．本研究はこれらの運用基盤と競合するのではなく，ローカル開発段階の可視化と編集性を補完する位置付けを狙う．

2.1.4 HashiCorp Configuration Language

HashiCorp Configuration Language は Terraform のために設計された言語であり，JSON と互換性を持ちつつ，人間にとっての可読性と書きやすさを重視している [2,32]．例えば，リソース定義は以下のようにブロック構造で記述される．

```
resource "aws_instance" "web" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"

  tags = {
    Name = "HelloWorld"
  }
}
```

HCL は JSON への変換を経由して解析されることが多いものの，その過程でコメントや空行といった意味的に不要だが人間には必要な情報が失われる．この問題に対し，hclwrite [33] や tree-sitter-hcl [62] のようなツールは構文解析を支援するが，ロスレス編集を前提とした統合 UI まで踏み込む例は少ない．本研究では typescript-parsec [63] を利用し，トークンを保持した具象構文木を構築する．

2.2 宣言的基盤とモデル駆動の潮流

Infrastructure as Code は Terraform だけでなく，より広い宣言的基盤と連続性を持つ．TOSCA はクラウドアプリケーションのトポロジーを宣言的に記述する標準であり，サービス間依存やライフサイクルをモデルとして扱う [64]．Kubernetes は宣言的なリソース定義によってクラスタ状態を制御し，コントローラが差分を解消するモデルを採用している [65]．Crossplane は Kubernetes の制御ループをクラウドリソースに拡張し，インフラの宣言的管理を Kubernetes API の枠組みに統合する [66]．

これらは宣言的定義を単一の真実源とし，実体の状態を収束させるという Infrastructure as Code の原理を共有する．TerraGUI は Terraform に焦点を当てる一方で，こうした宣言的基盤で共通する可視化・編集課題を解決する枠組みとして位置付けられる．

2.3 構文解析技術

プログラミング言語の解析において，ソースコードは通常，抽象構文木に変換される．抽象構文木はプログラムの意味的に不要な情報を捨象し，論理構造のみを木構造で表現する．コンパイラやインタプリタにとってはこれで十分であるが，ソースコードを再生成するフォーマッタやリファクタリングツールにとっては不十分である．

これに対し，具象構文木と呼ばれるデータ構造は，ソースコード上の全てのトークンを保持する．Python における LibCST [67] や JavaScript の Recast [68] など，ロスレス編集を志向した具象構文木ライブラリが存在する．本研究では，HCL の編集において元のコードの体裁を維持するために，具象構文木のアプローチを採用する．

構文解析の実装手法としては，ANTLR のようなパーサ生成器を用いて文法仕様から解析器を生成する方法が一般的である [69]．一方で，パーサコンビネータによる実装は，式の優先順位やエラー箇所の特定といった処理をコードレベルで柔軟に記述できる利点がある．本研究で採用した ts-parsec はこの系譜にあり，構文規則を関数合成で記述することで，HCL の式文法を細かく制御できる [63]．

また，DSL に対する解析手法として Parsing Expression Grammar が広く知られており，バックトラックを伴う決定的な認識規則として定義される [70]．Parsing Expression Grammar を効率的に扱うための Packrat Parsing は，メモ化によって線形時間解析を保証する手法である [71]．HCL のように条件式やスプラット式など多様な構文要素を持つ言語では，このような規則性の高い解析モデルが有効である．

構文解析手法と CST 利用の位置付けは 図 1 に示す．

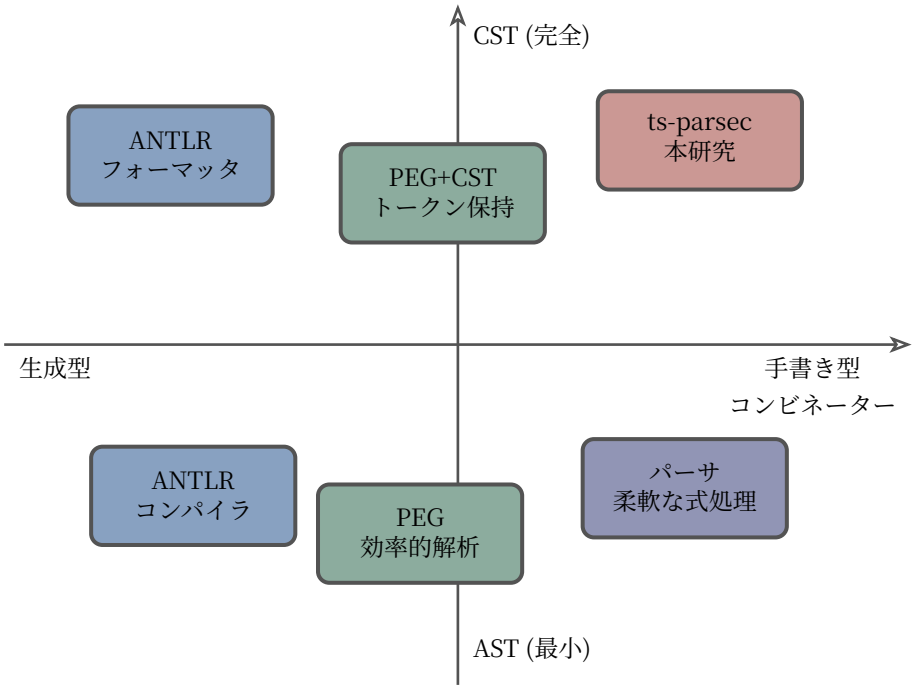

図 1: 構文解析手法と CST 利用の位置付け

2.4 グラフ可視化とレイアウト

依存関係を可視化する手段として，Graphviz [72] と DOT 言語 [73] は代表的な手法である．Terraform の terraform graph も DOT を出力するため，Graphviz と組み合わせて静的な依存関係図を生成できる [17].

一方, 本研究ではインタラクティブな編集を前提とするため, Web 上でのグラフ操作を支える React Flow (XYFlow) [74] と，自動レイアウトエンジン ELK/elkjs [34,75] を採用する．ELK の layered アルゴリズムは依存方向が明確なグラフに適しており，階層構造を維持した配置を生成できる．

Layered レイアウトは, Sugiyama 法に代表される階層型グラフ描画の系譜に属する [76]．ノードを階層ごとに割り当て, エッジ交差を最小化しながら配置を最適化するため, 大規模な依存関係図でも読みやすい配置を得られる．TerraGUI では, ELK のレイアウト結果を React Flow の座標に変換し, エッジラベル位置も含めて保持することで，静的図と同等の視認性を保ちながらインタラクション性を維持する．

2.5 Web フロントエンド技術

2.5.1 Next.js App Router

Next.js [77] は React ベースの Web フレームワークであり，App Router は React Server Components (RSC) を基盤としている．RSC により，コンポーネントをサーバー側でレンダリングし，クライアントへの転送データ量を削減できる．また，Server Actions 機能により，API エンドポイントを明示的に作成することなく，関数呼び出しのようにサーバー側の処理を実行できる．本研究では，ファイルシステムへのアクセスや DB 操作を伴う処理において Server Actions を多用している．

2.5.2 React Flow と ELK.js

React Flow は，ノードベースのアプリケーションを構築するための React ライブラリである．ノードのドラッグ移動，ズーム，パンといった基本的なインタラクション機能を提供し，カスタムノードやカスタムエッジの実装も容易である [74]． ELK.js (Eclipse Layout Kernel for JavaScript) は，グラ

フを自動レイアウトするライブラリである [34]．複雑なノード間の接続関係を解析し，エッジの交差を最小化しつつ，階層構造を持った見やすい配置座標を計算する．本システムでは，React Flow の表示座標を決定するために ELK.js を利用している．

2.5.3 コードエディタと UI 基盤

コードエディタには Monaco Editor [78] を用いる．Monaco は VS Code のエディタコアを提供し，高速なレンダリングと拡張性を持つ．UI の構築には React [79]，Next.js [77] を基盤とし，Tailwind CSS [80] と Radix UI [81] を利用してコンポーネントの一貫性を担保する．

UI レイアウトの分割には Allotment を使用し，グラフとエディタの 2 ペイン構成を柔軟に調整できる [82]．フォーム入力は React Hook Form で状態を管理し，Zod で入力を検証することで，入力エラーを早期に検知する [83,84]．アイコンは Lucide の SVG セットを利用し，機能や操作の視認性を高める [85]．

2.6 データ管理と ORM

データベースには SQLite [86] を採用し，ORM として Drizzle [87] を利用する．SQLite は軽量でローカル配布に適しており，Drizzle は型安全な SQL 生成で保守性を高める．この組み合わせにより，ローカルファーストなアプリケーション設計を実現できる．

2.7 開発・配布基盤

開発環境には Node.js [88] を基盤とし，Docker [89] と Dev Containers [90] で実行環境を再現可能にする．エディタには VS Code [91] を想定し，Biome [92] で静的解析，Lefthook [93] と commitlint [94] で品質を担保する．

3 関連研究・既存ツール

3.1 Infrastructure as Code の品質・欠陥・保守性に関する研究

Infrastructure as Code の品質に関する研究では，欠陥分類やアンチパターン，コードスメルの体系化が進んでいる．欠陥分類としては Gang of Eight や欠陥特徴量の分析が報告されている [18,20,21,95]．また，アンチパターンやスメルの定義と検出方法に関する研究が進められており，Infrastructure as Code 特有の設計上の問題が指摘されている [19,22,23]．さらに，Infrastructure as Code に潜む状態不整合や運用時の欠陥を分析する研究や，文書とコードの不整合を機械学習で検出する研究が報告されている [96–98]．Infrastructure as Code 言語を跨いでスメルを検出する GLITCH のようなアプローチも提案されている [99,100]．ツール支援の有効性を比較する研究や，DevOps 成熟度に対する Infrastructure as Code の影響を示す事例研究も存在する [101,102]．体系的マッピング研究により，Infrastructure as Code 研究の空白領域や欠陥の分類観点が整理されている [103,104]．これらの研究は，Infrastructure as Code が一般的なソフトウェアと同様に品質課題を抱えることを示す．

3.2 Infrastructure as Code のセキュリティと秘密情報管理

Infrastructure as Code はセキュリティ設定をコード化するため，誤設定の影響が広範囲に及ぶ．セキュリティスメルの分類や実証評価，秘密情報管理の実践指針が提案されている [24–26]．さらに大規

模な脆弱性調査や脅威モデリングの研究も進展し，Infrastructure as Code の安全性確保が重要な課題である [27,105].

Terraform を対象としたセキュリティスメルの研究では，インフラ定義に潜む脆弱な構成が網羅的に整理されている [106]．Infrastructure as Code スクリプト内で共起する不安全パターンの分析も行われ，セキュリティルールの設計が課題となっている [107]．最近ではタクソノミの更新や新たなスメルの追加が提案され，より広い脆弱性クラスを扱う動きがある [108]．また，クラウドプロバイダ間でのセキュリティポリシー採用状況を調べる研究や，DevSecOps 環境でのポリシー自動化の実装事例も報告されている [109,110]．さらに，Infrastructure as Code で構築された環境を対象にした侵入検知やネットワーク保護の研究も進んでおり，運用段階における安全性検証が重視されつつある [111].

3.3 理解容易性・レビュー・運用

Infrastructure as Code の理解容易性やレビューに関する研究では，設計レベルの理解やレビュー実践の実証研究が報告されている [12,28]．また，Infrastructure as Code の研究動向を俯瞰する体系的マッピングや技術レビューも行われており，研究領域の成熟が進んでいる [112,113].

運用面では, 設計レベルのセキュリティ実践が理解可能性に与える影響を測定する研究があり, コードだけでなく図やメトリクスを併用する重要性が指摘されている [114]．さらに，API ドキュメントや Infrastructure as Code との間で生じるドキュメント複製の実態調査も報告されており，設計情報の一元化が課題として浮上している [115]．Ansible の依存関係を抽出しサプライチェーンを可視化する研究もあり，構成の見通しを支えるツールの必要性が示唆される [116]．教育面では，Infrastructure as Code のセキュア開発を対象にした教材の有効性が検証されている [117]．こうした知見は，可視化や編集環境が学習効率とレビュー品質を底上げできる可能性を示す.

3.4 Terraform 特化のデータセット・メトリクス・静的解析

Terraform の HCL コードを対象にしたデータセットやメトリクス研究が進展している．TerraDS は Terraform プログラムを大規模に収集したデータセットであり，解析手法の再現性向上に寄与する [118]．TerraMetrics は Terraform 向けの品質メトリクスを実装した OSS であり，モジュール構造や依存度を定量化する基盤として利用できる [119].

静的解析の観点では，Terraform マニフェストに対する警告の分類や実運用での有効性が検証されている [120]．品質計測の枠組みを包括的に整理した研究も登場しており，Infrastructure as Code の品質評価指標の整備が進んでいる [121]．これらの研究は，本研究の評価設計やテスト指標選定に直接的な示唆を与える．

Terraform 特化研究の分類と評価指標は 表 1 に整理する．

3.5 LLM・自動化・エージェント

近年は LLM を用いた Infrastructure as Code 支援が注目され，コードスメル検出，脆弱性修正，自動生成ベンチマークなどが提案されている [122–124]．複数エージェントによるコード生成や修正，バグ発見支援といった研究も進んでおり, 本研究の将来機能として重要な示唆を与える [125–127]. また, Infrastructure as Code 修復，ファジング，テンプレート再利用を目指す研究も登場しており，品質向上の自動化に向けた基盤が整備されつつある [128–130].

表 1: Terraform 特化研究の分類と評価指標

研究分類	主な研究内容	評価指標
セキュリティと秘密情報管理	・セキュリティスメルの分類と実証評価 ・脆弱性調査と脅威モデリング ・秘密情報管理の実践指針	・スメル検出精度 ・脆弱性カバレッジ ・ポリシー採用率
理解容易性・レビュー・運用	・設計レベルの理解とレビュー実践 ・研究動向の体系的マッピング ・可視化とドキュメント管理	・理解時間 ・レビュー効率 ・ドキュメント整合性
データセット・メトリクス・静的解析	・大規模データセットの構築（TerraDS） ・品質メトリクスの実装（TerraMetrics） ・静的解析警告の分類と有効性検証	・データセット規模 ・メトリクスカバレッジ ・警告の精度と再現率
LLM・自動化・エージェント	・コードスメル検出と脆弱性修正 ・自動生成ベンチマーク ・複数エージェントによるコード生成	・生成品質 ・修正精度 ・実行成功率
実運用事例と応用領域	・大学 IT 基盤の再構築事例 ・CI/CD パイプライン統合 ・マルチクラウド環境での運用	・導入効果 ・運用コスト削減 ・パフォーマンス比較
HCL 解析・変換ツール	・構成の部分編集支援（hcledit） ・HCL-JSON 変換（hcl2json） ・依存関係抽出（terraform-config-inspect）	・解析精度 ・変換正確性 ・コメント保持性

3.6 実運用事例と応用領域

Terraform は研究用途だけでなく, 実際の運用事例においても多様な適用が報告されている. 大学組織の IT 基盤を Terraform とデータ連携ツールで再構築した事例や, CI/CD パイプラインと統合した監視システムの構築事例が報告されており，現場での導入プロセスの課題と効果が示されている [131,132]. 複数クラウドでのビッグデータ処理基盤や, エネルギー効率を考慮したワークロード配置など，運用設計に関わる応用研究も存在する [133,134].

また, Infrastructure as Code 成果物からマイクロサービスパターンを検出する研究や, 構成情報を設計レベルに引き上げる試みも進んでいる [135]．Terraform と AWS CDK の比較や，Terraform と Cloudify のオーケストレーション性能比較など，ツール選択に直結する研究も存在する [136,137]．これらの応用領域は, Infrastructure as Code が単なるプロビジョニングに留まらず, 設計と運用をつなぐ媒介となり得ることを示す.

