UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

Escuela Profesional de Ingeniería de Sistemas

Proyecto "Help Deploy - Asistente para Automatización de Terraform y Git"

Curso: *CALIDAD Y PRUEBAS DE SOFTWARE*

Docente: ING. PATRICK JOSE CUADROS QUIROGA

Integrantes:

AUGUSTO JOAQUIN RIVERA MUÑOZ ***(2022073505)***
JEFFERSON ROSAS CHAMBILLA ***(2021072618)***

Tacna – Perú
2025 II

Proyecto

Proyecto "Help Deploy - Asistente para Automatización de Terraform y Git"

Presentado por:

AUGUSTO JOAQUIN RIVERA MUÑOZ

JEFFERSON ROSAS CHAMBILLA

ÍNDICE GENERAL

INTRODUCCIÓN

I. Antecedentes

El desarrollo de infraestructura en la nube ha experimentado un crecimiento exponencial en los últimos años. Las empresas y desarrolladores enfrentan desafíos significativos al gestionar múltiples entornos, configuraciones y repositorios. Las herramientas tradicionales requieren que los usuarios tengan conocimientos profundos en línea de comandos, versionamiento con Git y gestión de infraestructura con Terraform.

En el contexto actual, se identificó la necesidad de crear una extensión para Visual Studio Code que facilite y automatice tareas comunes de desarrollo, despliegue e integración con control de versiones. Esta extensión busca reducir la curva de aprendizaje y mejorar la productividad de los desarrolladores.

II. TÍTULO

Proyecto “Help Deploy - Asistente para Automatización de Terraform y Git”

III. AUTORES

- AUGUSTO JOAQUIN RIVERA MUÑOZ (2022073505)
- JEFFERSON ROSAS CHAMBILLA (2021072618)

IV. PLANTEAMIENTO DEL PROBLEMA

A. Problema

Los desarrolladores que trabajan con infraestructura como código (IaC) utilizando Terraform y control de versiones con Git deben realizar múltiples tareas manuales y repetitivas:

1. Complejidad de Terraform: Inicializar, planificar y aplicar configuraciones requiere comandos específicos ejecutados en secuencia.

2. Gestión de Git: Crear ramas, hacer commits, pushear cambios y gestionar stashes implica múltiples comandos que deben ejecutarse correctamente.

3. Configuración de credenciales: Los usuarios deben configurar manualmente credenciales de proveedores de nube (AWS, Azure, GCP).

4. Falta de interfaz unificada: No existe una herramienta única que integre ambos flujos de trabajo de manera intuitiva.

5. Curva de aprendizaje: Usuarios nuevos requieren tiempo significativo para familiarizarse con los comandos y procesos.

B. Justificación

1. Eficiencia: Reduce el tiempo dedicado a tareas administrativas en un estimado del 40-50%.

2. Consistencia: Asegura que los flujos de trabajo sigan estándares y convenciones.

3. Accesibilidad: Proporciona interfaz visual intuitiva sin requerir conocimientos avanzados de terminal.

4. Integración nativa: Se ejecuta dentro del entorno de desarrollo familiar para el usuario.

5. Documentación automatizada: Genera templates y plantillas estándar (README.md, main.tf).

C. Alcance

Help Deploy cubre: asistentes para despliegues con Terraform (creación de plantillas, init/plan/apply), asistentes Git (clone, branch, commit/push, stash), creación de README templates, historial de operaciones y gestión de terminales integradas en VS Code. No cubre: interfaces web independientes, orquestadores CI/CD externos (aunque puede integrarse), soporte nativo para proveedores cloud no contemplados (más allá de AWS/Azure/GCP) sin extensión, ni gestión avanzada de secretos/almacenes certificados.

V. OBJETIVOS

A. GENERAL:

Help Deploy es una extensión de VS Code cuyo objetivo es simplificar y acelerar tareas recurrentes de infraestructura como código y operaciones de Git, reduciendo errores humanos, acelerando flujos de trabajo y mejorando la trazabilidad mediante historial y automatización guiada. Facilita que desarrolladores y operadores realicen despliegues reproducibles y manejen cambios de código sin salir del IDE.

