Infrastructure Documentation

Overview

This infrastructure is built using CDK for Terraform (CDKTF) with TypeScript, providing a robust and scalable infrastructure management system. The infrastructure is organized in a hierarchical structure with clear separation of concerns and well-defined responsibilities at each level.

Infrastructure Hierarchy

```
graph TD
    A[Root Level] --> B[Global Level]
    B --> C1[Jurisdiction A]
    B --> C2[Jurisdiction B]
    C1 --> D1[Development Environment]
    C1 --> D2[Production Environment]
    C2 --> D3[Development Environment]
    C2 --> D4[Production Environment]
```

Core Components

1. Root Level (`root-admin-stack.ts`)

The foundation of the entire infrastructure, managing organization-wide settings and security.

Key Responsibilities:

- Organization-wide security and access control
- Provider account management
- Sensitive credential management
- Root-level spaces and contexts
- Base IAM roles and permissions

Dependencies:

- AWS IAM roles and policies
- Google Cloud organization settings
- Cloudflare API tokens
- GitHub organization tokens
- MongoDB Atlas keys
- SonarQube tokens
- Terraform S3 credentials
- Slack tokens

2. Global Level (`global-admin-stack.ts`)

Manages cross-jurisdiction configurations and shared resources.

Key Responsibilities:

- Shared infrastructure components
- Common policies across jurisdictions
- Global service integrations
- Cross-jurisdiction dependencies

Dependencies:

- Spacelift provider
- GitHub integration
- SonarQube integration
- HCL configuration parser

3. Jurisdiction Level (`jurisdiction-admin-stack.ts`)

Handles department/region-specific infrastructure and configurations.

Key Responsibilities:

- Jurisdiction-specific infrastructure
- Environment-specific configurations
- Local policies and access controls
- Jurisdiction-specific integrations

Dependencies:

- Multiple cloud providers:
  - AWS (IAM roles and policies)
  - Google Cloud (organization and project settings)
  - MongoDB Atlas (organization and API keys)

Root-Level Configuration Files

The root directory contains several important configuration files that define the overall infrastructure setup, development environment, and deployment configurations.

Configuration Files

```
graph TD
    A[Root Configuration] --> B[Infrastructure Config]
    A --> C[Development Tools]
    A --> D[Deployment Config]
    A --> E[Documentation]

    B --> F[root.hcl]
    B --> G[backend.hcl]
    C --> H[package.json]
    C --> I[.prettierrc]
    D --> J[npm_registry_config.hcl]
    D --> K[.pre-commit-config.yaml]
    E --> L[README.md]
    E --> M[OPERATIONS.md]
```

Infrastructure Configuration

1. Root Configuration (`root.hcl`)

- **Purpose**: Defines global infrastructure settings and mappings
- **Key Components**:
  - Environment type mappings
  - Jurisdiction configurations
  - Data center locations
  - Account settings
  - Service configurations
- **Features**:
  - Environment type to stage mapping
  - Environment type to short code mapping
  - Jurisdiction to data center mapping
  - Path and directory management
  - Service and subsystem detection
  - Cloud provider account configurations

2. Backend Configuration (`backend.hcl`)

- **Purpose**: Configures Terraform state storage
- **Features**:
  - S3 backend configuration
  - Cloudflare R2 integration
  - State encryption
  - Credential management
- **Security**:
  - Encrypted state storage
  - Secure credential handling
  - Access control
  - Endpoint configuration

Development Tools

1. Package Management (`package.json`)

- **Purpose**: Manages Node.js dependencies
- **Features**:
  - Dependency management
  - Script definitions
  - Project metadata
  - Development tools

2. Code Formatting (`.prettierrc`, `.prettierignore`)

- **Purpose**: Maintains code style consistency
- **Features**:
  - Code formatting rules
  - File exclusions
  - Style enforcement
  - Integration with editors

3. Pre-commit Hooks (`.pre-commit-config.yaml`)

- **Purpose**: Enforces code quality checks
- **Features**:
  - Pre-commit validations
  - Code quality checks
  - Format verification
  - Security scanning

Deployment Configuration

1. NPM Registry (`npm_registry_config.hcl`)

- **Purpose**: Configures NPM package management
- **Features**:
  - Registry settings
  - Package access
  - Authentication
  - Version management

Documentation

1. README (`README.md`)

- **Purpose**: Project overview and setup instructions
- **Features**:
  - Project description
  - Setup guide
  - Usage instructions
  - Contribution guidelines

2. Operations Guide (`OPERATIONS.md`)

- **Purpose**: Operational procedures and guidelines
- **Features**:
  - Operational procedures
  - Maintenance tasks
  - Troubleshooting
  - Best practices

Global Directory Structure

The `global` directory contains shared infrastructure components and configurations that are used across all jurisdictions. This directory is organized to support a zero-trust security model and centralized repository management.