3.7 HCL 解析・変換ツール

HCL を対象とした解析・変換ツールとして，構成の部分編集を支援する hcledit や，HCL を JSON へ変換する hcl2json, 構成から依存関係を抽出する terraform-config-inspect が存在する [138–140]. これらは既存コードの解析や可視化の前処理に有効であるが，コメントやフォーマットを保持したまま編集することは想定していない. 本研究のロスレス編集は, こうした補助ツールの限界を踏まえた上で，UI 操作とコードの同一性を維持することを目的とする.

3.8 既存の可視化・編集ツール

Terraform 構成の可視化ツールとして, Terraform Visual [14], Inframap [15], Blast Radius [16], Brainboard [13] が存在する．Terraform Visual と Inframap は HCL や state を入力としてグラフ化

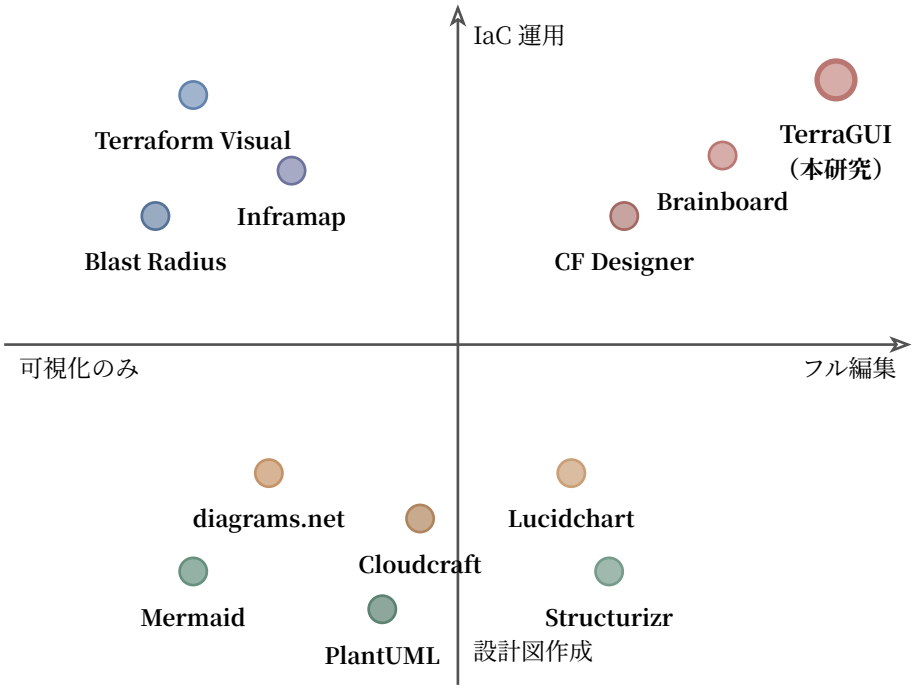

図 2: 既存ツールの位置付けと編集粒度の比較

するが，編集は限定的である．Blast Radius は依存関係の探索に優れるが，GUI 編集には対応しない．Brainboard は GUI 編集をサポートするが独自フォーマットに依存し，既存コードとの整合性確保が課題となる．また，AWS CloudFormation Designer [141] はクラウド固有の可視化機能を提供するが，他クラウドとの統合性は低い．

さらに，クラウド設計やアーキテクチャ図に特化したツールとして Cloudcraft や diagrams.net，Lucidchart があり，学習用途やプレゼン資料作成に利用される [142–144]．Mermaid や PlantUML，Structurizr はテキストベースで図を記述でき，C4 モデルのような抽象レベルの切り替えを支援する [145–148]．これらは設計コミュニケーションに優れる一方，Terraform コードとのロスレスな同期は想定されていないため，開発中のコードと設計図が乖離しやすい．

既存ツールの位置付けと編集粒度の比較は 図 2 に示す．

3.9 本研究との位置付け

既存研究は Infrastructure as Code 品質の観点で有用な知見を提供するが，現場での編集体験を直接改善するツールは限られている．本研究はロスレスな編集と意味的な可視化を同時に実現し，既存の Git ベースの運用と両立する点に新規性がある．

4 提案システム: TerraGUI

4.1 システム概要

TerraGUI（開発コードネーム: TerraGUI）は，ブラウザ上で動作する Terraform 統合開発環境である．ユーザはプロジェクトを作成し，テンプレートから構成を開始するか，既存の HCL ファイルをインポートできる．メイン画面は「アーキテクチャエディタ」と呼ばれ，左側にグラフビュー，右側にプロパティエディタまたはコードエディタを配置した 2 ペイン構成となっている．

システム全体の構成は 図 3 に示す．

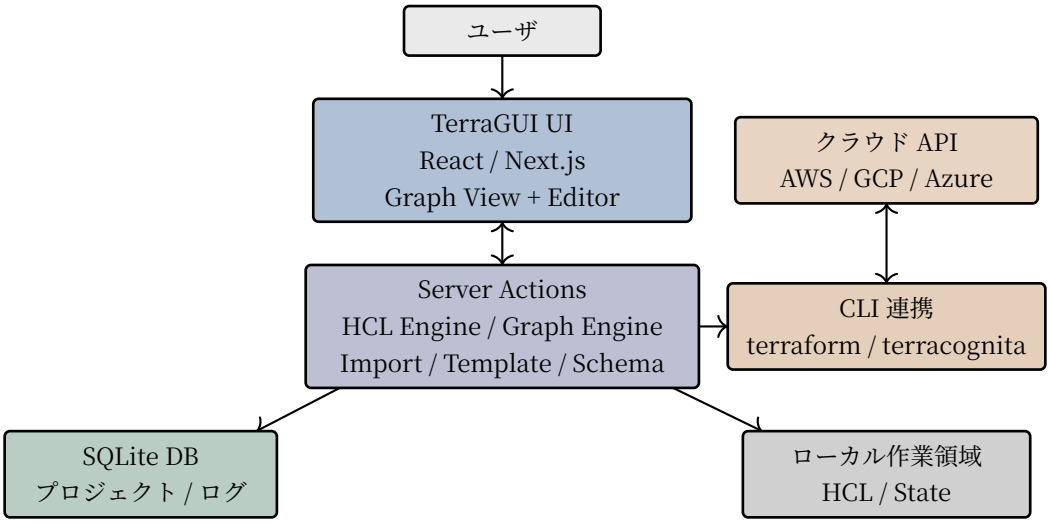

図 3: TerraGUI のシステム構成図

4.2 ユースケースと要件整理

TerraGUI の主な対象ユーザは，Terraform を用いてクラウド構成を設計・運用するエンジニアと，Infrastructure as Code を学ぶ学生である．ユーザの操作シナリオは大きく 3 つに分かれる．

第一に，既存コードの可視化と編集である．既存の HCL ファイルを読み込み，依存関係や構成全体を視覚的に把握しながら編集する．コメントやフォーマットを保持し，レビュー文化を破壊しないことが必須条件となる．第二に，既存環境のインポートである．既存クラウド環境を terracognita によりインポートし，HCL とグラフを生成する [149]．インポート時のログを可視化し，失敗原因を追跡可能にする必要がある．第三に，テンプレートからの開始である．代表的な構成をテンプレートとして提供し，必要なパラメータを入力して新規プロジェクトを作成する．

これらのユースケースから，ロスレス編集，既存ワークフローとの統合，学習容易性と視認性向上，ローカル環境で完結する軽量性の 4 点が要件として導出される．

ユースケースと要件の対応関係は 表 2 にまとめる．

4.3 システムアーキテクチャとデータフロー

システムは，フロントエンド UI，サーバー側アクション，SQLite データベースの 3 層を中心に構成される．フロントエンドでは React Flow と Monaco Editor が連携し，選択状態やハイライトを同期する．サーバー側は HCL を解析しグラフを生成し，architecture_files テーブルに HCL テキストを保存する．インポート処理では一時ディレクトリに HCL と state を生成し，そこから解析する．SQLite は描画高速化と状態管理のための永続ストレージとして利用される．

DB とインポート作業領域のデータフローは 図 4 に示す．

表 2: ユースケースと要件の対応

ユースケース	ロスレス編集	既存ワークフローとの統合	学習容易性と視認性向上	ローカル環境で完結する軽量性
既存コードの可視化と編集	●	●	●	
既存環境のインポート		●	●	●
テンプレートからの開始			●	●

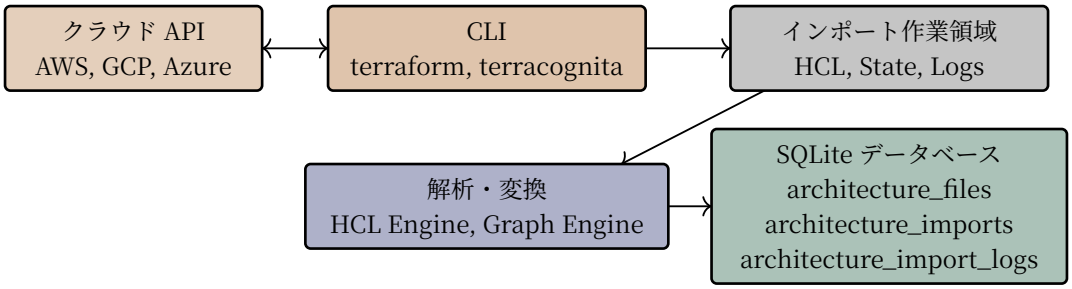

図 4: DB とインポート作業領域のデータフロー

4.4 設計思想

4.4.1 Single Source of Truth as Code

TerraGUI は HCL テキストを単一の真実源として扱い, architecture_files テーブルに保存された内容を基準に動作する．graph_json はあくまで派生データであり，必要に応じて再生成できる．この設計により，UI 上の編集結果は常に HCL テキストに反映され，視覚的な操作とコードの整合性が維持される．

4.4.2 ロスレス編集

GUI ツールが普及しない最大の要因は，勝手にコードを書き換えられることへの嫌悪感である．TerraGUI は，ユーザが意図的に変更したパラメータ以外には一切手を加えない．これを実現するために，HCL の解析と再生成のプロセスにおいて完全な可逆性を保証するアーキテクチャを採用した．

4.4.3 意味的階層化

クラウドインフラストラクチャには, VPC の中にサブネットがあり, サブネットの中にインスタンスがあるといった包含関係が存在する．しかし HCL の構文上は，これらは全て並列な resource ブロックとして記述される．TerraGUI はリソースの属性を解析し，UI 上で自動的に親子関係を構築して表示する．これにより，ユーザはメンタルモデルに近い形でインフラストラクチャを把握できる．

4.4.4 ローカルファーストと拡張性

ローカル環境での利用を前提とし，Docker や Dev Containers で容易に起動できる．また，プロバイダスキーマやアイコンを自動更新する仕組みにより，クラウドサービスの更新に追従可能とする．

意味的階層化の概念は 図 5 で可視化する．

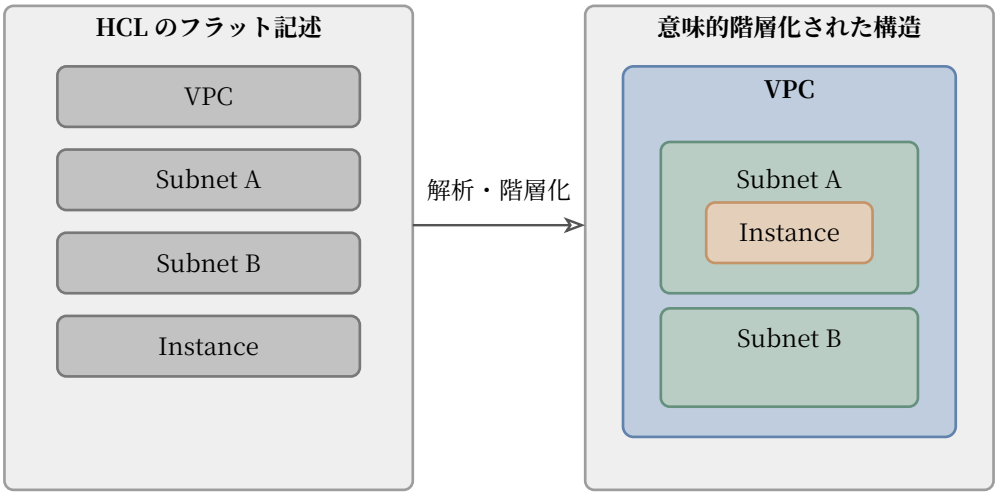

図 5: 意味的階層化の概念図

5 実装詳細

本章では，TerraGUI の中核となる 2 つのエンジン，HCL Engine と Graph Engine の実装詳細について述べる．

5.1 プロジェクト構造とモジュール境界

アプリケーションは Next.js App Router を基盤とし，src/app 配下に画面と Server Actions が配置される．トップページや新規作成フローは src/app/new に集約され，インポートの UI とアクションは src/app/new/import 以下に分離されている．アーキテクチャ編集画面は src/app/architecture/[architecture]にあり，グラフキャンバス，コードエディタ，プロパティエディタなどのコンポーネントが_components 以下に集約される．

コアロジックは src/lib に集約し，HCL の解析とロスレス再生成は src/lib/hcl.ts，グラフ変換と意味的階層化は src/lib/graph.ts が担当する．データベース定義は src/db/schema にあり，プロジェクト情報，テンプレート，インポートログ，プロバイダスキーマが独立したテーブルとして定義される．アイコン生成やデータ初期化のワークフローは src/workflows に配置され，src/workflows/icon.ts がクラウド公式アイコンの収集とマッピングを担う．

プロジェクト構造と主要モジュールの対応は 表 3 に示す．

5.2 HCL Engine の実装

HCL Engine は，src/lib/hcl.ts に実装されており，テキストと具象構文木の相互変換を担う．

5.2.1 トークン定義と字句解析

HCL を構成する要素を Token として定義する．typescript-parsec を用い，正規表現ベースのルールで字句解析する．重要な点は，空白やコメントもトークンとして保持することである．また，HCL の仕様に沿って文字列リテラル，数値リテラル，ブロックコメント，ヒアドキュメントなどを個別トークンとして扱う [32,63]．

表 3: プロジェクト構造と主要モジュールの対応

パス	役割・主要モジュール
src/app	画面と Server Actions の基盤
src/app/new	トップページ／新規作成フロー
src/app/new/import	インポート UI とアクション
src/app/architecture/[architecture]	アーキテクチャエディタ画面
src/app/architecture/[architecture]/_components	グラフキャンバス／コード／プロパティ
src/lib	共有ユーティリティとドメインロジック
src/lib/hcl.ts	HCL 解析とロスレス再生成
src/lib/graph.ts	グラフ変換と意味的階層化
src/db/schema	DB スキーマ（プロジェクト／テンプレート／ログ／スキーマ）
src/workflows	ワークフロー定義と自動化
src/workflows/icon.ts	クラウド公式アイコンの収集とマッピング

```
export enum HclTokenKind {
  Whitespace,
  LineComment,
  HashComment,
  BlockComment,
  Identifier,
  NumberLiteral,
  BooleanLiteral,
  NullLiteral,
  StringLiteral,
  HeredocLiteral,
  Equals,
  Colon,
  // ...
}
```

本実装では，文書全体を対象とするレキサーと，式解析専用のレキサーを分離している．前者は空白・コメントをトークンとして保持し，後者は式の評価に不要なトークンをスキップする設計である．これにより，属性値の境界を正確に特定しつつ，ロスレス編集に必要な文書全体のトークン列を維持できる．Heredoc や文字列リテラルのデコード処理も字句解析段階で扱い，後段の式評価で正規化が進み過ぎないように調整している．

