

Unlocking 10x Productivity: A Blueprint for Distributed Multi- Agentic AI Platform Engineering.

How Outshift by Cisco leveraged Community AI Platform Engineering (CAIPE) and AGNTCY components to orchestrate a Distributed Multi-agents System (D-MAS) to accelerate developer workflows, troubleshoot complex platform engineering issues, and deliver 10× engineering productivity.

Sri Aradhyula

AI Agentic Platform Engineering Architect, Outshift by Cisco

Sunny Whang

AI Agentic Platform Engineering Lead, Outshift by Cisco

Hasith Kalpage

Director of Platform Engineering & CISO, Outshift by Cisco

Arthur Drozdov

AI Agentic Platform Engineer, Outshift by Cisco

Shubham Bakshi

AI Agentic Platform Engineering Lead, Outshift by Cisco

Stefan Braicu

AI Agentic Platform Engineering Lead, Outshift by Cisco

Contents

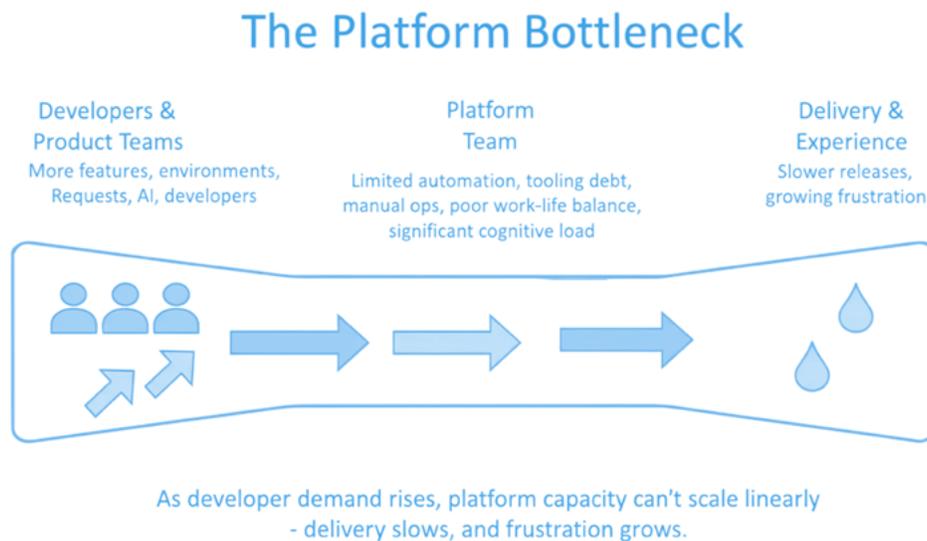
1 Executive summary	5
2 What is CAIPE?	5
3 Agentic AI basic definitions	6
3.2 What is an AI agent?	6
3.4 ReAct loop	7
3.5 What is a multi-agent system (MAS)	8
3.6 What is a distributed multi-agent system (D-MAS)	8
3.8 Agent2Agent (A2A) Protocol	8
3.9 Open Agentic Schema Framework (OASF)	9
3.10 Secure Low-Latency Interactive Messaging (SLIM)	9
3.11 AGNTCY Identity	9
3.11.1 AGNTCY Observability and evaluation	9
4 Common MAS architecture patterns	9
4.1 Supervisor agents	9
4.2 Deep agents / planner-based multi-agent systems	10
4.3 Conversational distributed multi-agent systems	11
5 Agentic AI and distributed MAS adoption challenges and mitigations	11
5.1 Deterministic workflows:	11
5.2 Spec conformance:	12
5.3 Agent and tool grounding	12
5.4 Artifact consistency	12
5.5 Human-in-the-loop (HITL) gating	12
5.6 Cost and latency management	12
6 Architectural considerations	12
6.1 Single agent vs. multi-agent architecture	12
6.2 Distributed orchestration	13
6.3 Agent-to-agent vs. agent group conversation	13