Structure

```
global/
└── shared/
    ├── repos/           # Repository management
    │   ├── private/     # Private repositories
    │   ├── public/      # Public repositories
    │   └── archived/    # Archived repositories
```

```
    └── zero-trust/      # Zero-trust security configurations
        ├── .terraform.lock.hcl
        └── terragrunt.hcl
```

Components

1. Repository Management (`repos/`)

The repository management structure is organized into three main categories:

```
graph TD
    A[Repository Management] --> B[Private Repos]
    A --> C[Public Repos]
    A --> D[Archived Repos]

    B --> E[Internal Projects]
    B --> F[Sensitive Code]
    C --> G[Open Source]
    C --> H[Documentation]
    D --> I[Legacy Code]
    D --> J[Deprecated Projects]
```

- **Private Repositories**
  - Internal projects and applications
  - Sensitive code and configurations
  - Access-controlled resources
  - Team-specific projects
- **Public Repositories**
  - Open-source projects
  - Public documentation
  - Community resources
  - Shared libraries
- **Archived Repositories**
  - Legacy code preservation
  - Deprecated projects
  - Historical reference
  - Compliance requirements

2. Zero-Trust Security (`zero-trust/`)

Implements a zero-trust security model across the infrastructure:

```
graph TD
    A[Zero-Trust Security] --> B[Access Control]
    A --> C[Network Security]
    A --> D[Data Protection]

    B --> E[Identity Verification]
    B --> F[Least Privilege]
    C --> G[Network Segmentation]
    C --> H[Traffic Control]
```

```
    D --> I[Data Encryption]
    D --> J[Access Logging]
```

Key Features:

- **Access Control**
  - Identity verification
  - Least privilege principle
  - Role-based access control
  - Multi-factor authentication
- **Network Security**
  - Network segmentation
  - Traffic control
  - Secure communication
  - Perimeter security
- **Data Protection**
  - Data encryption
  - Access logging
  - Audit trails
  - Compliance monitoring

Jurisdictions Directory Structure

The `jurisdictions` directory contains environment-specific configurations and resources for different jurisdictions. Each jurisdiction represents a distinct organizational unit or region with its own set of environments and configurations.

Structure

```
jurisdictions/
├── sf/                    # San Francisco jurisdiction
│   ├── apps.hcl           # Application configurations
│   ├── jurisdiction.hcl   # Jurisdiction-level settings
│   ├── development/       # Development environments
│   ├── non-production/    # Non-production environments
│   ├── operations/        # Operations environments
│   ├── production/        # Production environments
│   ├── quality/           # Quality assurance
│   └── sandbox/           # Sandbox environments
├── safecdx/               # SafeCDX jurisdiction
└── opencdx/               # OpenCDX jurisdiction
```

Environment Hierarchy

```
graph TD
    A[Jurisdiction] --> B[Development]
    A --> C[Non-Production]
    A --> D[Operations]
    A --> E[Production]
    A --> F[Quality]
    A --> G[Sandbox]
```

```
    B --> H[Dev-1]
    B --> I[Dev-2]
    C --> J[Staging]
    C --> K[Testing]
    E --> L[Prod-1]
    E --> M[Prod-2]
```

Components

1. Configuration Files

- **jurisdiction.hcl**
  - Jurisdiction-level settings
  - Common configurations
  - Shared resources
  - Environment defaults
- **apps.hcl**
  - Application-specific configurations
  - Service definitions
  - Resource allocations
  - Dependencies
- **env.hcl** (in each environment)
  - Environment-specific settings
  - Resource configurations
  - Provider settings
  - Security policies

2. Environment Types

```
graph TD
    A[Environment Types] --> B[Development]
    A --> C[Non-Production]
    A --> D[Operations]
    A --> E[Production]

    B --> F[Feature Development]
    B --> G[Integration Testing]
    C --> H[Staging]
    C --> I[Performance Testing]
    D --> J[Monitoring]
    D --> K[Maintenance]
    E --> L[Live Services]
    E --> M[Customer Data]
```

- **Development**
  - Feature development
  - Integration testing
  - Developer tools
  - Debugging environments

- **Non-Production**
  - Staging environments
  - Performance testing
  - User acceptance testing
  - Pre-production validation
- **Operations**
  - Monitoring systems
  - Maintenance tools
  - Support services
  - Administrative functions
- **Production**
  - Live services
  - Customer data
  - High availability
  - Disaster recovery
- **Quality**
  - Quality assurance
  - Testing automation
  - Performance monitoring
  - Security testing
- **Sandbox**
  - Experimental features
  - Proof of concept
  - Training environments
  - Development playground

Provider Integration

The infrastructure integrates with multiple cloud providers and services:

```
graph LR
    A[Infrastructure] --> B[AWS]
    A --> C[Google Cloud]
    A --> D[Cloudflare]
    A --> E[MongoDB Atlas]
    A --> F[GitHub]
    A --> G[SonarQube]
    A --> H[Spacelift]
```

Provider Configuration Details

1. AWS Providers

- **Standard AWS** (`aws.hcl`)

    - Basic AWS provider configuration
    - Region and profile management
    - CI/CD environment detection
    - Standard resource access

- **Multi-Account** (`aws_multi_account.hcl`)
    - Multiple AWS account management
    - Role assumption configuration
    - Cross-account resource access
    - Spacelift integration

- **Multi-Region** (`aws_multi_region.hcl`)
    - Cross-region resource management
    - Regional provider aliases
    - Region-specific settings
    - Global resource coordination

- **Cloud Control** (`awscc.hcl` , `awscc_multi_region.hcl`)
    - AWS Cloud Control API integration
    - Resource management
    - Multi-region support
    - Advanced resource control

2. Google Cloud Providers

- **Standard GCP** (`google.hcl`)
    - GCP resource management
    - Project configuration
    - Service account integration
    - Resource organization

- **Beta Features** (`google_beta.hcl`)
    - Beta feature support
    - Experimental resources
    - Preview functionality
    - Advanced configurations

3. Integration Providers

- **Cloudflare** (`cloudflare.hcl`)
    - DNS management
    - CDN configuration
    - Security settings
    - Zone management

- **GitHub** (`github.hcl`)
    - Repository management
    - Organization settings