5.2.2 構文解析と具象構文木構築

字句解析されたトークン列から，HclDocument として具象構文木を構築する．具象構文木はノードの配列であり，各ノードは元のテキスト上の位置範囲を持つ．parseBody 関数はトークン列を逐次走査し，属性とブロックを判定する．この過程では，行末のコメントや改行の扱いを考慮し，属性値の境界を厳密に検出する．

HCL トークンから CST 生成までの処理フローは 図 6 に整理する．

この構造により，例えば ami 属性の値を書き換える操作は，Attribute ノードの expression 部分に対応するトークンを置換する操作として実装できる．それ以外のトークンはそのまま維持されるため，ロスレスな編集が可能となる．

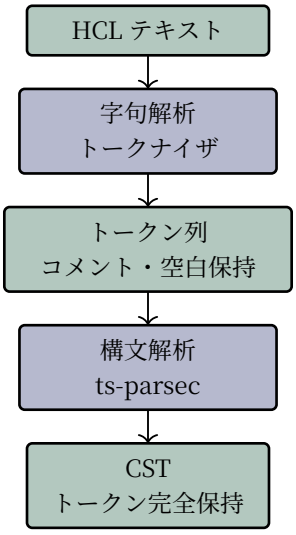

図 6: HCL トークンから CST 生成までの処理フロー

5.2.3 式構文と演算子優先順位

HCL の式は算術演算子，論理演算子，条件演算子，nullish 演算子など多様な構文を含む [36]．実装では lrec_sc を用いた左再帰パーサを多段に構成し，算術演算子，比較演算子，論理演算子，条件演算子といった優先順位を明示的に表現している．また，タプル/オブジェクト，関数呼び出し，インデックス参照，属性アクセス，スプラット式までを式ノードとして扱うことで，Terraform の評価モデルに沿った構文木を構成する．

for 式や条件式は，リソース定義の属性値だけでなく dynamic ブロックによる繰り返し構造にも影響するため，GUI 編集での参照解決や依存抽出にも影響する [37]．TerraGUI では式ノードを保持したままグラフ解析に渡すことで，依存関係抽出とロスレス編集の両立を図っている．

5.2.4 差分適用とロスレス再生成

編集時には，対象ブロックのトークン範囲のみを書き換え，それ以外はそのまま保持する．値更新は valueRange を基準に行い，必要に応じて未定義属性を末尾へ挿入する．astToHcl はトークンを連結するだけで再生成できるため，元の書式が維持される．この方式により，GUI 側の編集が最小差分としてコードに反映される．

具体的には，ブロック内部のトークンを複製し，更新対象の値トークンだけを書き換え，残りのトークンを空文字にすることで差分の最小化を実現する．新規属性は末尾の閉じ括弧を検出して適切なインデントを推定し，既存のコードスタイルを崩さないように挿入される．このアプローチにより，レビュー時に差分が最小化され，チーム内のコードスタイルの一貫性が保たれる．

5.3 Graph Engine の実装

Graph Engine は src/lib/graph.ts に実装されており，具象構文木からグラフ構造への変換を担う．

5.3.1 ノード変換と親子関係の構築

HCL の resource ブロックを解析し，React Flow のノードへ変換する．リソースは resourceType と name の組をアドレスとして扱い，同一アドレスが複数存在する場合は識別子にサフィックスを付与する．

ここで最も重要な処理が親子関係の構築である．HCL 上ではフラットに記述されているリソースに対し，以下の優先順位でグルーピングを行う．最優先は Availability Zone であり，availability_zone 属性を持つリソースが対象となる．次に Subnet であり，subnet_id 属性を持つリソースが対象となる．続いて VPC であり，vpc_id 属性を持つリソースが対象となる．最後に Provider/Region であり，プロバイダ設定に基づくリージョンが対象となる．

この際，collectReferencedResourceIds により参照先リソースを解決し，親候補と優先度を比較して決定する．また，Availability Zone からリージョンを逆算し，プロバイダ設定が省略されている場合でもグルーピングできるようにしている．

プロバイダブロックは独立ノード化され，リージョンが指定されている場合は仮想グループが生成される．これにより，VPC やサブネットが存在しない構成でも，プロバイダ単位の階層を維持できる．セキュリティグループやサブネットグループといった実体のある集約リソースは実グループとして扱い，Availability Zone やリージョンは仮想グループとして扱うことで，論理的な境界と物理的な境界を視覚的に区別する．

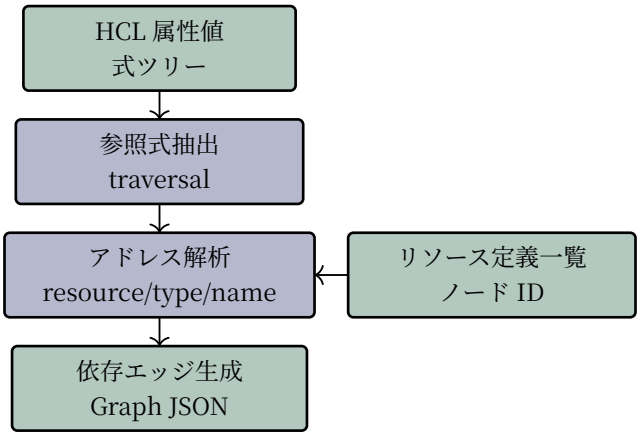

図 7: 参照式と依存エッジ生成の処理

5.3.2 依存関係抽出

依存関係は 2 種類に分けて抽出する．第一に，depends_on に明示的に列挙された依存である．第二に，属性値内の参照式である．

参照式の抽出には，式木からアクセスチェーンを辿る unwrapAccessChain を用いる．配列やオブジェクトのネストを含む場合でも，式木を走査して参照先 ID を収集する．

depends_on は Terraform のメタ引数として明示的な依存関係を表すため，Graph Engine では最優先のエッジ生成対象として扱う [38]．一方で，式内参照は暗黙的な依存であり，関数呼び出しや条件式を含む場合もあるため，式木を再帰的に探索して参照先を抽出する．この処理により，単純な属性参照だけでなく，配列のインデックスアクセスや for 式による集合生成など，実運用で頻出する表現に対しても依存関係を補足できる．

参照式と依存エッジ生成の処理は 図 7 に示す．

5.3.3 エッジ生成と自動レイアウト

依存関係を表すエッジは，depends_on 属性や参照式から生成される．生成されたノードとエッジの配置計算には elkjs を使用している．ELK の layered アルゴリズムを採用し，上から下へのレイアウトやネスト構造の維持といったオプションを設定することで，複雑な依存関係を持つグラフでも交差が少なく，視認性の高い配置を実現している [34]．

レイアウト結果は React Flow の座標系に変換され，エッジの曲がり点（bend points）やラベル配置も保持される．エッジラベルは依存関係のインデックスや参照経路を表現するために利用し，視覚的な説明性を補強する．また，グループノードには内側のパディングや最小サイズを設定し，子ノードが重ならないように調整している．

5.3.4 ノード種別と UI への写像

UI 上ではノードを resource，group-real，group-virtual に分類する．VPC や Subnet など，実体としてのまとまりを持つリソースは group-real として描画し，Availability Zone や Region など論理的な区分は group-virtual として表現する．この区別により，物理的リソースと論理的境界を視覚的に区別できる．

さらに，ドキュメント全体のトークン列を保持する document ノードと，プロバイダ設定を表す provider ノードを内部的に管理する．document ノードは順序情報や原文テキストを保持し，ロスレス再生成に不可欠なメタデータを担う．provider ノードはリージョン推論やグルーピングに利用され，UI 上ではグループと同等の扱いで折り畳み可能な構造として扱われる．

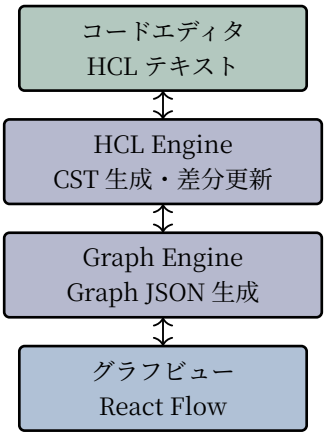

図 8: コードとグラフの双方向同期

5.4 Text-Graph Sync

グラフ上のノードとコードエディタの対応付けは，トークン位置情報を用いて実現する．各ブロックの headerRange をオフセットとして記録し，Monaco Editor 上のカーソル移動に応じて該当ノードをハイライトする．逆方向の同期では，ノード選択時にコードエディタをスクロールし，対応するブロックを中央に表示する．

コードとグラフの双方向同期は 図 8 で整理する．

コード側の編集が発生した場合は，hclToAst で再解析した後，astToReactFlow により再度グラフを構築する．この際，既存のノード位置を mergeGraphWithExisting により引き継ぎ，視覚的な配置が不用意にリセットされないようにしている．逆にグラフ側の編集は reactFlowToAst を介して HCL へ反映し，astToHcl によりトークンを再結合することでロスレスな反映を担保する．

5.5 プロパティエディタの実装

プロパティエディタは，選択されたリソースの属性を編集するための UI である．

5.5.1 スキーマ駆動フォーム

Terraform のリソースには数百の属性が存在しうるため，これらを静的に定義できない．本システムでは，バックエンドで Terraform プロバイダのスキーマ情報を取得し，それをフロントエンドに送信する [31]．フロントエンドでは，スキーマ情報に基づいて，動的にフォームフィールドをレンダリングする．

```
{field.input === "boolean" ? (
  <Switch ... />
) : field.input === "textarea" ? (
  <Textarea ... />
) : (
  <Input ... />
)}
```

属性型の判定には bool，number，string といったスキーマ型を利用し，配列や複合型はテキストエリアとして扱う．さらに，既存の HCL から属性値を抽出し，初期値としてフォームに反映することで，コードとフォームの同期を維持する．

変更内容は react-hook-form で管理され，デバウンス処理を経て，Server Actions 経由で HCL ファイルに書き込まれる [83]．スキーマ取得は fetchProviderSchema を通じて行われ，取得結果はクライアント側でキャッシュされるため，同一プロバイダのリソース編集時に再取得が発生しにくい．

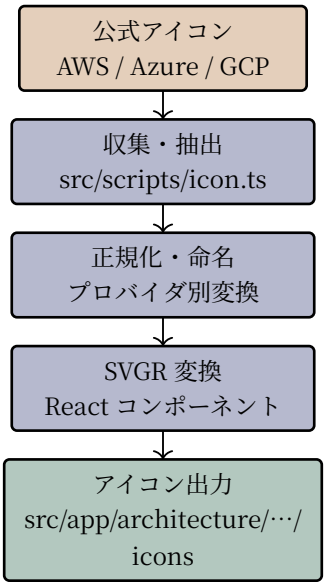

図 9: アイコン自動生成パイプライン

5.6 アイコン自動生成

クラウドリソースは数千種類に及ぶため，手動でのアイコン対応付けは困難である．本システムでは src/workflows/icon.ts を用い，Amazon Web Services，Google Cloud Platform，Microsoft Azure 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 [150–152]. 具体的には, Terraform CLI の providers schema -json によってプロバイダのリソース一覧を取得し，リソース名をトークン化してアイコン名と照合する [31].

スコアリングでは一致トークン数を基本とし，先頭トークン一致やファイル名の連結一致を加点する．解決したアイコンは map.json に保存し，UI 側ではリソースタイプ名から即座にアイコンパスを決定できる．この方式により，プロバイダの更新やリソース追加にも柔軟に追従できる．

アイコン自動生成のパイプラインは 図 9 に示す.

5.7 データベース連携

SQLite を用いてプロジェクト情報, アーキテクチャ, インポートログ, プロバイダスキーマを保持する．architectures テーブルはプロジェクトメタデータを保持し，architecture_files テーブルが HCL コードの本体を保持する．architecture_imports テーブルと architecture_import_logs テーブルはインポートの状態とログストリームを管理し，provider_schemas テーブルはプロバイダスキーマをバージョン付きでキャッシュする. テンプレート機能は templates テーブル, template_parameters テーブル，template_tags テーブルに分割され，テンプレートの入力仕様を正規化して保持する．

architectures テーブルの graph_json は描画キャッシュとして保存されるため，グラフの再レンダリングが高速化される．保存時にはトランザクションを用いて graph_json と architecture_filesテーブルの更新を同期し，UI とコードの整合性を確保する．

6 ユーザインターフェースとワークフロー

本章では，TerraGUI が提供する主要な機能と UI について紹介する．

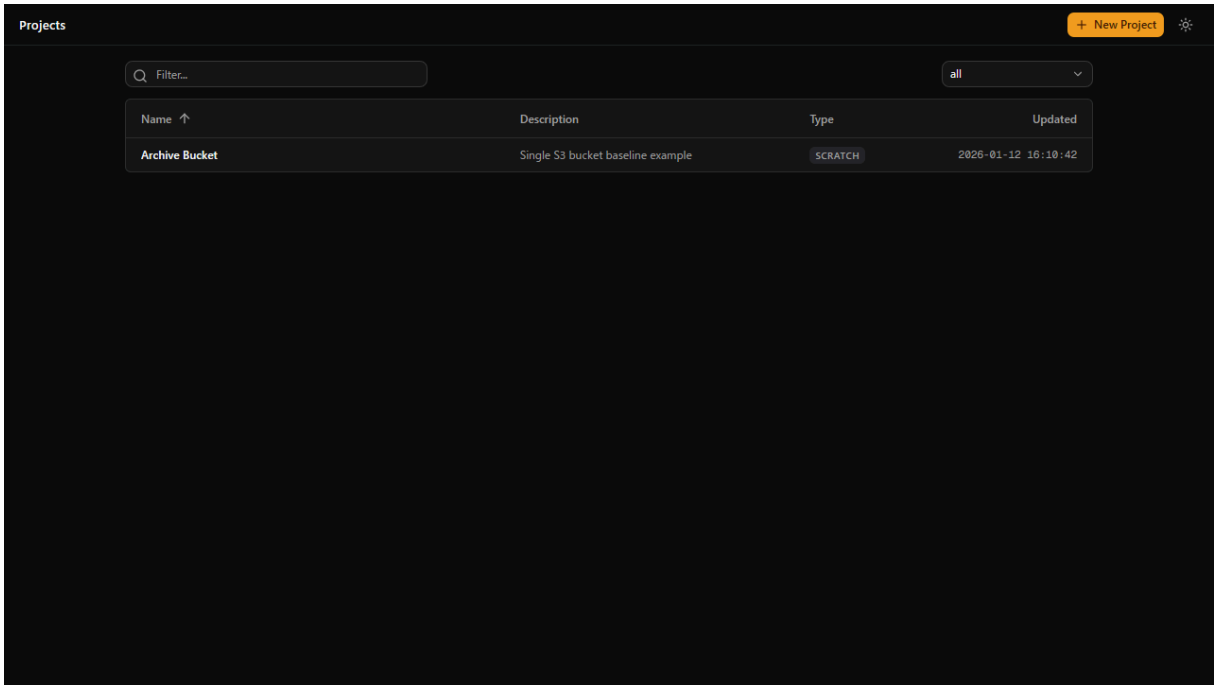

図 10: トップページとプロジェクト一覧

6.1 画面構成とナビゲーション

トップページではプロジェクト一覧を表示し，新規作成，テンプレート作成，既存環境インポートへの導線を提供する．各プロジェクトは名称と概要を持ち，選択するとアーキテクチャエディタへ遷移する．

トップページとプロジェクト一覧の構成は 図 10 で確認できる．

一覧画面には検索バーとタイプフィルタが用意され，プロジェクト名や説明文に対する部分一致検索が可能である．更新日時のソートやタイプ別の絞り込みにより，プロジェクト数が増加しても必要な構成を迅速に特定できる．