6.4 Context engineering	13
7 CAIPE architecture	14
8 How is CAIPE used in Outshift production	15
8.1 Use case 1: Knowledge base retrieval	15
8.1.1 Problem	15
8.1.2 Implementation	15
8.1.3 Outcome	16
8.2 Use case 2a: Developer SELF-SERVICE (GOLDEN PATH) – LLM KEY Creation	19
8.2.1 Problem	19
8.2.2 Implementation	19
8.2.3 Outcome	20
8.3 Use case 2b: Deploy an application using ArgoCD	20
8.3.1 Problem	20
8.3.2 Implementation	21
8.3.3 Outcome	21
8.4 Use case 3: Troubleshooting platform issues	23
8.4.1 Problem	23
8.4.2 Implementation	24
8.4.3 Outcome	24
8.5 Use case 4: Service desk management	25
8.5.1 Problem	25
8.5.2 Implementation	25
8.5.3 Outcome	25
8.6 Use case 5: Incident management	26
8.6.1 Problem	26
8.6.2 Implementation	26
8.6.3 Outcome	27
9 Impact at Outshift	28
10 How is CAIPE evaluated?	28

10.1 Tracing foundation	28
10.2 Evaluation method (offline and online)	29
10.3 Evaluators, metrics and rubrics	29
12 Lessons learned	31
12.1 Distributed multi-agent orchestration is complex	31
12.2 Version everything	32
12.3 Context engineering and deep agents	32
12.4 Plan for detailed agent evaluation	32
12.5 LLMs are not static	32
12.6 Treat the agent UI as a first-class citizen	32
12.7 Prefer standard protocol-compatible agents	32
12.8 Propagate user context identity	32
13 CAIPE roadmap	32
14 Get involved in CAIPE open source community	33
15 References	34

1 Executive summary

Platform Engineering, Site Reliability Engineering (SRE), and DevOps teams today operate across fragmented environments that include multiple clouds, services, clusters, pipelines, and security and compliance frameworks. As environments grow in complexity, traditional approaches to developer experience like tickets, static runbooks, disjointed portals, and siloed bots are brittle and often lead to delays, increased operational overhead, and developer frustration.



By adopting agentic AI and distributed multi-agent systems (D-MAS), platform engineering teams can move from semi-manual, task-driven operations to adaptive, automated workflows that better support development and business goals. This includes scalable long-running troubleshooting to identify root causes and self-service workflow automation without major system changes.

The platform engineering team at Outshift, Cisco's incubation engine, achieved this goal by co-creating and leveraging [Community AI Platform Engineering \(CAIPE\)](#) an open source and distributed multi-agent system.

2 What is CAIPE?

CAIPE is a community-driven, open source, distributed multi-agent system that provides a customizable, ready-to-deploy AI platform engineer. CAIPE's vision is to build a distributed multi-agent AI platform engineering system that can be tailored to any platform team's needs, with the goal of enabling:

- Agents that understand platform context
- Agents that collaborate like a high-performing SRE or DevOps team
- An architecture that is secure, low-latency, observable, and standards-aligned
- A design that any organization can adopt incrementally

CAIPE applies agentic AI and distributed multi-agent systems (D-MAS) to platform engineering by:

- Building a platform engineering knowledge graph and retrieving relevant contextual information
- Ability to troubleshoot complex incidents
- Enabling policy and security-aware self-service automation
- Performing cross-tool correlation, analysis, and report generation

CAIPE is built on open standards, protocols, and popular community agentic SDKs and tools:

