© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715267

Autonomous Multi-Agent System for Cloud Architecture Design and Infrastructure Deployment

RAUNAK B SINHA
Master's in Data Science and AI, BITS Pilani, India

Abstract - Cloud infrastructure design and deployment traditionally require significant expertise in cloud services, networking, security, and Infrastructure-as-Code (IaC). Translating high-level business requirements into production-ready infrastructure can take days of manual effort and often involve multiple domain experts. This research presents Cloud Infrastructure Crew, an autonomous multi-agent system built using the CrewAI framework that automates cloud architecture design, IaC generation, and deployment validation. The system utilizes three specialized Large Language Model (LLM) powered agents that collaborate sequentially to convert business requirements into infrastructure artifacts such as architecture diagrams, Terraform configuration files, and deployment reports. A human-in-the-loop approval mechanism ensures architectural accuracy before infrastructure generation, also added at agents steps. Experimental evaluation shows that the proposed system significantly reduces infrastructure planning time from several days to minutes while maintaining transparency, auditability, and extensibility. The architecture is designed to be cloud-agnostic and supports integration with multiple LLM providers.

I. INTRODUCTION

Cloud computing has transformed the way modern applications are designed, deployed, and scaled. Organizations increasingly rely on cloud platforms such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) to build scalable and reliable systems.

Traditionally, cloud architecture design involves translating business requirements into infrastructure components, selecting appropriate cloud services, designing network topologies, and implementing infrastructure using Infrastructure-as-Code tools such as Terraform. This process is time-consuming and often requires collaboration between solution architects, DevOps engineers, and site reliability engineers.

Recent advancements in Large Language Models (LLMs) have opened opportunities for automating complex engineering workflows. Multi-agent frameworks allow multiple AI agents with specialized roles to collaborate in solving complex tasks.

This research introduces Cloud Infrastructure Crew, a multi-agent system that automates the cloud infrastructure lifecycle — from business requirements to deployment-ready infrastructure code. The system uses three specialized AI agents coordinated through the CrewAI framework.

II. RESEARCH AND IDEA

Existing studies on AI-assisted DevOps highlight the growing importance of automated infrastructure generation and intelligent deployment systems. Tools such as Terraform, AWS CloudFormation, and Pulumi provide Infrastructure-as-Code capabilities but still require manual design and configuration.

Key challenges identified include:

- Translating business requirements into technical architecture
- Generating production ready Terraform configurations
- Ensuring compliance, security, and cost efficiency
- Coordinating multiple infrastructure components.

To address these challenges, the study explored the use of autonomous AI agents that can simulate domain experts such as cloud architects, DevOps engineers, and site reliability engineers.

Through technical experimentation and evaluation of agent orchestration frameworks, CrewAI was selected as the orchestration platform for building the multi-agent system.

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715267

II. DESIGN AND FINDINGS

A. Architecture and Implementation

The proposed system is built using a modular technology stack designed to support autonomous agent orchestration, infrastructure generation, and deployment simulation. The agent orchestration layer is implemented using the CrewAI 0.86.0 framework, which enables the coordination of multiple specialized agents working in a sequential pipeline. For language intelligence and reasoning, the system primarily utilizes Azure OpenAI GPT-4o as the main Large Language Model (LLM) responsible for architecture design, infrastructure generation, and deployment analysis. To ensure reliability and continuity, the system also integrates a fallback mechanism that supports Anthropic Claude Sonnet and OpenAI GPT-4o, allowing the pipeline to continue operating even if the primary model becomes unavailable.

For infrastructure provisioning, the system generates configuration files using HashiCorp Terraform, a widely adopted Infrastructure-as-Code (IaC) language that allows declarative definition of cloud resources. To visualize cloud architecture components, the system leverages the Python diagrams library integrated with Graphviz, which enables automatic generation of infrastructure diagrams based on architecture specifications produced by the agents.

The command-line interaction layer of the system is built using the Typer CLI framework, which provides a structured interface for executing system commands and managing the workflow of agents. Additionally, the Rich terminal UI library is used to enhance command-line output by presenting structured logs, progress indicators, and formatted results, thereby improving usability and monitoring during pipeline execution. Together, this technology stack enables the development of a scalable and extensible multi-agent platform capable of automating complex cloud infrastructure design and deployment tasks.

B. Agents Plan and Design

Agent 1:Cloud Solutions Architect

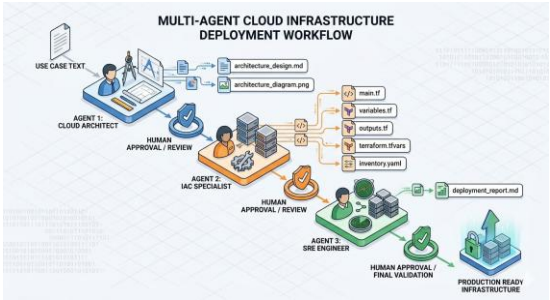
The first agent (architecture_agent) acts as a cloud solutions architect responsible for analyzing the business requirements and generating a detailed architecture design document along with a system architecture diagram with tech stack. This agent defines components such as networking, compute services, data storage, security layers, monitoring, and cost considerations. After producing the architecture document and diagram, the system pauses for a human approval step, allowing experts to review and validate the proposed architecture before the workflow proceeds to the next stage.