6.2 プロジェクト作成とテンプレート

プロジェクト作成フローは，テンプレート起点・インポート起点・スクラッチ起点の 3 方式を提供する．それぞれの選択肢は新規作成画面で一覧化され，初心者でも迷わず導線を辿れるように設計されている．

6.2.1 新規作成の入口

新規作成画面では「From Templates」「From Existing Infra」「From Scratch」をカード形式で表示し，目的に応じた入口を明確にする．ここでの選択は後続のフォームやインポート処理に直結するため，UI 上で選択肢の説明文を明示することでミスを防ぐ．

新規作成の入口画面は 図 11 に示す．

6.2.2 テンプレート一覧

テンプレート一覧では検索とフィルタが可能であり，テンプレート名・概要・タグを横断的に検索できる．プロバイダ別の絞り込みと作成日時のソートにより，多数のテンプレートから目的の構成を素早く見つけられる．テンプレートは DB 上の templates テーブルと template_tags テーブルによって管理されるため，タグ分類を柔軟に拡張できる．

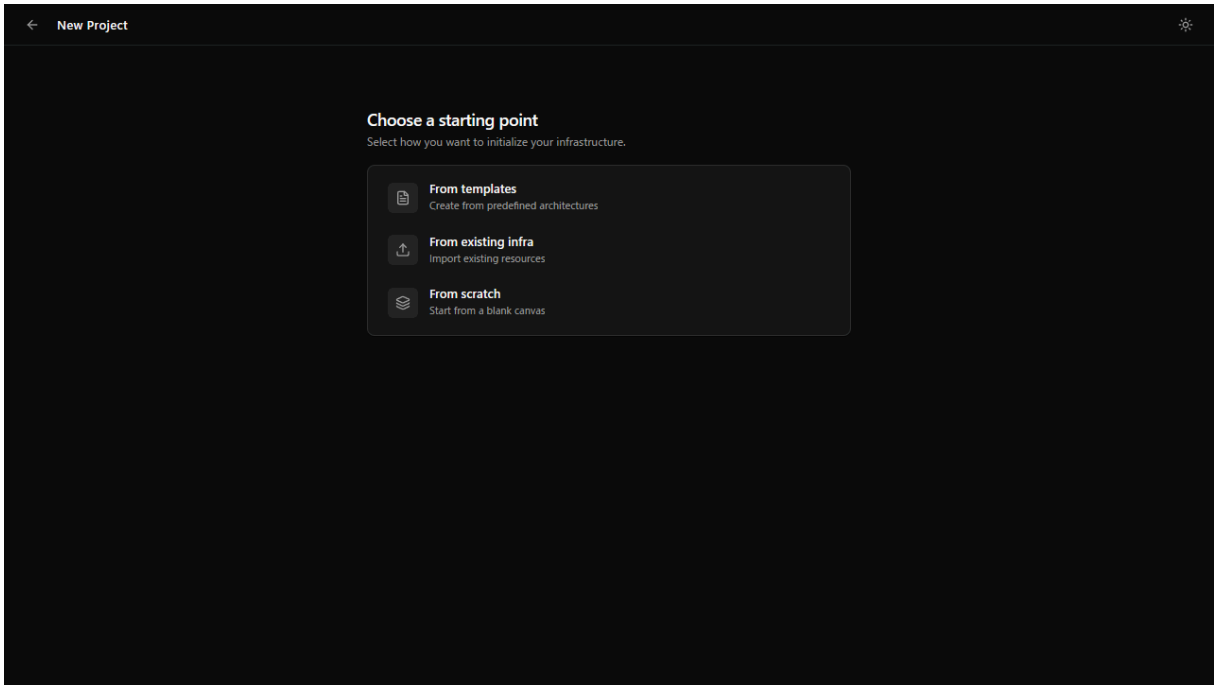

図 11: 新規作成の入口画面

テンプレート一覧と検索フィルタの構成は 図 12 に示す.

6.2.3 テンプレート詳細設定

テンプレート詳細画面では，パラメータフォームを通じて環境名や CIDR，リージョンなどを入力する．入力フォームは react-hook-form と zod によって検証され，必須項目や形式不一致を即座に通知できる [83,84]．テンプレートの説明文やタグも同時に表示されるため，構成内容を理解した上で作成できる．

テンプレート詳細とパラメータ入力の画面は 図 13 に示す.

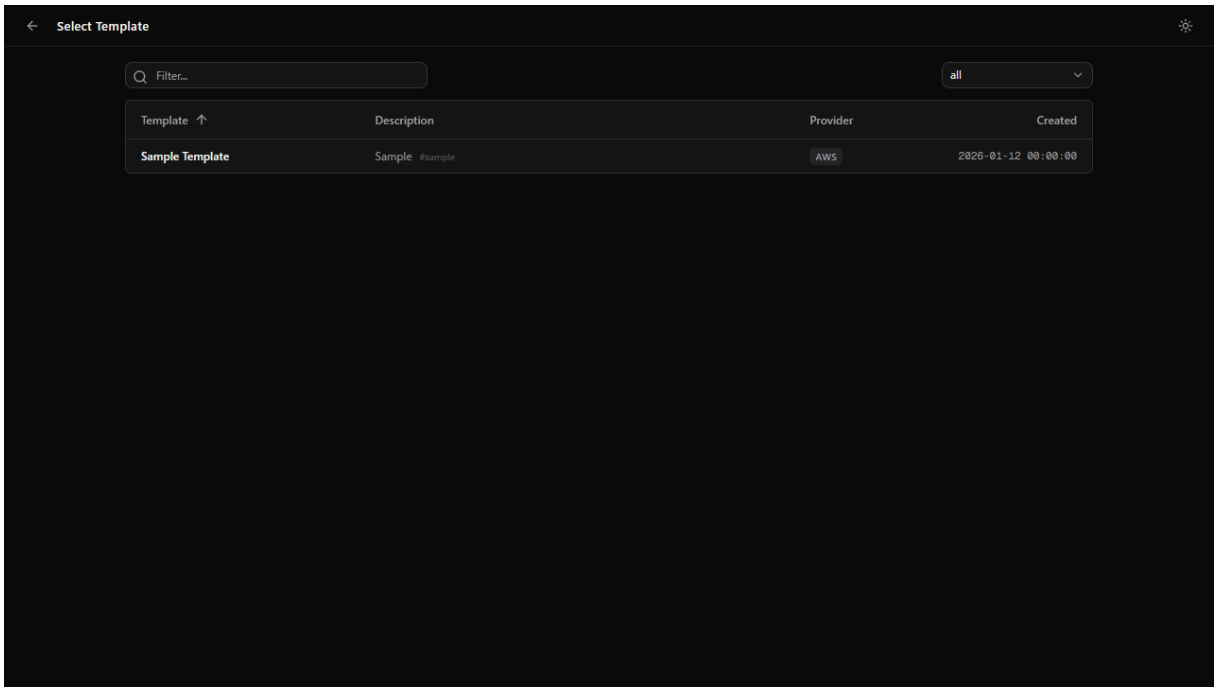

図 12: テンプレート一覧と検索フィルタ

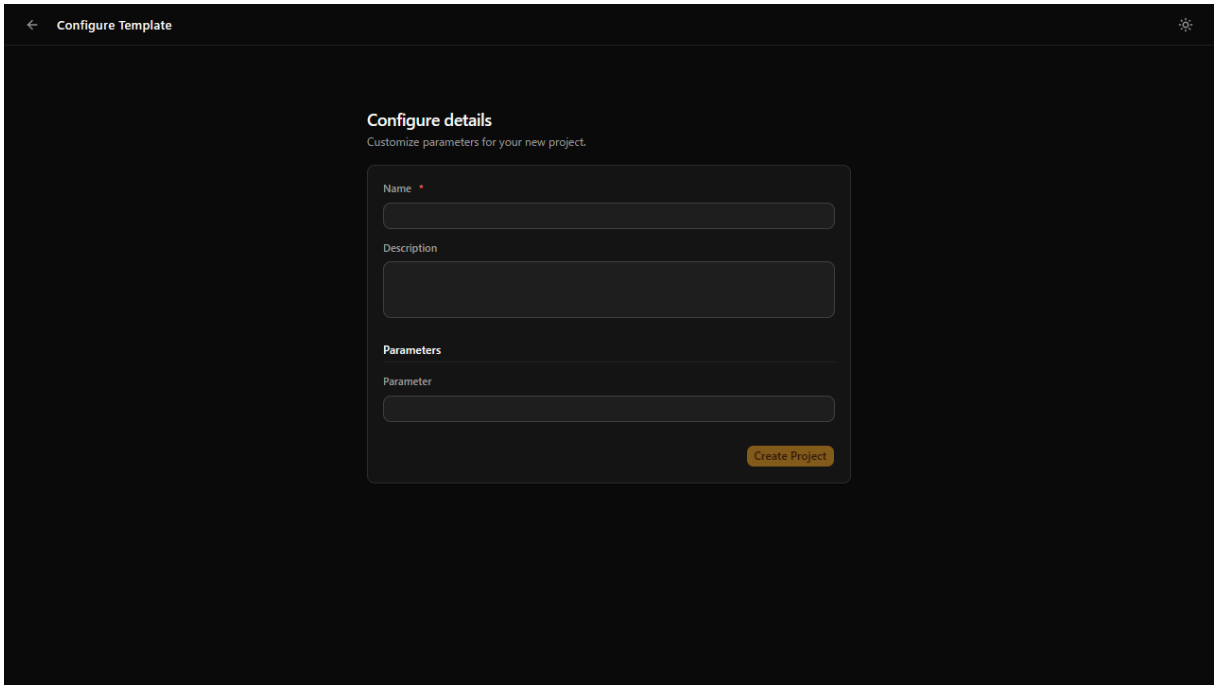

図 13: テンプレート詳細とパラメータ入力

6.2.4 スクラッチ作成

スクラッチ作成は最小限のフォーム入力で開始できる．名称と概要のみを指定し，空の構成を生成することで，試行錯誤しながらリソースを追加するワークフローを支援する．

スクラッチ作成フォームの構成は 図 14 に示す．

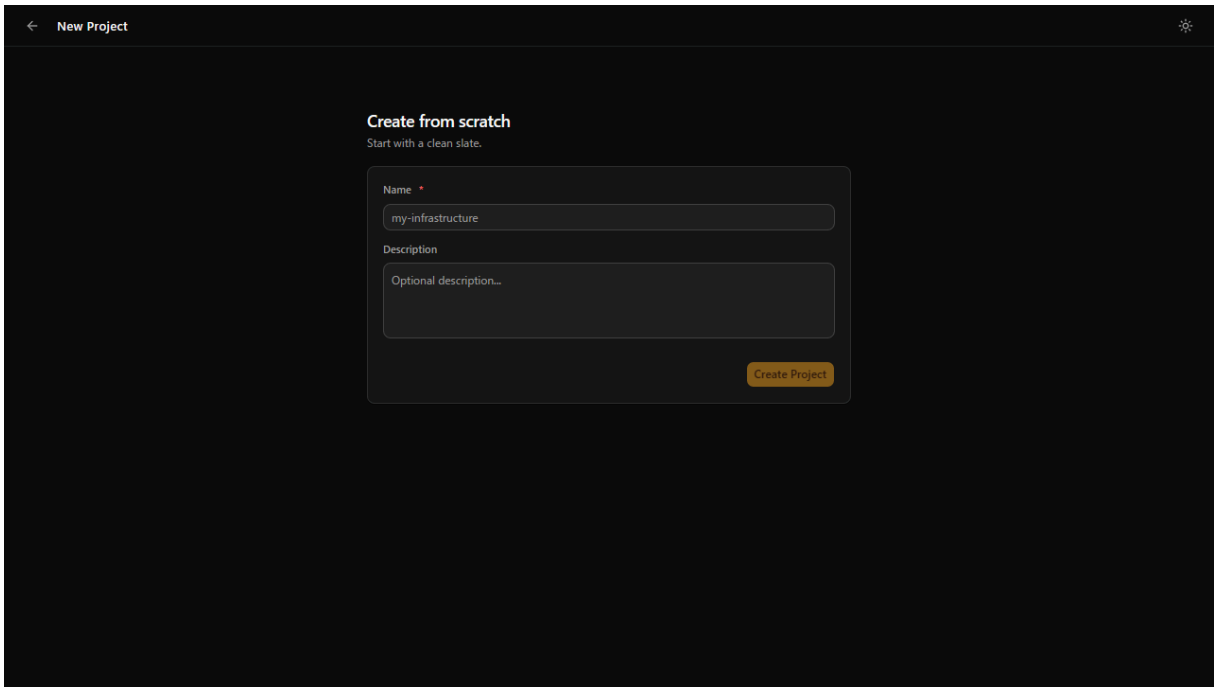

図 14: スクラッチ作成フォーム

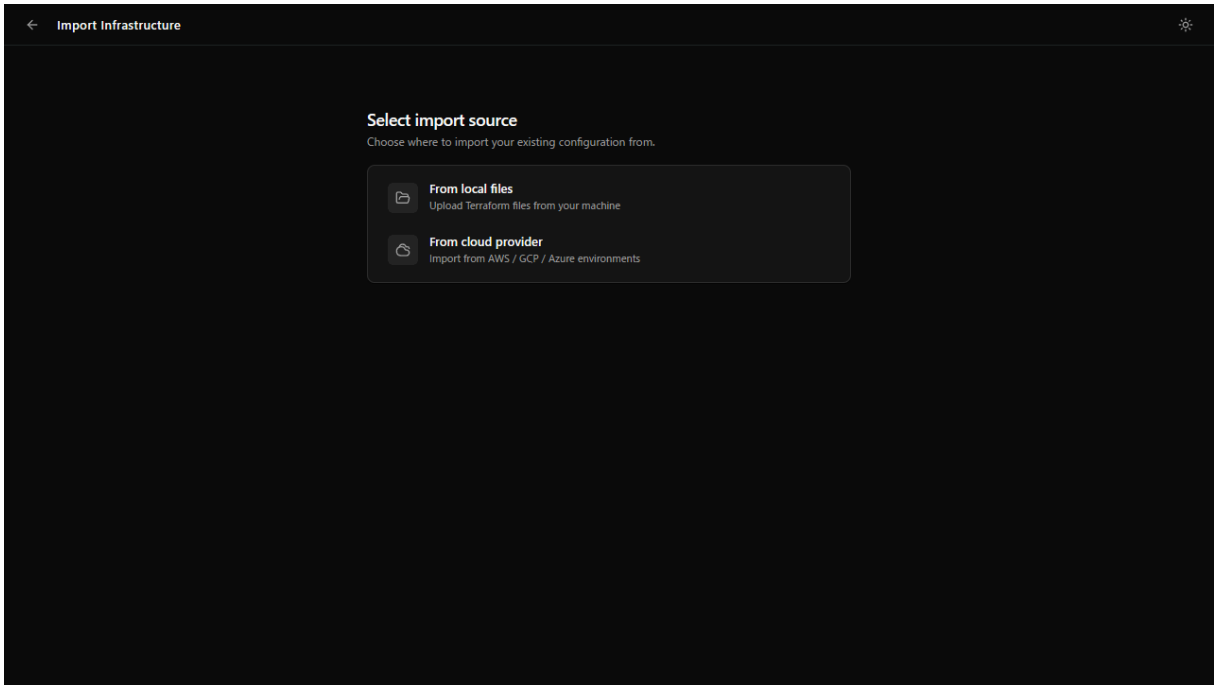

図 15: インポート元の選択画面

6.3 既存環境のインポート

インポートは「ローカルファイルからの取り込み」と「クラウド環境からの取り込み」に分かれる．どちらもプロジェクト作成フローの一部として実行され，完了後は自動的にアーキテクチャエディタへ移行する．

6.3.1 インポート元選択

インポート元選択画面では，ローカルファイルとクラウドインポートの 2 択を提示する．クラウドインポートは terracognita とクラウド CLI の可用性が確認できた場合のみ表示されるため，環境差によるエラーを事前に抑制できる [149,153–155].