    - Access control
    - Webhook configuration

- **MongoDB Atlas** (`mongodbatlas.hcl`)
    - Database management
    - Cluster configuration
    - Access control
    - Network settings

- **Fastly** (`fastly.hcl`)
    - CDN configuration
    - Edge computing
    - Cache management
    - Service configuration

- **Control Plane** (`cpln.hcl`)
    - Platform management
    - Resource control
    - Service configuration
    - Access management

- **Signal Sciences** (`sigsci.hcl`)
    - Security monitoring
    - WAF configuration
    - Attack detection
    - Security rules

4. **Utility Providers**

- **Random** (`random.hcl`)
    - Resource name generation
    - Random string creation
    - ID generation
    - Unique resource naming

- **Null** (`null.hcl`)
    - Resource dependencies
    - Conditional resources
    - Resource lifecycle
    - State management

- **Local** (`local.hcl`)
    - Local file operations
    - Local resource management
    - Development testing
    - Local state handling

Module Categories

1. Cloud Provider Modules

- **AWS Modules**
  - Compute resources
  - Storage solutions
  - Networking components
  - Security services
  - Integration services
- **Google Cloud Modules**
  - Compute Engine
  - Cloud Storage
  - Cloud SQL
  - VPC networking
  - IAM services
- **Cloudflare Modules**
  - DNS management
  - CDN configuration
  - Security rules
  - Zone settings

2. Integration Modules

- **GitHub Integration**
  - Repository management
  - Organization settings
  - Access control
  - Webhook configuration
- **Bitbucket Integration**
  - Repository management
  - Project settings
  - Access control
  - Pipeline configuration
- **PagerDuty Integration**
  - Incident management
  - Service configuration
  - Team management
  - Alert routing
- **Lightstep Integration**
  - Distributed tracing
  - Performance monitoring
  - Service metrics

    - Alert configuration

- **Grafana Integration**
    - Dashboard management
    - Data source configuration
    - Alert rules
    - User management

3. Shared Infrastructure

- **Common Components**
    - Shared networking
    - Security services
    - Monitoring tools
    - Logging solutions

- **SafeHealth Specific**
    - Custom integrations
    - Specialized services
    - Business logic
    - Compliance requirements

4. Monitoring & Observability

- **Grafana Dashboards**
    - System metrics
    - Application metrics
    - Business metrics
    - Custom visualizations

- **Lightstep Monitoring**
    - Service tracing
    - Performance metrics
    - Error tracking
    - Dependency analysis

- **PagerDuty Alerts**
    - Incident management
    - On-call scheduling
    - Alert routing
    - Response automation

Module Design Principles

1. **Reusability**
    - Modular design
    - Parameterized configurations
    - Clear interfaces

- Version control

2. **Security**
   - Least privilege access
   - Encryption at rest
   - Secure defaults
   - Audit logging
3. **Maintainability**
   - Consistent structure
   - Clear documentation
   - Version management
   - Dependency tracking
4. **Scalability**
   - Resource optimization
   - Performance considerations
   - Cost management
   - Capacity planning

Module Integration

```
graph TD
    A[Module Integration] --> B[Provider Integration]
    A --> C[Service Integration]
    A --> D[Security Integration]

    B --> E[Cloud Providers]
    B --> F[Infrastructure Services]
    C --> G[Monitoring]
    C --> H[Version Control]
    D --> I[Access Control]
    D --> J[Compliance]
```

- **Provider Integration**
  - Cloud provider specific modules
  - Service-specific configurations
  - Resource management
  - Cost optimization
- **Service Integration**
  - Third-party service modules
  - Monitoring and observability
  - Version control integration
  - Incident management
- **Security Integration**
  - Access control modules
  - Compliance management
  - Security monitoring

- Audit logging

Utility Scripts

The infrastructure includes several utility scripts for common operations and maintenance tasks.

1. S3 Management Scripts

S3 Permanent Delete (`s3-permanent-delete/`)

- **Purpose**: Permanently delete objects from S3 buckets
- **Features**:
  - Versioned bucket support
  - Delete markers removal
  - Batch processing
  - Logging and audit trail
- **Use Cases**:
  - Data cleanup
  - Compliance requirements
  - Storage optimization
  - Cost reduction

S3 Point-in-Time Restore (`s3-point-in-time-restore/`)

- **Purpose**: Restore S3 objects to a specific point in time
- **Features**:
  - Version-based restoration
  - Selective object recovery
  - Batch restoration
  - Progress tracking
- **Use Cases**:
  - Disaster recovery
  - Data recovery
  - Testing and validation
  - Compliance requirements

2. Database Management

MongoDB Cleanup (`mongo-cleanup/`)

- **Purpose**: Clean and maintain MongoDB databases
- **Features**:
  - Collection cleanup
  - Index optimization
  - Data validation
  - Performance monitoring
- **Use Cases**:
  - Database maintenance
  - Performance optimization
  - Storage management
  - Data integrity

3. Infrastructure Management

Terraform Provider Versions (`terraform-provider-versions/`)

- **Purpose**: Manage and update Terraform provider versions
- **Features**:
  - Version compatibility checks
  - Automated updates
  - Dependency management
  - Change tracking
- **Use Cases**:
  - Provider updates
  - Version control
  - Compatibility management
  - Infrastructure maintenance

CRO Release Tagger (`cro-release-tagger/`)

- **Purpose**: Manage and tag releases in the infrastructure
- **Features**:
  - Release versioning
  - Tag management
  - Change tracking
  - Deployment coordination
- **Use Cases**:
  - Release management
  - Version control
  - Deployment tracking
  - Change management