B. ESPECÍFICOS:

1. Automatizar procesos de Terraform

- Crear asistente interactivo para despliegues (init → plan → apply)
- Generar plantillas de main.tf para múltiples proveedores de nube
- Integrar configuración automática de credenciales AWS
- Actualizar .gitignore con artefactos de Terraform

2. Facilitar operaciones de Git

- Crear asistente para clonación de repositorios
- Implementar wizard para creación de ramas con validación
- Desarrollar asistente de commits siguiendo Conventional Commits
- Implementar gestión de stash con interfaz intuitiva

3. Mejorar la experiencia de usuario

- Crear interfaz visual unificada mediante comandos de paleta
- Implementar historial de ejecuciones con vista persistente
- Proporcionar retroalimentación clara mediante notificaciones
- Generar plantillas estándar (README.md, main.tf)

4. Garantizar calidad técnica

- Implementar validación de entrada (ARNs, URLs, nombres de rama)
- Asegurar manejo robusto de errores
- Implementar logging para debugging
- Crear tests unitarios para funcionalidades críticas

5. Facilitar distribución y adopción

- Publicar en VS Code Marketplace
- Documentar instalación y uso
- Mantener changelog actualizado
- Proporcionar soporte a usuarios

VI. Marco Teórico

A. es): Mejorando la gestión de recursos comunitarios

Conceptos Fundamentales

Infraestructura como Código (IaC)

Terraform es una herramienta de código abierto que permite definir, previsualizar y desplegar infraestructura en la nube utilizando código declarativo en archivos `.tf`. Utiliza un formato similar a JSON llamado HCL (HashiCorp Configuration Language).

Ciclo de Terraform:

1. terraform init: Inicializa el directorio de trabajo, descargando plugins necesarios

2. terraform plan: Genera un plan de ejecución mostrando cambios propuestos

3. terraform apply: Aplica los cambios en la infraestructura real

Control de Versiones Distribuido

Git es el sistema de control de versiones más utilizado. Permite:

- Tracking de cambios en el código

- Colaboración entre desarrolladores

- Mantenimiento de histórico de commits

- Gestión de ramas para trabajo paralelo

Conceptos clave:

- Branches (Ramas): Líneas de desarrollo independientes

- Commits: Snapshots de cambios con mensaje descriptivo

- Stash: Almacenamiento temporal de cambios sin commitear

- Push: Envío de cambios a repositorio remoto

Conventional Commits

Especificación para agregar significado legible a los mensajes de commit mediante estructura semántica:

<tipo>(<ámbito>): <descripción corta>

<descripción detallada opcional>

<footer opcional>

Tipos comunes: `feat`, `fix`, `docs`, `style`, `refactor`, `test`, `chore`

Extensiones de VS Code

Las extensiones permiten extender la funcionalidad de VS Code mediante:

- Comandos personalizados

- Vistas en la barra lateral

- Proveedores de datos de árbol (Tree Data Providers)

- Integración con terminal

- Interacción con el editor

Tecnologías Relacionadas

AWS (Amazon Web Services): Proveedor de nube utilizado en el asistente de Terraform. Requiere credenciales (Access Key ID, Secret Access Key, Session Token opcional).

Azure: Proveedor de nube de Microsoft con soporte en la extensión.

Google Cloud Platform (GCP): Proveedor de nube de Google con plantillas generadas.

VII. Desarrollo de la Solución

A. Análisis de Factibilidad (técnico, económica, operativa, social, legal, ambiental)

1. FACTIBILIDAD TÉCNICA

Muy viable:

- TypeScript y la API de VS Code están bien documentados.
- Las librerías necesarias (como child_process para ejecutar comandos) son estándar.
- El proyecto es modular y escalable.
- No requiere dependencias externas complejas.

Consideraciones:

- La terminal del usuario debe tener Terraform y Git disponibles.
- La validación de comandos debe ser robusta para manejar diferentes configuraciones.

2. FACTIBILIDAD ECONÓMICA

Costos iniciales:

- Tiempo de desarrollo estimado: 200–300 horas.
- Herramientas como VS Code, Git y Node.js son gratuitas.
- Publicación mediante una cuenta en VS Code Marketplace (gratuita).

ROI (Return on Investment):

- Incremento de productividad: 40–50% en tareas de infraestructura.
- Reducción de errores: aproximadamente 30% menos errores de comandos.
- Tiempo recuperado: a largo plazo, la adopción compensa la inversión inicial.