インポート元の選択画面は 図 15 に示す.

6.3.2 ローカルファイルインポート

ローカルインポートでは.tf と.tfvars を複数選択し，プロジェクト名と概要を入力してアップロードする．react-hook-form と zod で入力を検証し，ファイル未選択時にはエラーを表示する [83,84]．設計上はインポート後に HCL を解析しグラフを生成し，最初の編集状態を自動生成することを想定している．現段階では UI と入力検証の整備を優先し，ファイル内容の取り込みは今後の拡張課題とする．

ローカルファイルの選択と検証の流れは 図 16 に示す.

6.3.3 クラウドインポート

クラウドインポートでは Amazon Web Services，Google Cloud Platform，Microsoft Azure のいずれかを選択し，プロバイダごとに必要な認証情報を入力する．Amazon Web Services はアクセスキー方式とプロフィール方式を切り替えられ，Google Cloud Platform はサービスアカウントキー，Microsoft Azure はサービスプリンシパル方式を採用する．入力値は CLI を用いて検証し，認証が通った場合のみ terracognita を実行する [149,153–155].

クラウドインポートの入力フォームは 図 17 に示す.

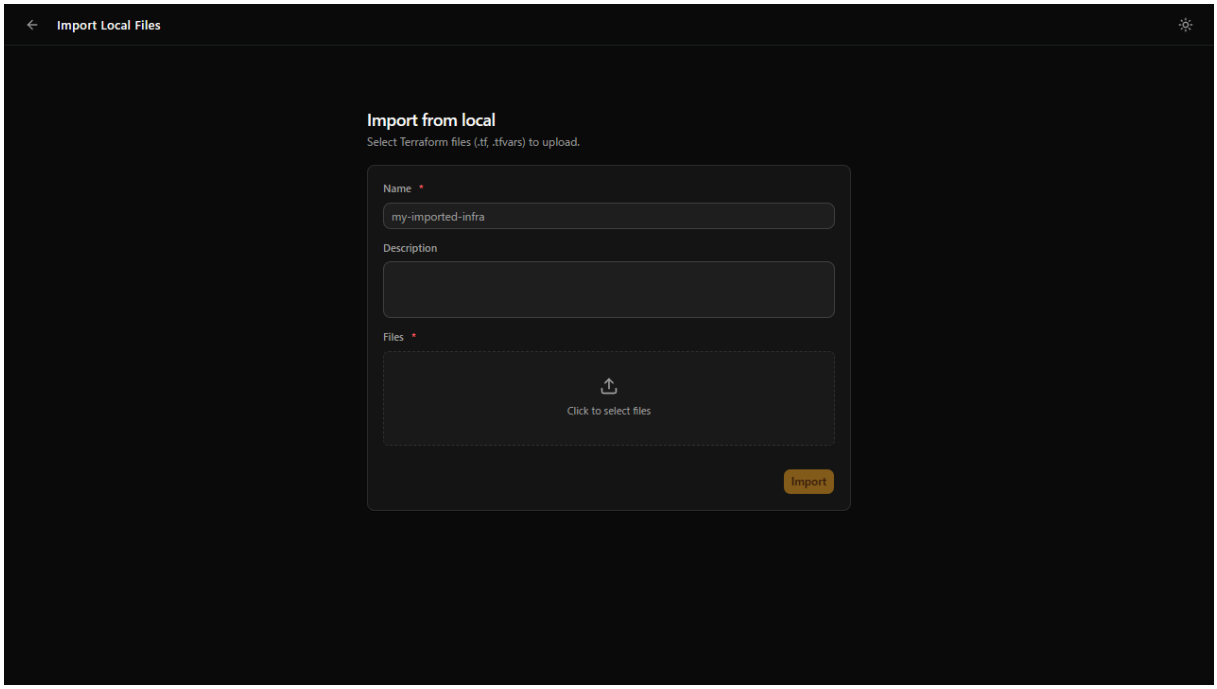

図 16: ローカルファイルの選択と検証

6.3.4 インポートログ監視

インポート実行中はログパネルに切り替わり，標準出力/標準エラーのログがストリームとして表示される．ステータスバッジにより進行状態を可視化し，成功時は自動的にエディタ画面へ遷移する．ログの自動スクロールとエラーメッセージの強調表示により，失敗時の原因特定を支援する．

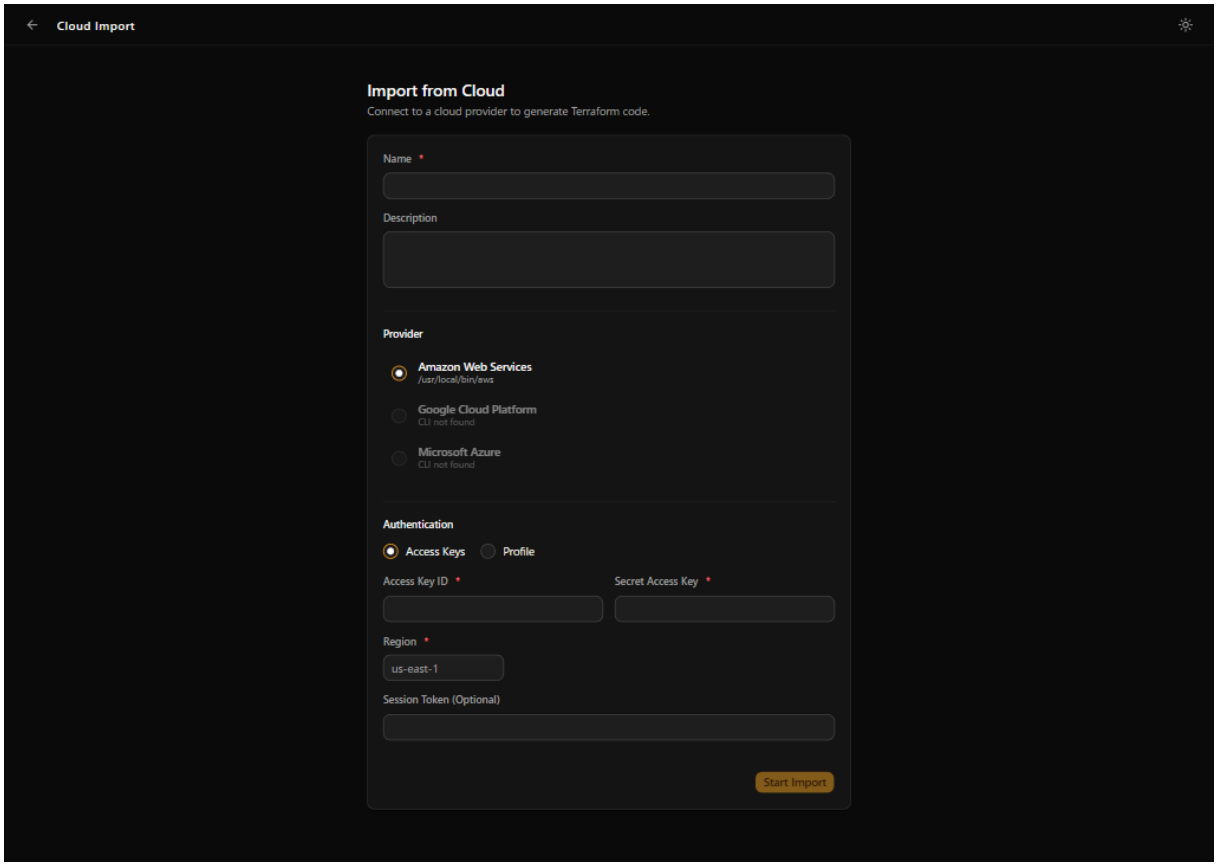

図 17: クラウドインポートの入力フォーム

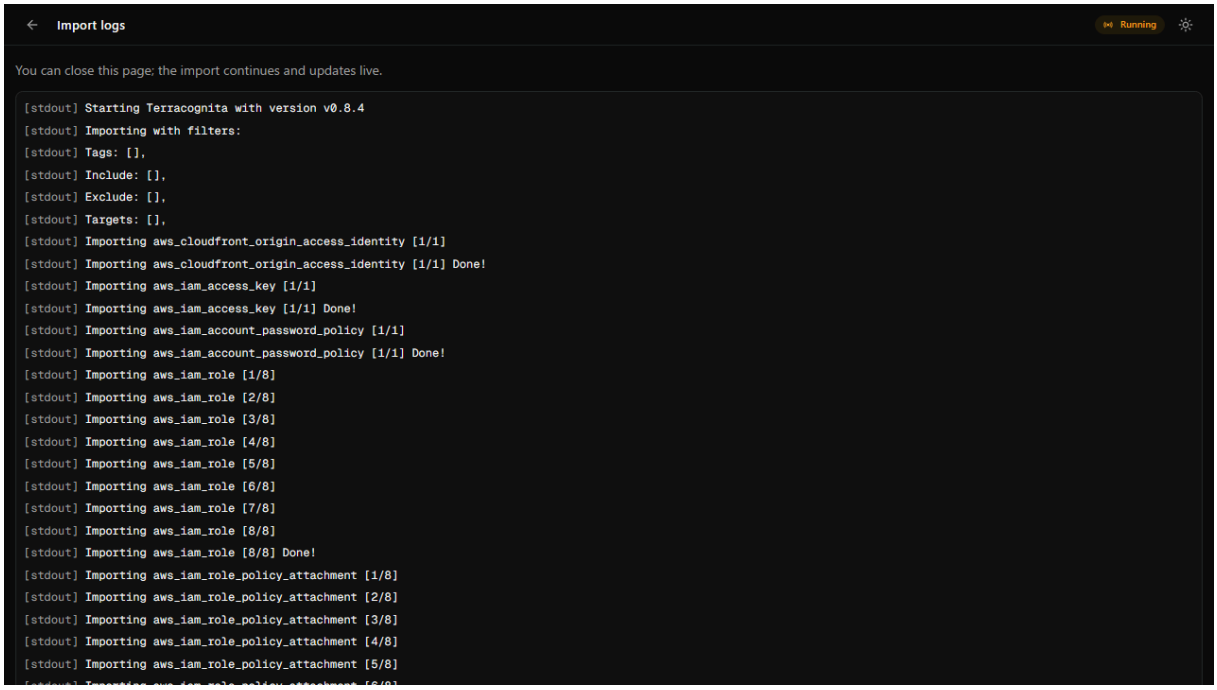

図 18: インポートログとステータス表示

インポートログとステータス表示は 図 18 に示す.

6.4 アーキテクチャエディタ

メイン画面は, グラフ操作を中心とした IDE のような構成である. 左側にはリソースノードが表示される領域である Canvas があり，ズームやパンが可能である．右側には Detail Panel があり，プロパティエディタやコードエディタを切り替えて表示する.

レイアウトは Allotment により左右 2 ペインで構成され，ユーザは編集対象に応じてパネル幅を調整できる [82]．キャンバスではスペースキーによるパンモードが有効になり，ノードのドラッグとビュー操作を切り替えられる. 選択中のノードは右ペインに反映され, プロパティ編集とコード編集をワンクリックで切り替えられる.

保存状態はヘッダのステータス表示で可視化される. 自動保存が走ると「Saving…」が表示され, 完了すると「Auto-saved」に切り替わるため，ユーザは明示的な保存操作を意識せずに作業できる.

アーキテクチャエディタの全体像は 図 19 に示す.

6.5 カスタムノードとアイコン

グラフ上のノードは，React Flow のカスタムノードとして実装されている [74]．各リソースアイコンは, Amazon Web Services, Google Cloud Platform, Microsoft Azure の公式アイコンセットを使用しており，リソースタイプ名から動的にアイコンファイルを解決して表示する [150–152]．また，グループノードは, 内部に他のノードを含むコンテナとして描画され，ドラッグ操作でグループごと移動できる.

仮想グループ（リージョンや Availability Zone）も同一のレイアウト体系で扱い, UI 上では背景色とラベルで区別する．これにより，物理的なネットワーク階層と論理的なスコープが同時に把握できる.

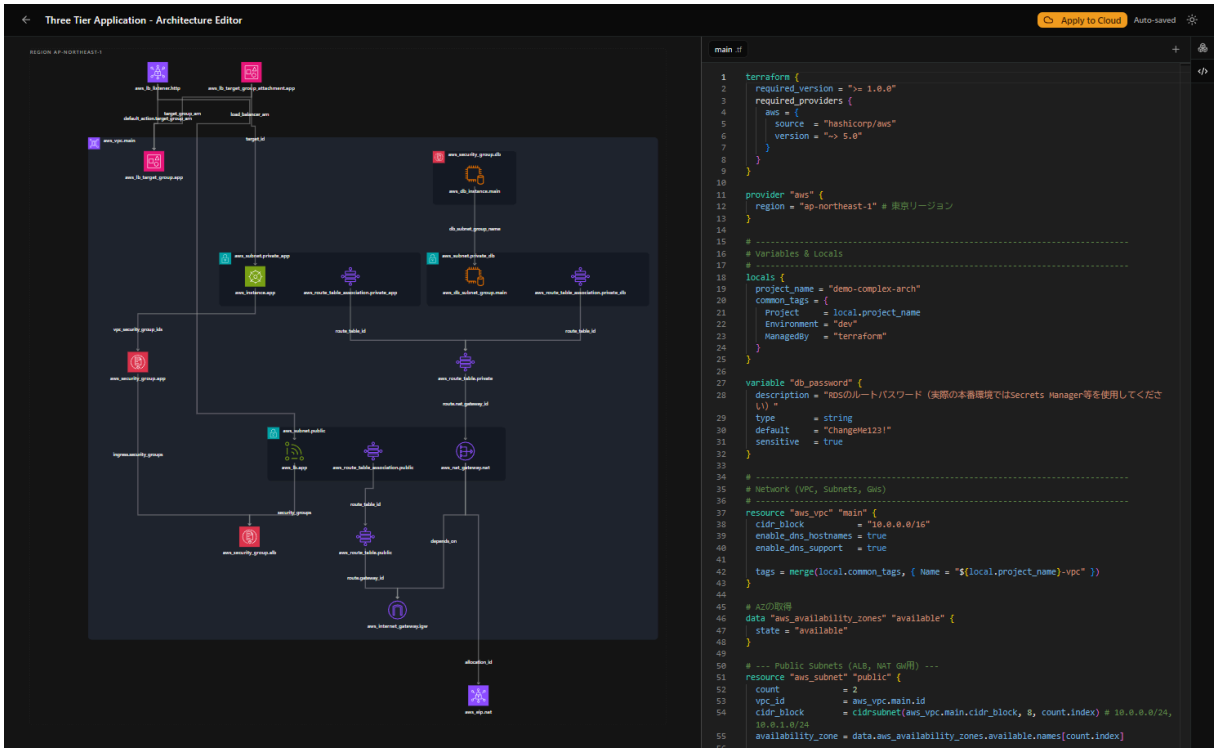

図 19: アーキテクチャエディタの全体像

6.6 コード連携 (Text-Graph Sync)

グラフ上のノードを選択すると，右側のコードエディタが自動的にスクロールし，該当するリソースブロックにハイライト表示する．逆に，コードエディタ上でカーソルを移動すると，グラフ上の対応するノードがフォーカスされる．この機能により，ユーザは「今どのリソースを編集しているか」を常に意識することなく，グラフィカルな操作と詳細なコード編集を行き来できる．

コードエディタとの同期表示は 図 20 に示す.