Script Usage Guidelines

1. **Execution Environment**
   - Required permissions
   - Environment variables
   - Dependencies
   - Configuration files
2. **Best Practices**
   - Script testing
   - Error handling
   - Logging
   - Documentation
3. **Security Considerations**
   - Access control
   - Credential management
   - Audit logging
   - Compliance requirements

4. **Maintenance**
    - Regular updates
    - Version control
    - Documentation
    - Testing

Script Integration

```
graph TD
    A[Utility Scripts] --> B[S3 Management]
    A --> C[Database Management]
    A --> D[Infrastructure Management]

    B --> E[Permanent Delete]
    B --> F[Point-in-Time Restore]
    C --> G[MongoDB Cleanup]
    D --> H[Provider Versions]
    D --> I[Release Tagger]
```

- **S3 Management**
    - Data lifecycle
    - Storage optimization
    - Recovery procedures
    - Compliance management
- **Database Management**
    - Performance optimization
    - Data integrity
    - Storage management
    - Maintenance procedures
- **Infrastructure Management**
    - Version control
    - Release management
    - Provider updates
    - Change tracking

Common Operations

This section outlines common operations and maintenance tasks for the infrastructure.

Environment Management

Creating New Environments

1. **Initial Setup**
    - Create new branch
    - Copy existing environment configuration
    - Remove unnecessary components

- Clear configuration files

2. **Configuration Process**
   - Two-phase pull request approach
   - Initial empty configuration
   - Copy necessary configurations
   - Iterative dependency resolution
3. **Best Practices**
   - Environment type consistency
   - Configuration validation
   - Dependency management
   - Documentation updates

Version Management

Pre-commit Hooks

1. **Version Updates**
   - Automatic updates via `pre-commit autoupdate`
   - Installation via `pre-commit install`
   - Configuration in `.pre-commit-config.yaml`
   - Pull request workflow
2. **Hook Types**
   - Code formatting
   - Linting
   - Security checks
   - Documentation validation

Terraform CDK Provider Updates

1. **Package Management**
   - Version checking via `yarn outdated`
   - Package updates via `yarn add`
   - CDKTF CLI updates
   - Documentation maintenance
2. **Configuration Updates**
   - Provider version synchronization
   - CDKTF configuration updates
   - Generated code updates
   - Cache management
3. **Update Process**
   - Cache cleaning
   - CDK updates
   - Lock file updates

- Pull request creation

Terraform Provider Updates

1. **Version Identification**
   - Current version analysis
   - Target version selection
   - Compatibility checking
   - Change impact assessment
2. **Update Process**
   - Module version updates
   - Cache cleaning
   - Jurisdiction updates
   - Lock file updates
3. **Validation**
   - Configuration testing
   - Dependency verification
   - Integration testing
   - Documentation updates

Operation Workflows

```
graph TD
    A[Common Operations] --> B[Environment Management]
    A --> C[Version Management]
    A --> D[Provider Updates]

    B --> E[New Environment]
    B --> F[Configuration]
    C --> G[Pre-commit]
    C --> H[CDK Updates]
    D --> I[Terraform]
    D --> J[CDK Providers]
```

- **Environment Management**
  - Environment creation
  - Configuration management
  - Dependency resolution
  - Validation processes
- **Version Management**
  - Hook updates
  - Package management
  - Configuration updates
  - Documentation maintenance
- **Provider Updates**
  - Version control

- Cache management
- Lock file updates
- Integration testing

Best Practices

1. **Environment Creation**
   - Consistent naming
   - Configuration validation
   - Dependency management
   - Documentation updates
2. **Version Updates**
   - Regular maintenance
   - Compatibility checking
   - Testing procedures
   - Documentation updates
3. **Provider Management**
   - Version synchronization
   - Cache management
   - Lock file maintenance
   - Integration testing
4. **Documentation**
   - Process documentation
   - Change tracking
   - Version history
   - Best practices

Prerequisites and Setup

This section outlines the prerequisites and setup process for working with the infrastructure.

Development Environment

1. Node.js Setup

- **Version**: Node.js 20.x
- **Manager**: nvm (recommended)
- **Version Pinning**: Use version in `cdk/terraform/package.json`
- **Installation**:

```
nvm install 20
nvm use 20
```

2. Terraform Tools

- **Installation**:

```
brew install tfenv tgenv terraform-docs hcledit
```

- **Terraform Version**:

```
tfenv install 1.3.6
tfenv use 1.3.6
```

- **Terragrunt Version**:

```
tgenv install latest
tgenv use latest
```

3. Terraform CDK CLI

- **Version**: Match `cdktf` package version in `cdk/terraform/package.json`
- **Installation**:

```
npm install -g cdktf-cli@<version>
```

4. Development Tools

- **Pre-commit**:

```
brew install pre-commit
pre-commit install
```

- **Google Cloud SDK**:

```
brew install --cask google-cloud-sdk
```

- **AWS CLI**:

```
brew install awscli
```

Repository Structure

```
graph TD
    A[Repository] --> B[CDK Directory]
    A --> C[Global Directory]
    A --> D[Jurisdictions Directory]
    A --> E[Modules Directory]
    A --> F[Providers Directory]
```

```
    A --> G[Scripts Directory]

    B --> H[Terraform CDK]
    B --> I[Admin Stacks]
    C --> J[Global Resources]
    D --> K[Environment Resources]
    E --> L[Terraform Modules]
    F --> M[Provider Configs]
    G --> N[Utility Scripts]
```