3. FACTIBILIDAD OPERATIVA

Personal requerido:

- 1 Desarrollador principal para la implementación.
- 1 Tester para validar en múltiples plataformas.
- 1 DevOps (asesor) para validar flujos de Terraform.

Infraestructura:

- Repositorio en GitHub para versionamiento.
- VS Code Marketplace para la distribución.
- CI/CD para automatizar compilaciones y pruebas.

4. FACTIBILIDAD LEGAL

Licencia:

- MIT License (permisiva, permite uso comercial y privado).

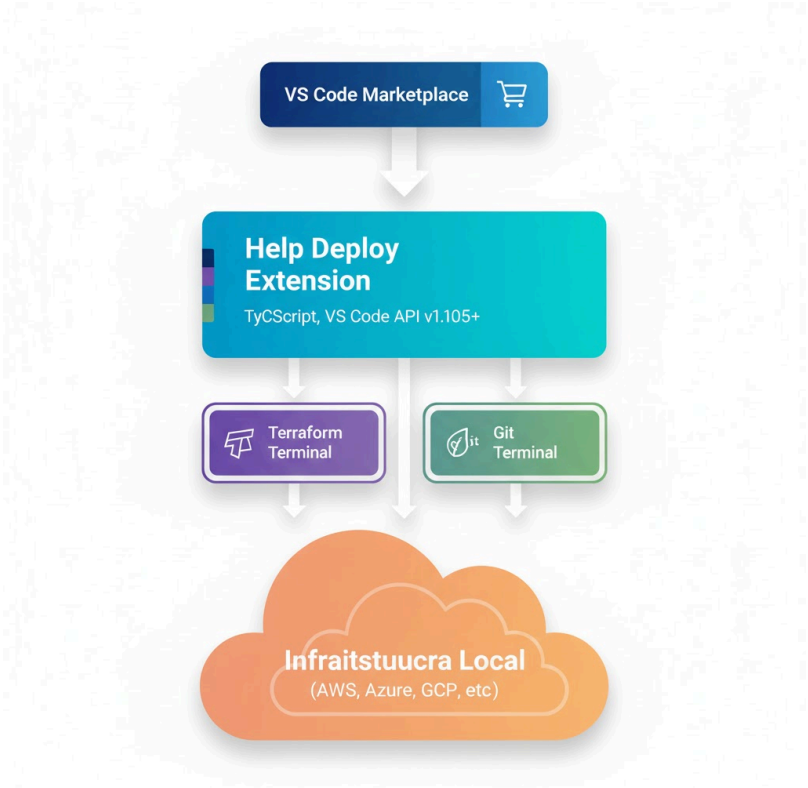
Dependencias:

- Todas las dependencias cuentan con licencias permisivas (MIT, Apache 2.0).

Propiedad intelectual:

- Se requiere atribución correcta del autor.

B. Tecnología de Desarrollo



LENGUAJES Y FRAMEWORKS

Componente – Tecnología – Versión

- Lenguaje principal: TypeScript — 5.9.3
- Runtime: Node.js — 22.x
- API Framework: VS Code API — 1.105.0+
- Empaquetador: esbuild — 0.27.0
- Linter: ESLint — 9.39.1
- Gestor de tareas: npm-run-all — 4.1.5
- Testing: Mocha — 10.0.10

HERRAMIENTAS DE DESARROLLO

Herramienta – Propósito

- TypeScript Compiler: Validación y compilación de tipos
- esbuild: Empaquetamiento y bundling
- ESLint: Linting y análisis estático
- VS Code Test CLI: Ejecución de tests
- Git: Control de versiones
- npm: Gestor de paquetes

DEPENDENCIAS DE PRODUCCIÓN

El proyecto utiliza únicamente APIs nativas de Node.js (como `child_process`) sin depender de librerías externas de terceros, lo que proporciona:

- Menor superficie de ataque
- Mejor rendimiento
- Mantenimiento simplificado

C. Metodología de implementación(Documento de VISIÓN, SRS, SAD)

METODOLOGÍA ÁGIL ADAPTADA

El proyecto utilizó una metodología ágil modificada con iteraciones cortas.