Monaco Editor のスクロールとハイライトは，ブロックのトークン位置から計算されるため，コメントや空行が混在する現実的なコードでも精度が高い．ノード選択とコード選択が連動することで，レビュー時のトレーサビリティが向上し，「このコードがどのノードか」を視覚的に確認できる．

6.7 プロパティエディタ

プロパティエディタはスキーマ駆動でフォームを生成し，必要項目，型，説明文を表示する．属性の編集結果はデバウンス付きで保存され，最小差分として HCL テキストに反映される．

プロパティエディタの動的フォームは 図 21 に示す.

フォームには必須/任意の区別や説明文が表示されるため，Terraform の詳細仕様を参照しなくても編集できる．ブール値にはトグルを用い，数値・文字列・複合型は入力形式を切り替えることで，初心者にも理解しやすい UI となっている．

6.8 Apply / Destroy（クラウド反映）

HCL 内にプロバイダブロックが存在する場合，ヘッダ右上に「Apply to Cloud」ボタンが表示される．state が検出できる場合は「Destroy from Cloud」も有効になり，既存リソースの破棄フローへ進める．ユーザ操作としては「Apply to Cloud」→ 認証情報入力 →「Plan」→ 差分確認 →「Apply」という二段階の流れになる．

図 20: コードエディタとの同期表示

認証情報入力モーダルでは，AWS/GCP/Azure のうち利用中のプロバイダのみが表示され，必要な認証情報を入力する．入力後に「Plan」を押すと，同一モーダル内で Plan ログがストリーミングされ，Plan の要約（追加・変更・削除件数）と Resource Changes 一覧が表示される．ユーザは差分内容を確認した上で「Apply」を実行する．

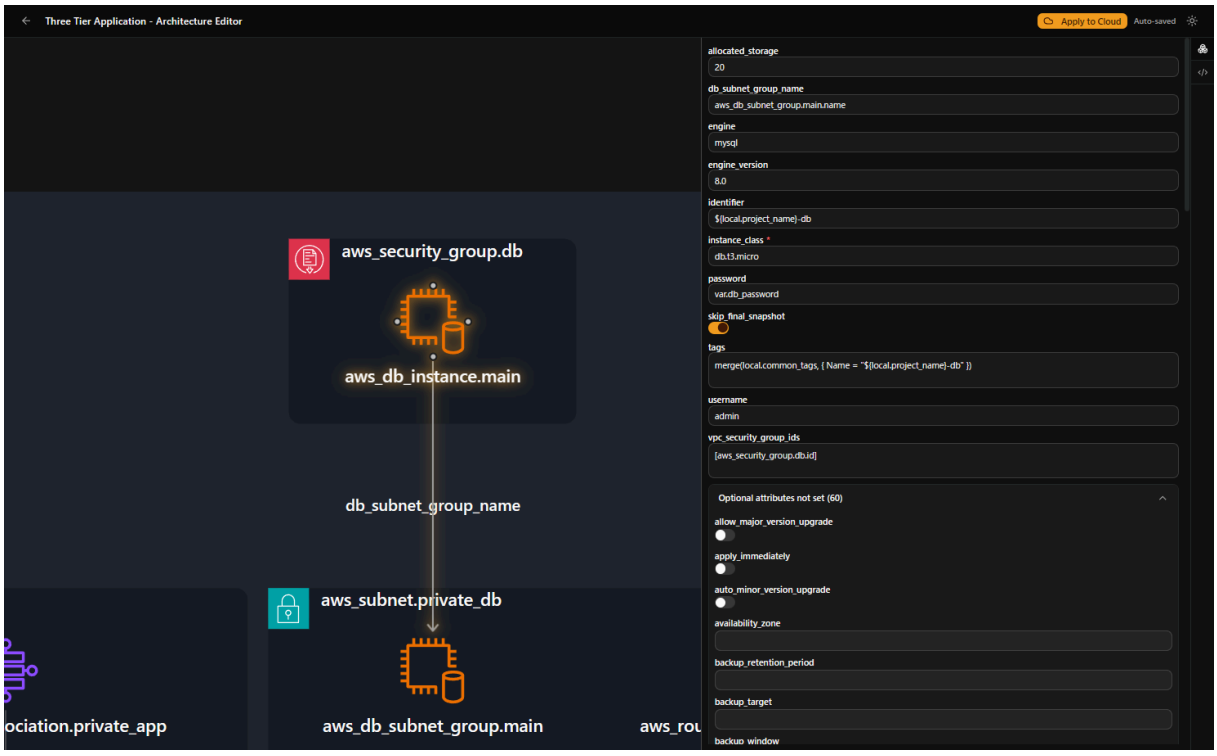

図 21: プロパティエディタの動的フォーム

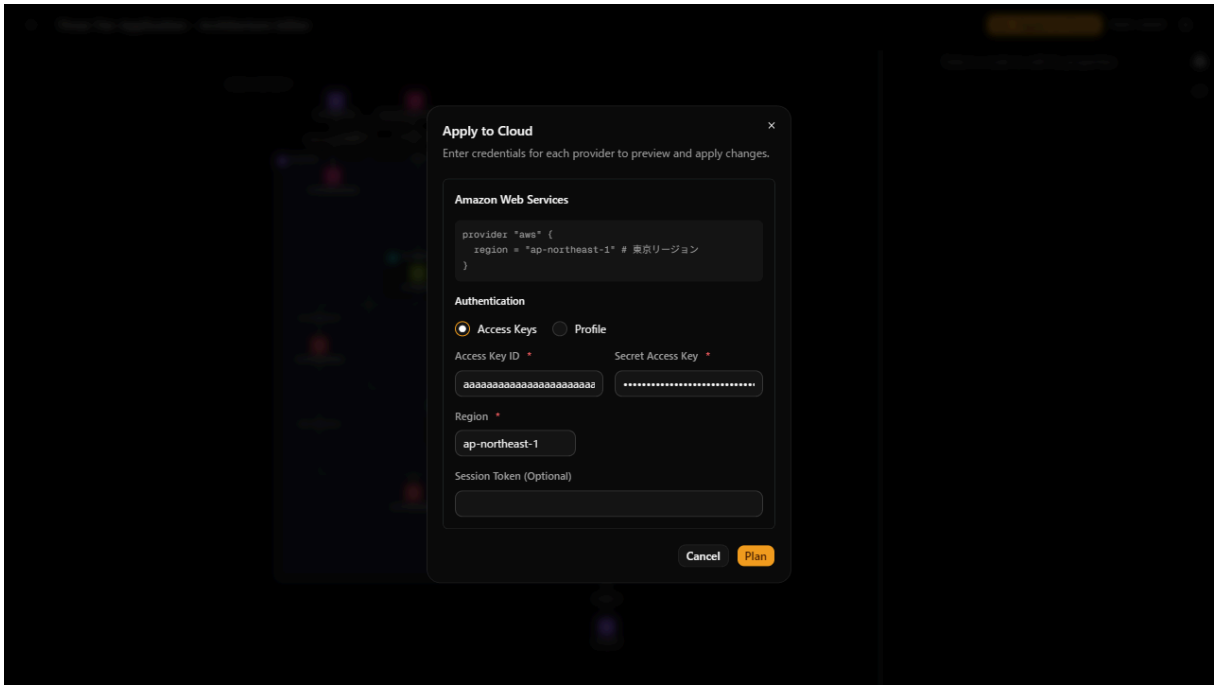

図 22: Apply to Cloud の認証情報入力と Plan 実行

Apply フローの認証情報入力と Plan 実行画面は 図 22 に示す．Plan 結果の要約と差分一覧は 図 23 に示す．

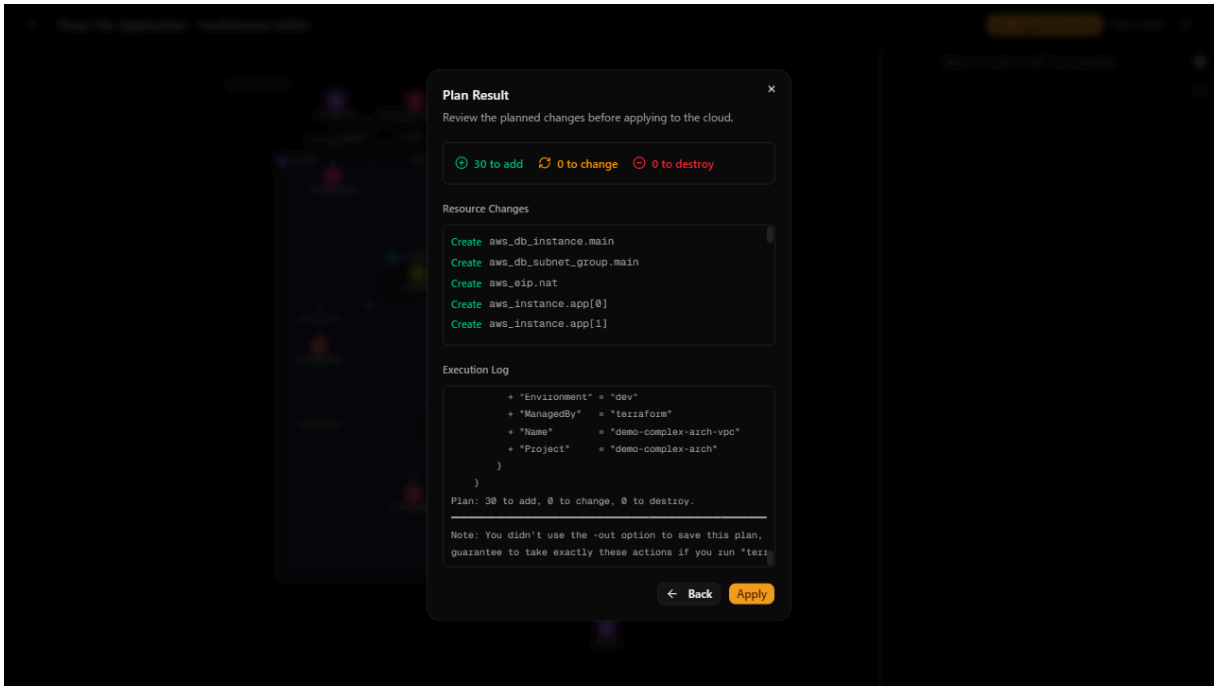

図 23: Plan 結果の要約と Resource Changes 一覧

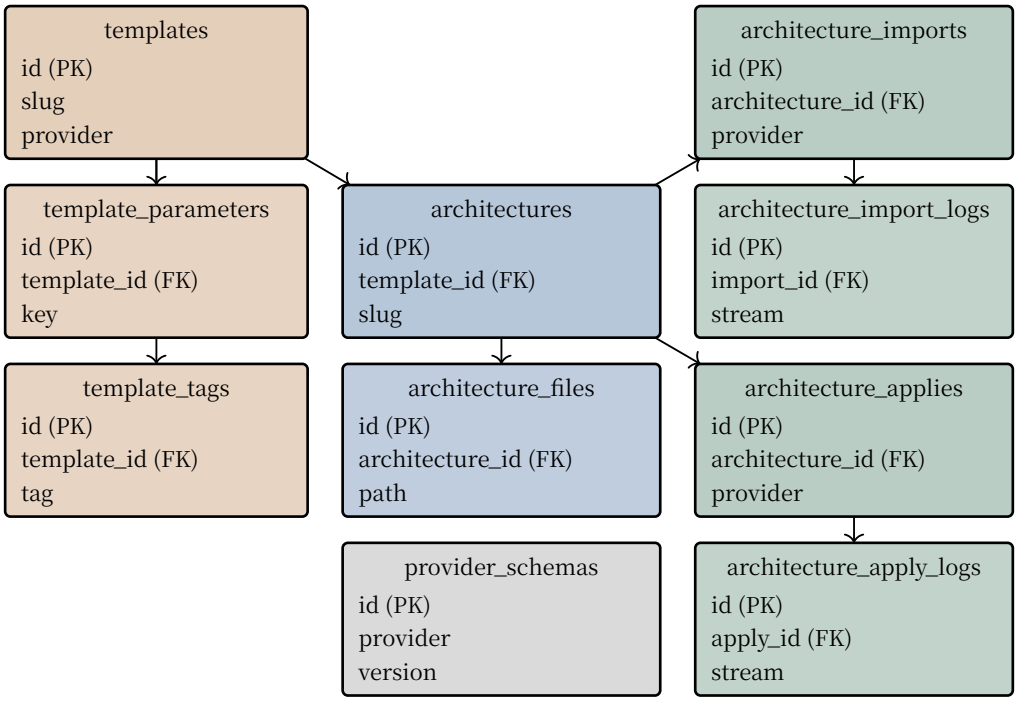

図 24: データベースの ER 図

7 データ管理と運用

7.1 データモデル

データベースは, architectures テーブル, architecture_files テーブル, architecture_importsテーブル, provider_schemas テーブルなどで構成される. architecture_files テーブルが HCL の本体を保持し，graph_json は描画キャッシュとして保存される.

データベースの ER 図は 図 24 に示す.

architectures テーブルはプロジェクト名や説明，インポート元種別を保持し，architecture_files テーブルはファイルパス単位で HCL を保持する. インポート処理は architecture_imports テーブルに状態を記録し，実行ログは architecture_import_logs テーブルに逐次追加されるため，処理の再現性と監査性が確保される. テンプレート機構は templates テーブルと template_parameters テーブルにより入力仕様を保持し，UI はこの定義に基づいて動的フォームを生成できる.

プロバイダスキーマは provider_schemas テーブルにバージョン付きで保存されるため，ネットワークに依存せず再利用できる. これらのテーブルはユニークインデックスで整合性を保証し，architecture_files テーブルでは architecture_id と path の組で重複を防止する.

7.2 キャッシュ戦略と整合性

TerraGUI では HCL テキストを architecture_files テーブルに保存し，graph_json を派生データとして扱う. グラフの再生成時には HCL を再解析し，変更点のみを反映することで整合性を維持する.