1. CDK Directory

- Terraform CDK code
- Terragrunt files
- Jurisdiction admin stacks
- Spacelift stack creation

2. Global Directory

- Shared resources
- Cross-environment resources
- GitHub repositories
- Global configurations

3. Jurisdictions Directory

- Environment-specific resources
- Terragrunt configurations
- Resource definitions
- Environment settings

4. Modules Directory

- Terraform modules
- Reusable components
- Module configurations
- Resource templates

5. Providers Directory

- Provider configurations
- Authentication settings
- Resource access
- Service integrations

6. Scripts Directory

- Utility scripts
- Maintenance tools
- Automation scripts
- Helper functions

Important Operations

1. Stack Dependencies

- **Update Process**:
  - Update `deps.json` files
  - Use `generate:stack-dependencies`
  - Environment-specific updates
  - Dependency validation

2. Control Plane Organization

- **Prerequisites**:
  - Org creator role
  - Safe Health selection
  - Account access
  - Quota management
- **Creation Process**:
  1. Log in to Control Plane
  2. Create organization
  3. Configure settings
  4. Add administrators
- **User Management**:
  1. Access control setup
  2. User invitations
  3. Group assignments
  4. Permission management

3. AWS Account Management

- **Account Updates**:
  - Update `aws.ts`
  - Account configuration
  - Access management
  - Resource allocation

4. GCP Project Management

- **Project Deletion**:
  - 30-day deletion window
  - Project ID preservation
  - Restoration process
  - Resource cleanup
- **Restoration Process**:
  1. Access GCP console
  2. Navigate to IAM & Admin

iii. Manage resources
iv. Restore project

Best Practices

1. **Version Management**
   - Use version managers
   - Pin versions
   - Regular updates
   - Compatibility checks
2. **Environment Setup**
   - Complete prerequisites
   - Tool installation
   - Configuration validation
   - Access management
3. **Resource Management**
   - Proper cleanup
   - Resource tracking
   - State management
   - Dependency handling
4. **Documentation**
   - Process documentation
   - Version tracking
   - Change management
   - Best practices

Root Configuration

The root configuration (`root.hcl`) defines the core settings and mappings used throughout the infrastructure.

Environment Configuration

1. Environment Type Mappings

- **Stage Mapping**:

```
env_type_to_stage = {
  non-production = "non-production"
  demo           = "non-production"
  development    = "non-production"
  quality        = "non-production"
  staging        = "production"
  sandbox        = "production"
  operations     = "production"
  production     = "production"
}
```

- **Short Code Mapping**:

```
env_type_to_short_code = {
  demo           = "demo"
  development    = "dev"
  quality        = "qa"
  sandbox        = "sandbox"
  staging        = "staging"
  non-production = "non-prod"
  operations     = "ops"
  production     = "prod"
}
```

2. Jurisdiction Configuration

- **Data Center Locations**:

```
jurisdiction_to_data_center_locations = {
  "usa"     = "north-america_usa",
  "sf"      = "north-america_usa",
  "opencdx" = "north-america_usa",
  "safecdx" = "north-america_usa",
}
```

Directory Structure

```
graph TD
    A[Root Configuration] --> B[Environment Types]
    A --> C[Jurisdictions]
    A --> D[Resources]
    A --> E[Cloud Providers]

    B --> F[Production]
    B --> G[Non-Production]
    C --> H[USA]
    C --> I[SF]
    D --> J[Workloads]
    D --> K[Subsystems]
    E --> L[Cloudflare]
    E --> M[Google]
```

1. Path Detection

- **CDK Detection**:

```
is_cdk = length(regexall("(?:infrastructure\\/)?cdk", local.original_dir)) > 0
```

- **Jurisdiction Detection**:

```
is_jurisdiction = length(regexall("(?:infrastructure\\/)?jurisdictions", local.original_dir)) > 0
```

2. Resource Paths

- Module Paths:

```
paths = {
  modules_root = "${get_terragrunt_dir()}/modules//terraform/safehealth"
}
```

- Global Directory:

```
global = {
  dir = "${get_terragrunt_dir()}/global"
}
```

Cloud Provider Configuration

1. Cloudflare

- Account Settings:

```
cloudflare = {
  account_name = "The Safe Group Inc Enterprise account"
  account_id   = "08118ea46e759a6cf41f866f2f48151a"
}
```

2. Google Cloud

- Organization Settings:

```
google = {
  org_domain         = "safehealth.me"
  project_suffix     = local.is_jurisdiction ? substr(sha256(local.jurisdiction.name), 0, min(30 - 16 - length
  billing_account_id = "01A34F-FDAC86-883200"
}
```

3. NPM Registry

- Organization:

```
npm = {
  organization = "@safe-health"
}
```

Infrastructure Management

1. Terraform Cloud

- Configuration:

```
terraform_cloud = {
  hostname     = "app.terraform.io"
  organization = "safehealth"
}
```

2. Spacelift

- **API Configuration**:

```
spacelift = {
  api_key_endpoint = "https://safehealth.app.spacelift.io"
}
```

Resource Management

1. Service Configuration

- **Path Pattern**:

```
service = try(regex("workloads\\/(?P<network_type>.*?)\\/(?P<system_name>.*?)\\/(?P<version>v\\d.*?)\\/(?P<nam
```

2. Subsystem Configuration

- **Path Pattern**:

```
subsystem = try(regex("workloads\\/(?P<network_type>.*?)\\/subsystems\\/(?P<name>.*?)\\/(?P<app_owner>.*?)\\/(
```

Best Practices

1. **Environment Management**
    - Consistent naming
    - Clear stage mapping
    - Environment isolation
    - Resource organization
2. **Jurisdiction Management**
    - Data center mapping
    - Resource allocation
    - Access control
    - Compliance requirements
3. **Cloud Provider Integration**
    - Account management
    - Resource organization
    - Billing configuration
    - Access control

4. **Resource Organization**
    - Clear structure
    - Consistent naming
    - Resource grouping
    - Access management

Backend Configuration

The backend configuration (`backend.hcl`) defines how Terraform state is stored and managed using Cloudflare R2 as the backend storage.