Agent 2: Infrastructure and IaC Specialist

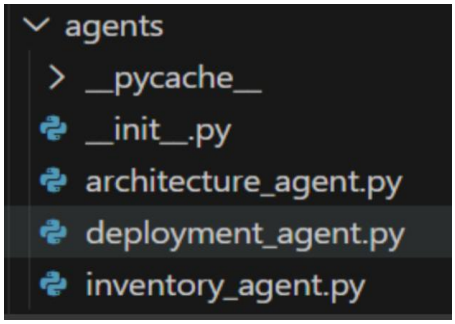
The second agent (inventory_agent) functions as an infrastructure and Infrastructure-as-Code specialist. It takes the approved architecture document as input and converts the design into Terraform configuration files and a structured infrastructure inventory. These files describe cloud resources, networking configuration, storage services, and deployment parameters required for the system. Once the Terraform files are generated, another human approval step is introduced so that engineers can review the generated infrastructure code and ensure correctness before moving forward.

Agent 3: Deployment and SRE Engineer

The third agent (deployment_agent) acts as a Site Reliability Engineer responsible for validating the generated infrastructure configuration and simulating the deployment process. It performs tasks such as pre-deployment checks, Terraform initialization, deployment planning, and post-deployment verification. The agent then produces a deployment report summarizing the results of the simulated deployment. After the report is generated, a final human approval step allows engineers to review the deployment results and confirm the readiness of the infrastructure before actual production deployment.



© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715267



IV. USE CASE

Here comes the most crucial step for your research publication. Ensure the drafted journal is critically reviewed by your peers or any subject matter experts. Always try to get maximum review comments even if you are well confident about your paper.

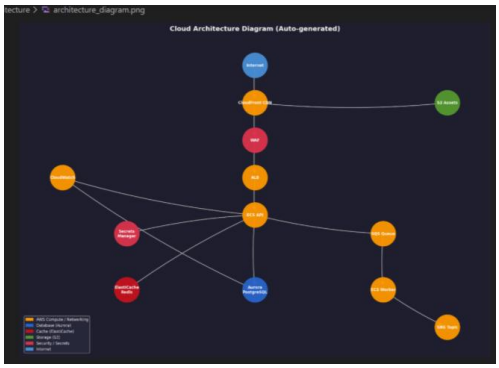
A). Business Requirement Specification

```
cloud_infra_crew > use_cases > ecommerce_platform.txt
USE CASE: Scalable E-Commerce Platform on AWS

BUSINESS CONTEXT:
A mid-size retail company "ShopNow" wants to migrate their monolithic on-premise
e-commerce application to the cloud. They expect 10,000 concurrent users on normal
days and 50,000+ during sales events (Black Friday / Cyber Monday).

FUNCTIONAL REQUIREMENTS:
- Product catalog with search & filtering (1M+ SKUs)
- Shopping cart and checkout with payment gateway integration (Stripe)
- User authentication and profile management
- Order management and tracking
- Real-time inventory updates
- Email/SMS notification system
- Admin dashboard for analytics
- CDN-served static assets (images, CSS, JS)

NON-FUNCTIONAL REQUIREMENTS:
- 99.9% uptime SLA
- Auto-scaling: handle 5x traffic spikes without manual intervention
- < 200ms API response time (p95)
- Data encryption at rest and in transit
- PCI-DSS compliance for payment data
- GDPR compliance (EU customers)
- Disaster recovery: RTO < 4h, RPO < 1h
- Daily automated backups retained for 30 days
- CI/CD pipeline for zero-downtime deployments

BUDGET CONSTRAINT:
- ~$3,000-$5,000/month estimated cloud spend
- Cost optimization is important; use reserved/spot instances where safe

PREFERRED CLOUD:
AWS (primary), with multi-region active-passive failover (us-east-1 primary, eu-west-1 DR)
```