7.3 プロバイダスキーマの取得と更新

プロパティエディタが参照するスキーマは，Terraform CLI の providers schema -json を用いて取得し，provider_schemas テーブルに保存する [31]. スキーマはプロバイダ名とバージョンで一意に管

理されるため，既に取得済みのバージョンは再取得を避けられる．このキャッシュ設計により，オフライン環境でも編集体験を維持できる．

7.4 インポートログの保全とストリーミング

インポート処理の進捗は architecture_import_logs テーブルに逐次保存され，UI 側ではストリーミングで取得する．標準出力と標準エラーを区別して保存することで，エラーメッセージの追跡や失敗原因の分析が容易になる．ログとステータスが DB に残るため，途中で画面を閉じても復帰可能である．

7.5 自動保存と競合

グラフの操作や属性編集は一定時間のデバウンス後に保存される．保存処理はトランザクションで包まれ，graph_json と architecture_files テーブルの更新が同時に行われる．競合が発生しうるため，将来的には差分マージや履歴管理の拡張が必要である．

7.6 依存 CLI の検出とセキュリティ

インポート機能では，AWS CLI，gcloud，Azure CLI と terracognita の利用可否を事前に検出する．認証情報はインポート実行時にのみ利用し，DB には保存しない設計とすることで，情報漏洩リスクを最小化する．

CLI 検出は PATH 上の実行ファイル探索とバージョンコマンドの実行により行い，存在しても実行できない場合は候補から除外する．クラウド認証情報は入力直後に CLI で検証し，正当性が確認された場合のみインポートを開始する．この手順により，不正な認証情報や環境不備による失敗を早期に発見できる．

7.7 開発・配布と再現性

Docker によるコンテナ化 [89] と Dev Containers [90] により，OS 依存を排除した開発環境を提供する．VS Code 拡張や Node.js 環境を含めることで，開発者が同一の設定で利用できる．さらに，TypeScript の厳格な型チェック [156] と Biome [92]，Lefthook [93]，commitlint [94] を組み合わせ，品質と保守性を確保する．

開発・配布のワークフローは 図 25 に示す．

TerraGUI はローカル環境で完結するため，追加の SaaS 契約やクラウド依存を必要としない．SQLite を利用することでデータベースの配布が容易になり，terragui.db を含めた状態をそのまま配布・バッ

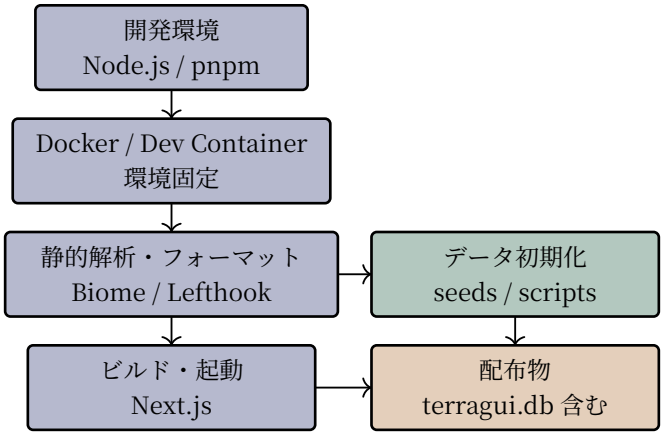

図 25: 開発・配布のワークフロー

表 4: 既存ツールとの機能比較

機能	TerraGUI (提案)	Brainboard	Inframap	Blast Radius
可視化方式	CST ベース （詳細）	独自モデル	State/HCL	State
編集機能	○（双方向・ロスレス）	○ （独自形式）	×	×
コメント保持	○	× （再生成時に消失）	N/A	N/A
既存コード利用	○ （直接編集）	△ （インポート必須）	○	○
UI 階層化	○ （自動・意味的）	○ （手動）	×	×
導入コスト	低 (OSS/Local)	高 (SaaS)	低	低

クアップできる．Dev Container により Ubuntu 環境が統一されるため，研究室やチーム内での再現性が高い．

8 評価

8.1 機能比較

提案システムと，既存の代表的な Terraform 可視化・編集ツールとの機能比較を表 表 4 に示す．

8.2 データセットと評価軸

評価対象としては，公開リポジトリの Terraform コードに加え，TerraDS のような研究用データセットを利用する予定である [118]．コード規模やリソース種類，モジュール構成の多様性を確保し，ロスレス性・グラフ生成精度・編集操作の安定性を測定する．また，TerraMetrics で提案されているメトリクスや，品質評価フレームワークの指標を参照し，客観的な評価軸を整理する [119,121]．

8.3 ロスレス性の検証

実装した HCL Engine の堅牢性を検証するため，GitHub 上の公開 Terraform リポジトリから無作為に抽出した.tf ファイルを用いてテストする計画である．各ファイルに対しパースから文字列化へのラウンドトリップを実施し，元のテキストと一致するかを確認する．また，プロパティの一部を変更して再生成した場合でも，変更箇所以外が維持されることを検証する．

評価では，差分行数と差分比率を指標として記録し，変更操作によって不要な差分が生じていないことを定量的に示す．さらに，HCL を JSON に変換する一般的なツールと比較し，コメントや空行が保持されないケースとの差を明確化する [139]．

8.4 パフォーマンス評価

大規模なグラフを表示した際の描画パフォーマンスを計測する．elkjs によるレイアウト計算は Web Worker 等で非同期化していない現状では，数百ノード規模で数百ミリ秒程度の計算時間を要するが，UI のフリーズは許容範囲内である．React Flow のレンダリングは仮想化が効いており，スクロールやズームは滑らかに動作することを確認した．

追加で，HCL 解析時間，グラフ生成時間，レイアウト時間を分離して測定する計画である．特にレイアウトはノード数とエッジ数に敏感であるため，入力規模に応じたスケーリング特性を評価し，必要に応じて非同期化や差分レイアウトの導入を検討する．

8.5 テスト戦略と指標

今後，Vitest [157] による単体テストとカバレッジ計測を導入し，HCL Engine と Graph Engine の主要関数を網羅的に検証する予定である．加えて，実際の操作を模擬したシナリオテストや，インポート処理の安定性検証を通じて機能品質を評価する．静的解析については TypeScript と Biome を用いて厳格な型安全性と規約準拠を担保する．

評価指標としては，パース成功率，ロスレス率（完全一致率），グラフ生成成功率，及びテストカバレッジを設定する．特に HCL Engine は入力の多様性が高いため，テストケースを TerraDS などの実データに近い分布で生成し，実運用に近い条件での安定性を確認する [118]．

9 考察

9.1 意味的階層化の効果と限界

意味的階層化はユーザのメンタルモデルに近い可視化を提供する一方，リソース属性の推論に依存するため，複雑な式やモジュール構成では意図通りの階層化を実現できない場合もある．特に複数の VPC やリージョンを跨る構成では推論の精度が低下しうるため，今後は明示的なグルーピング指定や手動調整機構が必要となる．

また，Terraform モジュールは内部で複数リソースを生成するため，モジュール境界を超えて依存が張られる場合に階層化が曖昧になる．for_each や count による動的生成は，参照先の実体が評価時に決定するため，静的解析での完全な復元は難しい．これらの制約は，HCL の表現力が高いことによる必然的なトレードオフといえる．

9.2 スケーラビリティと運用負荷

大規模構成に対するレイアウト計算は ELK に依存するため，ノード数増大に伴い計算コストが増加する．非同期レイアウトや差分レイアウトを導入することで改善可能だが，実装コストとのトレードオフが存在する．

さらに，プロバイダスキーマのサイズやアイコンマッピングの規模も増大するため，キャッシュの整理と更新頻度の最適化が重要となる．特にプロバイダの新サービス追加に追従する際には，アイコンセット更新の自動化が必要となる．

9.3 妥当性の脅威

本研究はユーザ評価を実施していないため，操作性や学習効果に関する外的妥当性は限定的である．今後はユーザ実験を通じて，認知負荷や作業効率の改善を定量的に検証する必要がある．

また，評価対象の HCL コードは公開リポジトリに偏る可能性があり，企業内の大規模構成や機密構成を十分に代表できない．複数プロバイダや複雑なモジュール構成に対する一般化可能性についても，今後の検証が求められる．

10 結論

本研究では，Infrastructure as Code の課題である視認性の低さと学習コストの高さを解決するために，Web ベースの視覚的編集環境「TerraGUI」を開発した．提案システムは，以下の 3 つの特徴を持つ．

第一に，ロスレスな具象構文木変換である．typescript-parsec を用いたパーサにより，HCL コードのコメントやフォーマットを維持したまま，GUI による直感的な編集を実現した．第二に，意味的階層化である．elkjs とリソース属性解析を組み合わせ，VPC や Subnet の包含関係を反映した見やすいグラフを自動生成した．第三に，既存ワークフローとの統合である．HCL テキストを.tf 互換のまま保持し，既存の Git ワークフローと矛盾しない形で編集できるようにした．

これらの機能により，開発者の認知負荷を低減し，Infrastructure as Code の保守性と生産性を向上させることが可能となった．

また，テンプレート機構やインポート機構を通じて，既存環境と新規構成の双方に対応できることを示した．HCL のロスレス編集と視覚的編集の融合は，Infrastructure as Code の導入障壁を下げ，設計と実装の往復を容易にする基盤になると考える．

今後の課題として，複数人による同時編集機能の実装や，AI エージェントや LLM を用いた自然言語からの構成提案機能の統合が挙げられる．これらの方向性は，LLM を活用した Infrastructure as Code 支援研究の動向とも整合しており，将来的な拡張の基盤となる [122,123,125]．

謝辞

本研究を進めるにあたり，熱心なご指導とご鞭撻を賜りました指導教員の井口教授に深く感謝いたします．また，日々の議論を通じて有益な助言をいただいた研究室の皆様，ならびに本ソフトウェアの評価に協力していただいた方々に心より感謝申し上げます．

参考文献

[1] HashiCorp, Terraform, https://www.terraform.io/.

[2] HashiCorp, HCL (HashiCorp Configuration Language), https://github.com/hashicorp/hcl.

[3] Amazon Web Services, AWS CloudFormation, https://aws.amazon.com/cloudformation/.

[4] Microsoft, Azure Resource Manager templates, https://learn.microsoft.com/azure/azure-resource-manager/templates/overview.

[5] Google Cloud, Google Cloud Deployment Manager, https://cloud.google.com/deployment-manager.

[6] Pulumi, Pulumi, https://www.pulumi.com/.

[7] HashiCorp, CDK for Terraform, https://developer.hashicorp.com/terraform/cdktf.

[8] Red Hat, Ansible, https://www.ansible.com/.

[9] Puppet, Puppet, https://www.puppet.com/.

[10] Progress Software, Chef, https://www.chef.io/.

[11] Broadcom, SaltStack, https://saltproject.io/.

[12] Pierre-Jean Quéval, Nicole Elisabeth Hörner, Evangelos Ntentos, 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.

[13] Brainboard, https://www.brainboard.co/.

[14] Hieven, Terraform Visual, https://github.com/hieven/terraform-visual.

[15] Cycloid, Inframap, https://github.com/cycloidio/inframap.

[16] 28mm, Blast Radius, https://github.com/28mm/blast-radius.

[17] HashiCorp, terraform graph command reference, https://developer.hashicorp.com/terraform/cli/commands/graph.

[18] Akond Rahman, Effat Farhana, Chris Parnin, Laurie A. Williams, Gang of eight: a defect taxonomy for infrastructure as code scripts., 2020, https://dblp.org/rec/conf/icse/RahmanFPW20.

[19] Akond Rahman, Effat Farhana, Laurie A. Williams, The 'as code' activities: development anti-patterns for infrastructure as code., 2020, https://dblp.org/rec/journals/ese/RahmanFW20.

[20] Akond Rahman, Characteristics of defective infrastructure as code scripts in DevOps., 2018, https://dblp.org/rec/conf/icse/Rahman18.

[21] Akond Rahman, Jonathan Stallings, Laurie A. Williams, Defect prediction metrics for infrastructure as code scripts in DevOps., 2018, https://dblp.org/rec/conf/icse/RahmanSW18.

[22] Akond Rahman, Anti-Patterns in Infrastructure as Code., 2018, https://dblp.org/rec/conf/icst/Rahman18.

[23] Julian Schwarz, Andreas Steffens, Horst Lichter, Code Smells in Infrastructure as Code., 2018, https://dblp.org/rec/conf/quatic/SchwarzSL18.

[24] Akond Rahman, Chris Parnin, Laurie A. Williams, The seven sins: security smells in infrastructure as code scripts., 2019, https://dblp.org/rec/conf/icse/RahmanPW19.

[25] Akond Rahman, Laurie A. Williams, Different Kind of Smells: Security Smells in Infrastructure as Code Scripts., 2021, https://dblp.org/rec/journals/ieeesp/RahmanW21.

[26] Akond Rahman, Farhat Lamia Barsha, Patrick Morrison, Shhh!: 12 Practices for Secret Management in Infrastructure as Code., 2021, https://dblp.org/rec/conf/secdev/RahmanBM21.

[27] Aicha War, Alioune Diallo, Andrew Habib, Jacques Klein, Tegawende F. Bissyande, Vulnerabilities in infrastructure as code: what, how many, and who?, 2025, https://dblp.org/rec/journals/ese/WarDHKB25.

[28] Narjes Bessghaier, Ali Ouni, Mohammed Sayagh, Moataz Chouchen, Mohamed Wiem Mkaouer, Towards understanding code review practices for infrastructure-as-code: An empirical study on OpenStack projects., 2025, https://dblp.org/rec/journals/ese/BessghaierOSCM25.

[29] HashiCorp, Terraform Registry, https://registry.terraform.io/.

[30] Mahi Begoug, Ali Ouni, Moataz Chouchen, How Do Infrastructure-as-Code Practitioners Update Their Dependencies? An Empirical Study on Terraform Module Updates., 2025, https://dblp.org/rec/conf/msr/Begoug0C25.

[31] HashiCorp, terraform providers schema command, https://developer.hashicorp.com/terraform/cli/commands/providers/schema.

[32] HashiCorp, HCL Native Syntax Specification, https://github.com/hashicorp/hcl/blob/main/hclsyntax/spec.md.

[33] HashiCorp, hclwrite package, https://pkg.go.dev/github.com/hashicorp/hcl/v2/hclwrite.

[34] Kieler, elkjs, https://github.com/kieler/elkjs.

[35] HashiCorp, Terraform Language Server, https://github.com/hashicorp/terraform-ls.

[36] HashiCorp, Terraform Expressions, https://developer.hashicorp.com/terraform/language/expressions.

[37] HashiCorp, Terraform Dynamic Blocks, https://developer.hashicorp.com/terraform/language/expressions/dynamic-blocks.

[38] HashiCorp, Terraform Meta-Arguments, https://developer.hashicorp.com/terraform/language/meta-arguments.

[39] HashiCorp, Terraform Modules, https://developer.hashicorp.com/terraform/language/modules.

[40] HashiCorp, Terraform State, https://developer.hashicorp.com/terraform/language/state.

[41] HashiCorp, Terraform Workspaces, https://developer.hashicorp.com/terraform/language/state/workspaces.

[42] HashiCorp, terraform plan command reference, https://developer.hashicorp.com/terraform/cli/commands/plan.

[43] HashiCorp, terraform apply command reference, https://developer.hashicorp.com/terraform/cli/commands/apply.

[44] HashiCorp, terraform validate command reference, https://developer.hashicorp.com/terraform/cli/commands/validate.

[45] HashiCorp, terraform fmt command reference, https://developer.hashicorp.com/terraform/cli/commands/fmt.

[46] HashiCorp, Terraform JSON Configuration Syntax, https://developer.hashicorp.com/terraform/language/syntax/json.

[47] HashiCorp, Terraform Configuration Language, https://developer.hashicorp.com/terraform/language.

[48] Gruntwork, Terragrunt, https://terragrunt.gruntwork.io/.

[49] Terraform Linters, TFLint, https://github.com/terraform-linters/tflint.

[50] Aqua Security, tfsec, https://github.com/aquasecurity/tfsec.

[51] Bridgecrew, Checkov, https://www.checkov.io/.

[52] Tenable, Terrascan, https://github.com/tenable/terrascan.

[53] Infracost, Infracost, https://www.infracost.io/.

[54] Open Policy Agent, Open Policy Agent, https://www.openpolicyagent.org/.

[55] Open Policy Agent, Conftest, https://www.conftest.dev/.

[56] HashiCorp, HCP Terraform (Terraform Cloud) Documentation, https://developer.hashicorp.com/terraform/cloud-docs.

[57] OpenTofu, OpenTofu, https://opentofu.org/.

[58] Atlantis, Atlantis, https://www.runatlantis.io/.

[59] Spacelift, Spacelift, https://spacelift.io/.

[60] Scalr, Scalr, https://www.scalr.com/.

[61] Cloudify, Cloudify, https://cloudify.co/.

[62] tree-sitter-grammars, tree-sitter-hcl, https://github.com/tree-sitter-grammars/tree-sitter-hcl.

[63] Microsoft, ts-parsec, https://github.com/microsoft/ts-parsec.

[64] OASIS, TOSCA, https://www.oasis-open.org/committees/tosca/.

[65] Kubernetes, Kubernetes, https://kubernetes.io/.

[66] Crossplane, Crossplane, https://www.crossplane.io/.

[67] Meta, LibCST, https://github.com/Instagram/LibCST.

[68] Ben Newman, Recast, https://github.com/benjamn/recast.