State Storage Configuration

1. Backend Generation

```
generate "backend" {
  path      = "backend.tf"
  if_exists = "overwrite_terragrunt"
  contents  = <<EOF
terraform {
  backend "s3" {
    bucket         = "terraform-state"
    key            = "${path_relative_to_include()}/terraform.tfstate"
    region         = "us-east-1"
    ...
  }
}
EOF
}
```

2. Cloudflare R2 Integration

- **Authentication**:

```
  access_key = "${get_env("CLOUDFLARE_R2_ACCESS_KEY")}"
  secret_key = "${get_env("CLOUDFLARE_R2_SECRET_KEY")}"
```

- **Endpoint Configuration**:

```
  endpoint = "https://${local.root.cloudflare.account_id}.r2.cloudflarestorage.com"
```

State Management

```
graph TD
    A[State Management] --> B[Storage Backend]
    A --> C[Authentication]
    A --> D[Configuration]

    B --> E[Cloudflare R2]
    B --> F[State Files]
    C --> G[Access Keys]
    C --> H[Permissions]
```

```
    D --> I[Validation]
    D --> J[Encryption]
```

1. Storage Settings

- **Bucket**: `terraform-state`
- **Region**: `us-east-1`
- **Key Pattern**: Path-based state file naming
- **Encryption**: Enabled by default

2. Backend Features

- **Skip Validations**:

```
skip_credentials_validation = true
skip_region_validation      = true
skip_metadata_api_check     = true
```

- **Security**:

```
encrypt = true
```

Best Practices

1. **State Management**
    - Centralized storage
    - Encryption at rest
    - Access control
    - State locking
2. **Security**
    - Secure credentials
    - Environment variables
    - Access restrictions
    - Audit logging
3. **Configuration**
    - Clear organization
    - Version control
    - Documentation
    - Backup strategy
4. **Operations**
    - State monitoring
    - Access management
    - Backup procedures
    - Recovery plans

NPM Registry Configuration

The NPM registry configuration (`npm_registry_config.hcl`) manages package management settings and authentication for the infrastructure's Node.js dependencies.

Package Management

1. Package.json Generation

```
generate "package" {
  path              = "package.json"
  if_exists         = "overwrite"
  disable_signature = true
  contents          = "{}"
}
```

2. NPM Configuration

```
generate "npmrc" {
  path      = ".npmrc"
  if_exists = "overwrite_terragrunt"
  contents  = <<EOF
registry=https://registry.yarnpkg.com/

@safe-health:registry=https://npm.pkg.github.com
//npm.pkg.github.com/:_authToken=${get_env("NPM_AUTH_TOKEN")}
always-auth=true
EOF
}
```

Registry Structure

```
graph TD
    A[NPM Configuration] --> B[Package Management]
    A --> C[Authentication]
    A --> D[Registry Settings]

    B --> E[Package.json]
    B --> F[Dependencies]
    C --> G[Auth Token]
    C --> H[Access Control]
    D --> I[Public Registry]
    D --> J[Private Registry]
```

Configuration Components

1. Registry Settings

- **Public Registry**: `registry.yarnpkg.com`
- **Private Registry**: `npm.pkg.github.com`
- **Organization Scope**: `@safe-health`
- **Authentication**: GitHub Package Registry token

2. Security Features

- **Authentication Token**: Environment variable based
- **Always Auth**: Enabled for all requests
- **Scoped Access**: Organization-specific registry
- **Token Management**: Secure environment variables

Best Practices

1. **Package Management**
   - Version control
   - Dependency tracking
   - Security updates
   - Audit logging
2. **Authentication**
   - Secure tokens
   - Environment variables
   - Access control
   - Token rotation
3. **Registry Usage**
   - Scoped packages
   - Version pinning
   - Dependency management
   - Update strategy
4. **Security**
   - Token security
   - Access management
   - Audit procedures
   - Vulnerability scanning

Pre-commit Configuration

The pre-commit configuration (`.pre-commit-config.yaml`) defines code quality checks and formatting rules that run before each commit.

Hook Configuration

1. Terraform Hooks

```yaml
- repo: https://github.com/antonbabenko/pre-commit-terraform
  rev: v1.96.1
  hooks:
    - id: terraform_fmt
    - id: terraform_wrapper_module_for_each
    - id: terraform_docs
    - id: terraform_tflint
```

```
      - id: terraform_validate
      - id: terragrunt_fmt
```

2. General Hooks

```
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v4.6.0
  hooks:
    - id: check-merge-conflict
```

3. Ticket Integration

```
- repo: https://github.com/milin/giticket
  rev: v1.4
  hooks:
    - id: giticket
      args: ['--mode=regex_match', '--regex=[A-Z]+-\d+', '--format={ticket} {commit_msg}']
```