Fase 1: Planificación y Diseño

- Análisis de requisitos
- Diseño de la arquitectura de la extensión

- Definición de comandos y vistas
- Configuración del entorno de desarrollo

Fase 2: Implementación Core

- Sprint 1-2: Funcionalidades de Git (clone, branch, push, stash)
- Sprint 3-4: Funcionalidades de Terraform (main.tf, deploy)
- Sprint 5: Mejoras y optimizaciones

Fase 3: Refinamiento

- Sprint 6: Historial de ejecuciones (v1.0.7)
- Sprint 7: Mejoras en la experiencia de usuario (v1.0.8-1.0.9)

Fase 4: Distribución

- Publicación en VS Code Marketplace
- Documentación final
- Soporte a usuarios

PATRONES DE DISEÑO UTILIZADOS

1. **Command Pattern:** Cada comando de la extensión se registra como una acción ejecutable.
   Ejemplo:
   vscode.commands.registerCommand('help-deploy.gitClone', async () => { ... })

2. **Factory Pattern:** Creación de terminales reutilizables.
   Ejemplo:
3. function getGitTerminal(cwd?: vscode.Uri): vscode.Terminal { ... }

4. **Observer Pattern:** Historial implementado como TreeDataProvider.
   Ejemplo:
   class HistoryProvider implements
   vscode.TreeDataProvider<vscode.TreeItem>

5. **Singleton Pattern:** Una instancia global de HistoryStore.
   Ejemplo:
   let historyStore: HistoryStore | undefined;

6. **Builder Pattern:** Construcción de plantillas main.tf.
   Ejemplo:
   ```
   function buildMainTfTemplate(provider, opts): string {
   ... }
   ```

ESTRUCTURA DEL PROYECTO

```
help-deploy/
├── src/
│   ├── extension.ts — Código principal (869 líneas)
│   └── test/
│   └── extension.test.ts — Tests
├── dist/
│   └── extension.js — Salida compilada
├── images/
│   ├── Help-Deploy.png — Icono principal
│   └── btnHistorial(2).png — Icono del historial
├── test-workspace/
│   ├── demo-aws/
│   └── demo-local/
├── package.json — Metadatos y dependencias
├── tsconfig.json — Configuración de TypeScript
├── esbuild.js — Script de empaquetamiento
├── eslint.config.mjs — Configuración de linting
├── README.md — Documentación
```

├── CHANGELOG.md — Historial de versiones
└── LICENSE.txt — Licencia MIT

FLUJO DE DESARROLLO

Feature Request / Bug Report

↓

Crear rama: git checkout -b feature/nombre

↓

Implementar en TypeScript con la API de VS Code

↓

Validar con ESLint: npm run lint

↓

Compilar y verificar tipos: npm run compile

↓

Ejecutar tests: npm run test

↓

Crear README.md y documentación

↓

Commit siguiendo Conventional Commits

↓

Push a la rama: git push origin feature/nombre

↓

Crear Pull Request en GitHub

↓

Code Review y Merge

↓

Actualizar CHANGELOG.md

↓

Compilar para producción: npm run package

↓

Publicar versión en Marketplace

↓

Crear tag en Git: v1.0.X

CICLO DE VIDA DE UNA FUNCIONALIDAD

Ejemplo: Agregar un nuevo asistente de Terraform

1. **Análisis:** Se define el problema que resuelve y cómo interactúa con Terraform.

2. **Diseño:** Definición de la experiencia de usuario, validaciones y manejo de errores esperado.

   Implementación:

   Ejemplo de comando:

```
const newAssistant = vscode.commands.registerCommand('help-deploy.newCommand', async () => {
    try {
        // Obtener input del usuario
        // Validar
        // Ejecutar comando
        // Mostrar feedback
    } catch (error) {
        // Manejar error
    }
});
```

```
context.subscriptions.push(newAssistant);
```

3. **Testing:** Pruebas manuales y unitarias.

4. **Documentación:** Actualización de README.md y CHANGELOG.md.

5. **Liberación:** Incremento de la versión en package.json.