[69] ANTLR, ANTLR Parser Generator, https://www.antlr.org/.

[70] Bryan Ford, Parsing Expression Grammars: A Recognition-Based Syntactic Foundation, POPL, (2004), https://doi.org/10.1145/964001.964011.

[71] Bryan Ford, Packrat Parsing: Simple, Powerful, Lazy, Linear Time, ICFP, (2002), https://doi.org/10.1145/581478.581483.

[72] Graphviz, Graphviz, https://graphviz.org/.

[73] Graphviz, The DOT Language, https://graphviz.org/doc/info/lang.html.

[74] xyflow, React Flow, https://reactflow.dev/.

[75] Eclipse Foundation, Eclipse Layout Kernel, https://www.eclipse.org/elk/.

[76] Kozo Sugiyama, Shojiro Tagawa, Mitsuhiko Toda, Methods for visual understanding of hierarchical system structures, 1981, https://doi.org/10.1109/TSMC.1981.4308636.

[77] Vercel, Next.js, https://nextjs.org/.

[78] Microsoft, Monaco Editor, https://microsoft.github.io/monaco-editor/.

[79] Meta, React, https://react.dev/.

[80] Tailwind Labs, Tailwind CSS, https://tailwindcss.com/.

[81] WorkOS, Radix UI, https://www.radix-ui.com/.

[82] Allotment, Allotment Split Pane for React, https://github.com/johnwalley/allotment.

[83] react-hook-form, React Hook Form, https://react-hook-form.com/.

[84] Zod, Zod, https://zod.dev/.

[85] Lucide, Lucide Icons, https://lucide.dev/.

[86] SQLite, SQLite, https://www.sqlite.org/.

[87] Drizzle Team, Drizzle ORM, https://orm.drizzle.team/.

[88] OpenJS Foundation, Node.js, https://nodejs.org/.

[89] Docker, Docker, https://www.docker.com/.

[90] Dev Containers, Development Containers Specification, https://containers.dev/.

[91] Microsoft, Visual Studio Code, https://code.visualstudio.com/.

[92] Biome, Biome, https://biomejs.dev/.

[93] Evil Martians, Lefthook, https://github.com/evilmartians/lefthook.

[94] commitlint, commitlint, https://commitlint.js.org/.

[95] Akond Rahman, Laurie A. Williams, Source code properties of defective infrastructure as code scripts., 2019, https://dblp.org/rec/journals/infsof/RahmanW19.

[96] Georgios-Petros Drosos, Thodoris Sotiropoulos, Georgios Alexopoulos, Dimitris Mitropoulos, Zhendong Su, When Your Infrastructure Is a Buggy Program: Understanding Faults in Infrastructure as Code Ecosystems., 2024, https://dblp.org/rec/journals/pacmpl/DrososSAM024.

[97] Md. Mahadi Hassan, John Salvador, Shubhra Kanti Karmaker Santu, Akond Rahman, State Reconciliation Defects in Infrastructure as Code., 2024, https://dblp.org/rec/journals/pacmse/HassanSSR24.

[98] Nemania Borovits, Indika Kumara, Dario Di Nucci, Parvathy Krishnan, Stefano Dalla Palma, Fabio Palomba, Damian A. Tamburri, Willem-Jan van den Heuvel, FindICI: Using machine learning to detect linguistic inconsistencies between code and natural language descriptions in infrastructure-as-code., 2022, https://dblp.org/rec/journals/ese/BorovitsKNKPPTH22.

[99] Nuno Saavedra, Joao Goncalves, Miguel Henriques, Joao F. Ferreira, Alexandra Mendes, Polyglot Code Smell Detection for Infrastructure as Code with GLITCH., 2023, https://dblp.org/rec/conf/kbse/SaavedraGHFM23.

[100] Nuno Saavedra, Joao F. Ferreira, Alexandra Mendes, GLITCH: Polyglot Code Smell Detection in Infrastructure as Code., 2024, https://dblp.org/rec/journals/ercim/Saavedra0M24.

[101] Julio Sandobalin, Emilio Insfran, Silvia Abrahao, On the Effectiveness of Tools to Support Infrastructure as Code: Model-Driven Versus Code-Centric., 2020, https://dblp.org/rec/journals/access/SandobalinIA20.

[102] Isac Sacchi e Souza, Daniel Pinheiro Franco, Joao Pedro Sao Gregorio Silva, Infrastructure as Code as a Foundational Technique for Increasing the DevOps Maturity Level: Two Case Studies., 2023, https://dblp.org/rec/journals/software/SouzaFS23.

[103] Akond Rahman, Rezvan Mahdavi-Hezaveh, Laurie A. Williams, Where Are The Gaps? A Systematic Mapping Study of Infrastructure as Code Research., 2018, https://dblp.org/rec/journals/corr/abs-1807-04872.

[104] Akond Rahman, Sarah Elder, Faysal Hossain Shezan, Vanessa Frost, Jonathan Stallings, Laurie A. Williams, Categorizing Defects in Infrastructure as Code., 2018, https://dblp.org/rec/journals/corr/abs-1809-07937.

[105] Anh-Duy Tran, Laurens Sion, Koen Yskout, Wouter Joosen, TerrARA: Automated Security Threat Modeling for Infrastructure as Code., 2025, https://dblp.org/rec/conf/codaspy/TranSYJ25.

[106] Akond Rahman, Md. Rayhanur Rahman, Chris Parnin, Laurie A. Williams, Security Smells in Infrastructure as Code Scripts., 2019, https://dblp.org/rec/journals/corr/abs-1907-07159.

[107] Farzana Ahamed Bhuiyan, Akond Rahman, Characterizing co-located insecure coding patterns in infrastructure as code scripts., 2020, https://dblp.org/rec/conf/kbse/BhuiyanR20.

[108] Aicha War, Serge Lionel Nikiema, Jordan Samhi, Jacques Klein, Tegawende F. Bissyande, Security smells in infrastructure as code: a taxonomy update beyond the seven sins., 2025, https://dblp.org/rec/journals/corr/abs-2509-18761.

[109] Alexandre Verdet, Mohammad Hamdaqa, Leuson M. P. da Silva, Foutse Khomh, Assessing the adoption of security policies by developers in terraform across different cloud providers., 2025, https://dblp.org/rec/journals/ese/VerdetHSK25.

[110] Maryna Lukaczyk, Andrzej Mycek, Automation of Security Policies in DevSecOps Environments: Implementation of Zero Trust Principles Using Ansible and Terraform in Linux Systems., 2025, https://dblp.org/rec/conf/ecms/LukaczykM25.

[111] Adilson G. Filho, Eduardo K. Viegas, Altair O. Santin, Jhonatan Geremias, A Dynamic Network Intrusion Detection Model for Infrastructure as Code Deployed Environments., 2025, https://dblp.org/rec/journals/jnsm/FilhoVSG25.

[112] Akond Rahman, Rezvan Mahdavi-Hezaveh, Laurie A. Williams, A systematic mapping study of infrastructure as code research., 2019, https://dblp.org/rec/journals/infsof/RahmanMW19.

[113] Claus Pahl, Niyazi Gokberk Gunduz, Ovgum Can Sezen, Ali Ghamgosar, Nabil El Ioini, Infrastructure as Code: Technology Review and Research Challenges., 2025, https://dblp.org/rec/conf/closer/PahlGSGI25.

[114] Evangelos Ntentos, Nicole Elisabeth Lueger, Georg Simhandl, Uwe Zdun, Simon Schneider, Riccardo Scandariato, Nicolas E. Diaz Ferreyra, On the Understandability of Design-

Level Security Practices in Infrastructure-as-Code Scripts and Deployment Architectures., 2025, https://dblp.org/rec/journals/tosem/NtentosLSZSSF25.

[115] Mohamed A. Oumaziz, Cloning beyond source code: a study of the practices in API documentation and infrastructure as code., 2020, https://dblp.org/rec/phd/hal/Oumaziz20.

[116] Ruben Opdebeeck, Bram Adams, Coen De Roover, Analysing Software Supply Chains of Infrastructure as Code: Extraction of Ansible Plugin Dependencies., 2025, https://dblp.org/rec/conf/saner/OpdebeeckAR25.

[117] Akond Rahman, Md. Shazibul Islam Shamim, Hossain Shahriar, Fan Wu, Can We use Authentic Learning to Educate Students about Secure Infrastructure as Code Development?, 2022, https://dblp.org/rec/conf/iticse/RahmanSS022.

[118] Christoph Buhler, David Spielmann, Roland Meier, Guido Salvaneschi, TerraDS: A Dataset for Terraform HCL Programs., 2025, https://dblp.org/rec/conf/msr/BuhlerSMS25.

[119] Mahi Begoug, Moataz Chouchen, Ali Ouni, TerraMetrics: An Open Source Tool for Infrastructure-as-Code (IaC) Quality Metrics in Terraform., 2024, https://dblp.org/rec/conf/iwpc/BegougC024.

[120] Hanyang Hu, Yani Bu, Kristen Wong, Gaurav Sood, Karen Smiley, Akond Rahman, Characterizing Static Analysis Alerts for Terraform Manifests: An Experience Report., 2023, https://dblp.org/rec/conf/secdev/HuBWSSR23.

[121] Pandu Ranga Reddy Konala, Vimal Kumar, David Bainbridge, Junaid Haseeb, A Framework for Measuring the Quality of Infrastructure-as-Code Scripts., 2025, https://dblp.org/rec/journals/corr/abs-2502-03127.

[122] Quoc-Huy Vo, Ha Dao, Kensuke Fukuda, Harnessing the Power of LLMs for Code Smell Detection in Terraform Infrastructure as Code., 2025, https://dblp.org/rec/conf/compsac/VoDF25.

[123] Dheer Toprani, Vijay Krishna Madisetti, LLM Agentic Workflow for Automated Vulnerability Detection and Remediation in Infrastructure-as-Code., 2025, https://dblp.org/rec/journals/access/TopraniM25.

[124] Patrick Tser Jern Kon, Jiachen Liu, Yiming Qiu, Weijun Fan, Ting He, Lei Lin, Haoran Zhang, Owen Park, George Elengikal, Yuxin Kang, Ang Chen, Mosharaf Chowdhury, Myungjin Lee, Xinyu Wang, IaC-Eval: A Code Generation Benchmark for Cloud Infrastructure-as-Code Programs., 2024, https://dblp.org/rec/conf/nips/KonLQFHLZPEK0CL24.

[125] Rana Nameer Hussain Khan, Dawood Wasif, Jin-Hee Cho, Ali Raza Butt, Multi-Agent Code-Orchestrated Generation for Reliable Infrastructure-as-Code., 2025, https://dblp.org/rec/journals/corr/abs-2510-03902.

[126] Yiming Xiang, Zhenning Yang, Jingjia Peng, Hermann Bauer, Patrick Tser Jern Kon, Yiming Qiu, Ang Chen, Automated Bug Discovery in Cloud Infrastructure-as-Code Updates with LLM Agents., 2025, https://dblp.org/rec/conf/icse/XiangYPBKQC25.

[127] Jingjia Peng, Yiming Qiu, Patrick Tser Jern Kon, Pinhan Zhao, Yibo Huang, Zheng Guo, Xinyu Wang, Ang Chen, Automated Lifting for Cloud Infrastructure-as-Code Programs., 2025, https://dblp.org/rec/conf/icse/PengQKZHGWC25.

[128] Nuno Saavedra, Joao F. Ferreira, Alexandra Mendes, InfraFix: Technology-Agnostic Repair of Infrastructure as Code., 2025, https://dblp.org/rec/conf/issta/Saavedra0M25.

[129] Emilio Coppa, Daniel Sokolowski, Guido Salvaneschi, Hybrid Fuzzing of Infrastructure as Code Programs (Short Paper)., 2025, https://dblp.org/rec/conf/issta/CoppaSS25.

[130] Haoran Wei, Nazim H. Madhavji, John Steinbacher, A Framework for Reusable Infrastructure as Code Templates in Cloud-Native Environments., 2025, https://dblp.org/rec/conf/icsr/WeiMS25.

[131] Giovanni Battista Barone, Gianluca Sabella, Innovating Cloud-Based Academic Services: Advancing the University of Naples Federico II's Infrastructure with Terraform and Oracle GoldenGate., 2025, https://dblp.org/rec/conf/aina/BaroneS25.

[132] Esteban Elias Romero, Carlos David Camacho, Carlos Enrique Montenegro-Marin, Oscar Esneider Acosta Agudelo, Ruben Gonzalez Crespo, Elvis Eduardo Gaona-Garcia, Marcelo Herrera Martinez, Integration of DevOps Practices on a Noise Monitor System with CircleCI and Terraform., 2022, https://dblp.org/rec/journals/tmis/RomeroCMACGM22.

[133] Nitin Naik, Cloud-Agnostic and Lightweight Big Data Processing Platform in Multiple Clouds Using Docker Swarm and Terraform., 2021, https://dblp.org/rec/conf/ukci/Naik21a.

[134] Denis B. Citadin, Fabio Diniz Rossi, Marcelo Caggiani Luizelli, Philippe O. A. Navaux, Arthur Francisco Lorenzon, Energy-Aware Node Selection for Cloud-Based Parallel Workloads with Machine Learning and Infrastructure as Code., 2025, https://dblp.org/rec/conf/closer/CitadinRLNL25.

[135] Carlos Eduardo Duarte, Automated Microservice Pattern Instance Detection Using Infrastructure-as-Code Artifacts and Large Language Models., 2025, https://dblp.org/rec/conf/icsa/Duarte25.

[136] Joao Frois, Lucas Padrao, Johnatan Oliveira, Laerte Xavier, Cleiton Silva Tavares, Terraform and AWS CDK: A Comparative Analysis of Infrastructure Management Tools., 2024, https://dblp.org/rec/conf/sbes/FroisPOXT24.

[137] Leonardo Reboucas de Carvalho, Aleteia Patricia Favacho de Araujo, Performance Comparison of Terraform and Cloudify as Multicloud Orchestrators., 2020, https://dblp.org/rec/conf/ccgrid/CarvalhoA20.

[138] Minamijoyo, hcledit, https://github.com/minamijoyo/hcledit.

[139] tmccombs, hcl2json, https://github.com/tmccombs/hcl2json.

[140] HashiCorp, terraform-config-inspect, https://github.com/hashicorp/terraform-config-inspect.

[141] Amazon Web Services, AWS CloudFormation Designer, https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/working-with-templates-cfn-designer.html.

[142] Cloudcraft, Cloudcraft, https://www.cloudcraft.co/.

[143] diagrams.net, diagrams.net, https://www.diagrams.net/.

[144] Lucidchart, Lucidchart, https://www.lucidchart.com/.

[145] Mermaid, Mermaid, https://mermaid.js.org/.

[146] PlantUML, PlantUML, https://plantuml.com/.

[147] Structurizr, Structurizr, https://structurizr.com/.

[148] C4 Model, The C4 Model for visualising software architecture, https://c4model.com/.

[149] Cycloid, TerraCognita, https://github.com/cycloidio/terracognita.

[150] Amazon Web Services, AWS Architecture Icons, https://aws.amazon.com/architecture/icons/.

[151] Google Cloud, Google Cloud Icons, https://cloud.google.com/icons.

[152] Microsoft, Azure Architecture Icons, https://learn.microsoft.com/azure/architecture/icons/.

[153] Amazon Web Services, AWS CLI, https://aws.amazon.com/cli/.

[154] Google Cloud, Google Cloud CLI, https://cloud.google.com/sdk.

[155] Microsoft, Azure CLI, https://learn.microsoft.com/cli/azure/.

[156] Microsoft, TypeScript, https://www.typescriptlang.org/.

[157] Vitest, Vitest, https://vitest.dev/.