Hook Structure

```
graph TD
    A[Pre-commit Hooks] --> B[Terraform]
    A --> C[General]
    A --> D[Ticket Integration]

    B --> E[Formatting]
    B --> F[Validation]
    B --> G[Documentation]
    C --> H[Merge Conflicts]
    D --> I[Ticket Format]
    D --> J[Commit Messages]
```

Hook Categories

1. Terraform Hooks

- **Formatting**:
  - `terraform_fmt` : Code formatting
  - `terragrunt_fmt` : Terragrunt formatting
- **Documentation**:
  - `terraform_docs` : Documentation generation
  - Arguments: `--lockfile=false`
- **Linting**:

```
  terraform_tflint:
    args:
      - '--args=--only=terraform_deprecated_interpolation'
      - '--args=--only=terraform_deprecated_index'
      - '--args=--only=terraform_unused_declarations'
      - '--args=--only=terraform_comment_syntax'
```

```
        - '--args=--only=terraform_documented_outputs'
        - '--args=--only=terraform_documented_variables'
        - '--args=--only=terraform_typed_variables'
        - '--args=--only=terraform_module_pinned_source'
        - '--args=--only=terraform_naming_convention'
        - '--args=--only=terraform_required_version'
        - '--args=--only=terraform_required_providers'
        - '--args=--only=terraform_standard_module_structure'
        - '--args=--only=terraform_workspace_remote'
```

- **Validation**:
  - `terraform_validate`: Configuration validation
  - Excludes:
  - CDK directory
  - Global directory
  - Jurisdictions directory
  - Specific modules

2. General Hooks

- **Merge Conflicts**:
  - Check for merge conflict markers
  - Prevent accidental commits
  - Clean code base
  - Version control safety

3. Ticket Integration

- **Format**:
  - Regex pattern: `[A-Z]+-\d+`
  - Message format: `{ticket} {commit_msg}`
  - Mode: `regex_match`
  - Ticket validation

Best Practices

1. **Code Quality**
   - Consistent formatting
   - Documentation standards
   - Linting rules
   - Validation checks

2. **Version Control**
   - Clean commits
   - Ticket tracking
   - Message formatting
   - Conflict prevention

3. **Documentation**
   - Automated generation
   - Standard format
   - Comprehensive coverage

- Regular updates

4. **Workflow**
   - Pre-commit checks
   - Validation process
   - Error handling
   - Quick feedback

Editor Configuration

The editor configuration (`.editorconfig`) ensures consistent coding styles across different editors and IDEs.

Configuration Settings

1. Global Settings

```
root = true

[*]
charset = utf-8
indent_style = space
indent_size = 2
insert_final_newline = true
trim_trailing_whitespace = true
```

2. Markdown Settings

```
[*.md]
max_line_length = off
trim_trailing_whitespace = false
```

3. IDE Settings

```
[.idea/*.xml]
insert_final_newline = false
```

Configuration Structure

```
graph TD
    A[Editor Config] --> B[Global Settings]
    A --> C[File-specific Settings]
    A --> D[IDE Settings]

    B --> E[Character Set]
    B --> F[Indentation]
    C --> G[Markdown]
    C --> H[XML]
    D --> I[IntelliJ]
    D --> J[Line Endings]
```

Setting Categories

1. Global Settings

- **Character Set**: UTF-8
- **Indentation**:
  - Style: Space
  - Size: 2 spaces
- **Line Endings**:
  - Final newline
  - Trim trailing whitespace

2. File-specific Settings

- **Markdown Files**:
  - No line length limit
  - Preserve trailing whitespace
- **IDE Files**:
  - No final newline in XML
  - Specific IDE configurations

Best Practices

1. **Code Style**
   - Consistent formatting
   - Clear indentation
   - Clean line endings
   - Character encoding
2. **File Management**
   - Type-specific rules
   - IDE integration
   - Version control
   - Collaboration
3. **Documentation**
   - Markdown formatting
   - Readability
   - Consistency
   - Maintainability
4. **Development**
   - Editor support
   - IDE compatibility
   - Team standards
   - Code quality

Docker Configuration

The Docker configuration (`.dockerignore`) specifies which files and directories should be excluded from Docker build context.

Ignore Patterns

1. Node.js Files

```
node_modules
```

2. Terraform Files

```
.terragrunt-cache
.terraform
*.tfstate
*.tfstate.*
*.auto.tfvars

override.tf
override.tf.json
*_override.tf
*_override.tf.json

.terraformrc
terraform.rc
```

Configuration Structure

```
graph TD
    A[Docker Ignore] --> B[Node.js]
    A --> C[Terraform]
    A --> D[Override Files]

    B --> E[Dependencies]
    C --> F[Cache]
    C --> G[State Files]
    D --> H[Local Overrides]
    D --> I[CLI Config]
```

Ignore Categories

1. Node.js Files

- **Dependencies**:
  - `node_modules` directory
  - Package management
  - Build artifacts
  - Cache files

2. Terraform Files

- **Cache and State**:

    - Terragrunt cache
    - Terraform cache
    - State files
    - Auto variables

- **Override Files**:
    - Local overrides
    - JSON overrides
    - Resource overrides
    - Configuration overrides

- **CLI Configuration**:
    - Terraform RC
    - Local settings
    - User preferences
    - Environment config

Best Practices

1. **Build Context**
    - Minimize size
    - Exclude unnecessary files
    - Speed up builds
    - Optimize performance

2. **Security**
    - Exclude sensitive files
    - State file protection
    - Configuration security
    - Access control

3. **Performance**
    - Cache management
    - Build optimization
    - Resource efficiency
    - Context size

4. **Maintenance**
    - Regular updates
    - Pattern review
    - Documentation
    - Version control

Git Configuration

The Git configuration (`.gitignore`) specifies which files and directories should be excluded from version control.