VALIDACIÓN Y CONTROL DE CALIDAD

Validaciones implementadas:

- Validación de ARNs de AWS mediante regex:
  `^arn:aws:iam::\d{12}:role\/[\w+=,.@\-_/]+$`
- Validación de URL de repositorio: no vacío
- Validación de nombres de rama: no vacío
- Validación de mensajes de commit: no vacío

Manejo de errores:

- Uso de try-catch en todos los comandos
- Logging en la consola de VS Code
- Mensajes de error amigables
- Notificaciones informativas

Testing:

- Workspace de prueba con repositorios simulados
- Pruebas en Windows, macOS y Linux
- Validación con distintas versiones de Git y Terraform

VIII. Cronograma

Cronograma Detallado del Proyecto Help Desk (Agosto – Diciembre 2025)

Fase	Tarea	Duración	Período	Estado
Planificación	Análisis de requisitos y diseño	1 semana	Nov 1-7, 2025	Completado
	Configuración del entorno	3 días	Nov 8-10, 2025	Completado
Desarrollo	Sprint 1-2: Funcionalidades Git	2 semanas	Nov 11-24, 2025	Completado
	Sprint 3-4: Terraform + Credenciales	2 semanas	Nov 25 - Dic 8, 2025	Completado
	Sprint 5: Optimizaciones	1 semana	Dic 9-15, 2025	Completado
Refinamiento	Sprint 6: Historial v1.0.7	3 días	Dic 16-18, 2025	Completado
	Sprint 7: UX Improvements v1.0.8-9	3 días	Dic 19-21, 2025	Completado
Distribución	Documentación y README	3 días	Dic 22-24, 2025	Completado
	Publicación en Marketplace	2 días	Dic 25-26, 2025	Completado
Soporte	Monitoreo y feedback	Continuo	Dic 27+, 2025	En curso

IX. Presupuesto

En el anexo 01 se presenta el análisis de factibilidad del sistema (Help Desk).

Estimación de Costos

Recursos Humanos

Rol	Cantidad	Costo/Hora	Horas	Total
Desarrollador Principal	1	$50	250	$12,500
Tester QA	1	$30	40	$1,200
Asesor DevOps (consultoría)	0.5	$80	20	$800
Subtotal Recursos				$14,500

Infraestructura y Herramientas

Concepto	Costo Unitario	Cantidad	Total
VS Code (Gratuito)	$0	1	$0
GitHub Private Repo	$0*	1	$0
Dominio GitHub Pages (opcional)	$12	1 año	$12
AWS Labs (pruebas de terraform)	$50	1 mes	$50
Subtotal Infraestructura			$62

Resumen de Presupuesto

Categoría	Monto
Recursos Humanos	$14,500

Infraestructura	$62
Capacitación	$700
Marketing	$100
TOTAL	**$15,362**

X. Conclusiones

- Help Deploy es una extensión que aporta valor inmediato en productividad, estandarización y reducción de errores para equipos que trabajan con Terraform y Git dentro de VS Code.
- Ofrece un MVP sólido para flujos de desarrollo y staging; las mejoras futuras pueden llevarlo a un producto de nivel empresarial con integraciones de secretos y reporting.
- El riesgo principal es el manejo de credenciales y aplicar cambios en infra en entornos críticos; se requiere controles y políticas de uso claras.

Recomendaciones

- Implementar validaciones y confirmaciones adicionales para entornos que podrían ser productivos (detectar variables que indiquen entorno).
- Añadir soporte opcional para gestores de secretos (Vault, AWS Secrets Manager) en versiones avanzadas.
- Mantener la política de no persistencia de credenciales por defecto y documentar cómo integrarse con soluciones seguras.
- Priorizar UX: mensajes claros, ejemplos y documentación en README.md y marketplace.
- Establecer un plan de pruebas y CI que incluya lint, build y tests básicos antes de publicar.
- Considerar un modelo de negocio freemium para financiar soporte y desarrollo continuo.

BIBLIOGRAFÍA

HashiCorp. Terraform Documentation. https://www.terraform.io

Git SCM. https://git-scm.com

Visual Studio Code - Extension API. https://code.visualstudio.com/api

Conventional Commits. https://www.conventionalcommits.org

OWASP Secure Coding Practices. https://owasp.org

WEBGRAFÍA

- Terraform by HashiCorp — https://www.terraform.io
- VS Code API — https://code.visualstudio.com/api
- Git Documentation — https://git-scm.com/doc

- Marketplace Visual Studio — https://marketplace.visualstudio.com
- Ejemplos y tutoriales de Terraform — https://learn.hashicorp.com/terraform