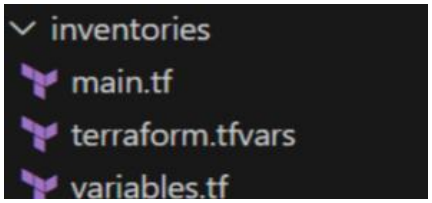
B). Crew Construction and Agent Initialization

```python
def build_crew(use_case: str) -> Crew:
    """Full 3-agent pipeline: design → inventory → deployment."""

    llm = get_llm()

    # Agents
    architect = create_architecture_agent(llm)
    inventory_mgr = create_inventory_agent(llm)
    deployer = create_deployment_agent(llm)

    # (sequential; Task 2 & 3 receive context from predecessors)
    task_arch = create_architecture_task(architect, use_case)
    task_inv = create_inventory_task(inventory_mgr, task_arch)
    task_dep = create_deployment_task(deployer, task_inv)

    # Crew
    return Crew(
        agents=[architect, inventory_mgr, deployer],
        tasks=[task_arch, task_inv, task_dep],
        process=Process.sequential,
        verbose=True,
        memory=False,
        output_log_file="outputs/crew_run.log",
    )
```

C). Output Generated by Agent 1 – Cloud Architecture Design

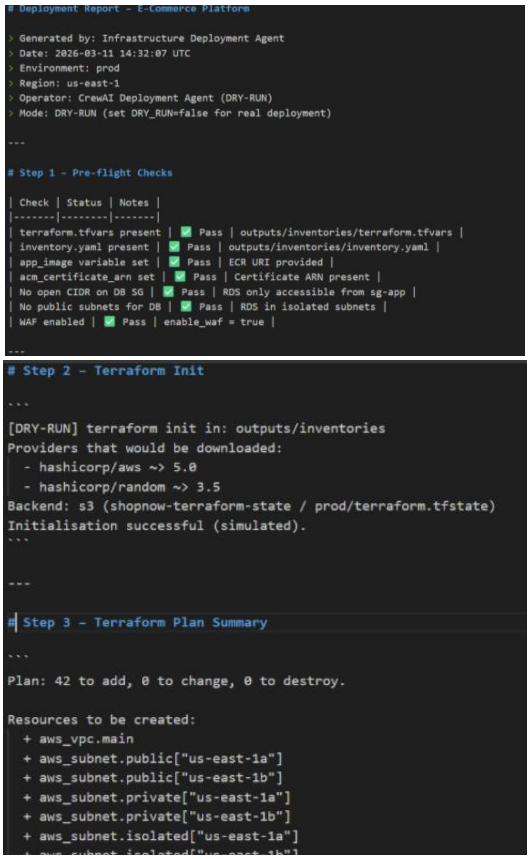


D). Output Generated by Agent 2 – Infrastructure as Code Generation



E). Output Generated by Agent 3 – Deployment Validation and Report

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715267



```
# Deployment Report – E-Commerce Platform

> Generated by: Infrastructure Deployment Agent
> Date: 2026-03-11 14:32:07 UTC
> Environment: prod
> Region: us-east-1
> Operator: CrewAI Deployment Agent (DRY-RUN)
> Mode: DRY-RUN (set DRY_RUN=false for real deployment)

---

# Step 1 – Pre-flight Checks

| Check | Status | Notes |
|-------|--------|-------|
| terraform.tfvars present | ✅ Pass | outputs/inventories/terraform.tfvars |
| inventory.yaml present | ✅ Pass | outputs/inventories/inventory.yaml |
| app_image variable set | ✅ Pass | ECR URI provided |
| acm_certificate_arn set | ✅ Pass | Certificate ARN present |
| No open CIDR on DB SG | ✅ Pass | RDS only accessible from sg-app |
| No public subnets for DB | ✅ Pass | RDS in isolated subnets |
| WAF enabled | ✅ Pass | enable_waf = true |

---
```



```
# Step 2 – Terraform Init

```
[DRY-RUN] terraform init in: outputs/inventories
Providers that would be downloaded:
  - hashicorp/aws ~> 5.0
  - hashicorp/random ~> 3.5
Backend: s3 (shopnow-terraform-state / prod/terraform.tfstate)
Initialisation successful (simulated).
```

Step 3 – Terraform Plan Summary

```
Plan: 42 to add, 0 to change, 0 to destroy.

Resources to be created:
  + aws_vpc.main
  + aws_subnet.public["us-east-1a"]
  + aws_subnet.public["us-east-1b"]
  + aws_subnet.private["us-east-1a"]
  + aws_subnet.private["us-east-1b"]
  + aws_subnet.isolated["us-east-1a"]
```

V. CONCLUSION

This paper demonstrates that LLM-based multi-agent systems can effectively automate the cloud infrastructure lifecycle. The combination of specialized agents, human approval gates, dual-path diagram generation, and dry-run simulation makes it practical for both exploration and production use.

The modular tool and agent design makes the system extensible to additional cloud providers, IaC frameworks (Pulumi, CDK), and compliance frameworks.

REFERENCES

[1] Khan, R. N. H., Wasif, D., Cho, J. H., & Butt, A. (2025).“Multi-Agent Code-Orchestrated Generation for Reliable Infrastructure-as-Code.”arXiv.

[2] Kolli, B. P. (2025).“The Rise of AI-Augmented DevOps: How Human Engineers and AI Co-manage Cloud Infrastructure.” World Journal of
```