Ignore Patterns

1. Build Output

```
# compiled output
dist
tmp
out-tsc
bin
obj
```

2. Dependencies

```
# dependencies
node_modules
```

3. IDE Files

```
# IDE - VSCode
.vscode/*
!.vscode/settings.json
!.vscode/tasks.json
!.vscode/launch.json
!.vscode/extensions.json

# IDE - VS
.vs/*

# JetBrains
.idea/aws.xml
.idea/protobuf.xml
.idea/copilot
.idea/codeStyles
.idea/dbNavigator.xml
```

4. Terraform Files

```
# Terraform
.terragrunt-cache
.terraform
*.tfstate
*.tfstate.*
*.auto.tfvars

# Override files
override.tf
override.tf.json
*_override.tf
*_override.tf.json

# CLI configuration
.terraformrc
terraform.rc
```

5. System and Temporary Files

```
# System Files
.DS_Store
Thumbs.db

# Temp files
*~

# Env
.env
.env.*
```

Configuration Structure

```
graph TD
    A[Git Ignore] --> B[Build]
    A --> C[Dependencies]
    A --> D[IDE]
    A --> E[Terraform]
    A --> F[System]

    B --> G[Compiled]
    B --> H[Output]
    C --> I[Node.js]
    D --> J[VSCode]
    D --> K[Visual Studio]
    D --> L[JetBrains]
    E --> M[Cache]
    E --> N[State]
    E --> O[Overrides]
    F --> P[OS Files]
    F --> Q[Temp Files]
    F --> R[Environment]
```

Ignore Categories

1. Build and Output Files

- **Compiled Output**:
  - Distribution files
  - Temporary files
  - Build artifacts
  - Binary files

2. Development Dependencies

- **Node.js**:
  - Module directory
  - Debug logs
  - Error logs
  - Cache files

3. IDE Configuration

- **VSCode**:

    - Workspace settings
    - Excluded patterns
    - Included settings
    - Extension configs

- **Visual Studio**:

    - Solution files
    - User settings
    - Project files
    - Cache files

- **JetBrains**:

    - AWS configuration
    - Protobuf settings
    - Copilot settings
    - Code styles

4. Infrastructure Files

- **Terraform**:
    - Cache directories
    - State files
    - Auto variables
    - Override files

5. System Files

- **Operating System**:

    - macOS files
    - Windows files
    - Temporary files
    - Backup files

- **Environment**:

    - Environment files
    - Local settings
    - Secrets
    - Configuration

Best Practices

1. **Version Control**

    - Exclude build artifacts
    - Protect sensitive data
    - Manage dependencies
    - Control file size

2. **Security**

- Exclude credentials
- Protect state files
- Secure configurations
- Environment isolation

3. **Performance**

- Repository size
- Clone efficiency
- Cache management
- Build optimization

4. **Development**

- IDE integration
- Local settings
- Team standards
- Collaboration

Conclusion

This infrastructure provides a robust, scalable, and secure foundation for managing cloud resources across multiple jurisdictions and environments. The hierarchical structure ensures clear separation of concerns while maintaining flexibility and scalability.

This comprehensive documentation outlines the infrastructure's architecture, components, and best practices. The infrastructure is built with scalability, security, and maintainability in mind.

Key Components

```
graph TD
    A[Infrastructure] --> B[Core Components]
    A --> C[Configuration]
    A --> D[Development Tools]
    A --> E[Best Practices]

    B --> F[CDK]
    B --> G[Modules]
    B --> H[Providers]
    C --> I[Environment]
    C --> J[Backend]
    C --> K[Registry]
    D --> L[Pre-commit]
    D --> M[Editor Config]
    D --> N[Docker]
    E --> O[Security]
    E --> P[Performance]
    E --> Q[Maintenance]
```

Infrastructure Overview

1. **Core Components**

- CDK structure and integration
- Module organization and reuse
- Provider configuration and management

- Script utilities and automation

2. **Configuration Management**
   - Environment configuration
   - Backend state management
   - Registry and package management
   - Development tools setup
3. **Development Workflow**
   - Code quality tools
   - Version control
   - Continuous integration
   - Deployment automation
4. **Best Practices**
   - Security considerations
   - Performance optimization
   - Maintenance procedures
   - Team collaboration

Key Features

1. **Scalability**
   - Modular design
   - Resource optimization
   - Performance monitoring
   - Load management
2. **Security**
   - Access control
   - Data protection
   - Compliance
   - Audit logging
3. **Maintainability**
   - Clear structure
   - Documentation
   - Version control
   - Automation
4. **Efficiency**
   - Resource management
   - Cost optimization
   - Development speed
   - Team productivity

Future Considerations

1. **Infrastructure Evolution**
   - Regular updates
   - New features
   - Technology adoption
   - Performance improvements
2. **Security Enhancement**
   - Threat monitoring
   - Vulnerability management
   - Access control
   - Compliance updates
3. **Development Experience**
   - Tool improvements
   - Workflow optimization
   - Documentation updates
   - Team training
4. **Resource Optimization**
   - Cost management
   - Performance tuning
   - Resource allocation
   - Efficiency improvements

Contact and Support

For questions, issues, or contributions:

1. Review the documentation
2. Check existing issues
3. Follow contribution guidelines
4. Contact the infrastructure team

This documentation is maintained by the infrastructure team and is regularly updated to reflect changes and improvements in the infrastructure.