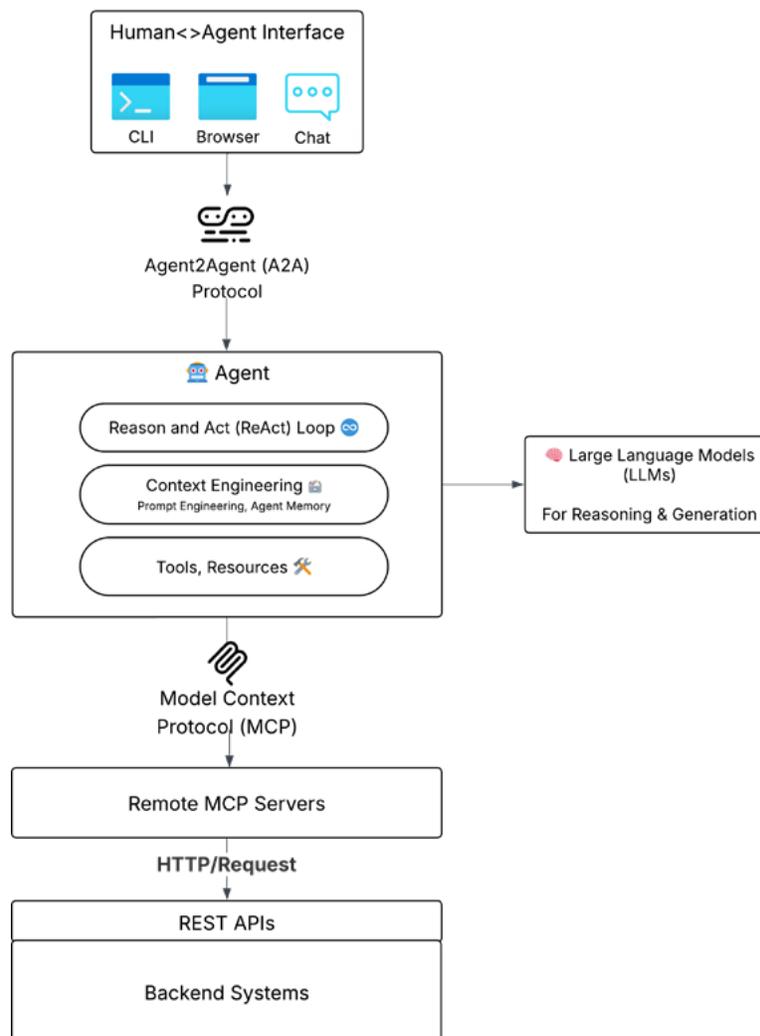
- Model Context Protocol ([MCP](#)) for standardized, schema-driven tool integration
- Agent2Agent ([A2A](#)) Protocol for interoperable agent-to-agent communication
- Open Agent Schema Framework ([OASF](#)) to represent agent artifacts such as prompts, tools, skills, evaluations, and large language model (LLM) parameters
- AGNTCY Secure Low-Latency Interactive Messaging (SLIM) for group and conversational patterns
- OAuth and [AGNTCY Identity](#) for user-context identity and verifiable agent-to-agent identity
- OpenTelemetry (OTel), [AGNTCY Observability and evaluation](#), and community evaluation tools.
- LangGraph, Strands, and other agentic SDKs are used across multiple agents built in CAIPE

3 Agentic AI basic definitions

3.2 WHAT IS AN AI AGENT?

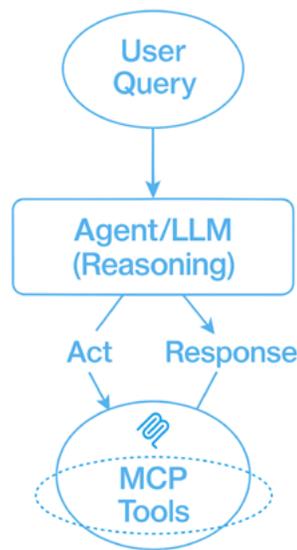
An AI agent is a system that uses a large language model (LLM) to reason on a natural language question and determine the flow of an application based on:

- System prompts or instructions given to the LLM
- Tools and resources it can access
- Agent contextual memory (short-term and long-term)



3.4 REACT LOOP

[Reason and Act \(ReAct\)](#) is a common design pattern used in agentic systems to help LLMs decide the next action or tool to use and perform actions in a loop until a desired result is achieved.



3.5 WHAT IS A MULTI-AGENT SYSTEM (MAS)

A multi-agent system (MAS) is an agentic AI system composed of multiple, independent, and interacting agents that collaborate to achieve a shared objective. Each agent typically is designed to have specialized capabilities such as domain specific system prompts, knowledge bases, and tools to reason and retrieve information.

3.6 WHAT IS A DISTRIBUTED MULTI-AGENT SYSTEM (D-MAS)

A distributed multi-agent system (D-MAS) extends the MAS paradigm across network boundaries. Agents are deployed on different nodes, services, or regions and communicate over secure, low-latency channels. This distribution introduces new design considerations around orchestration, trust boundaries, network reliability, observability, and identity management.

3.7 MODEL CONTEXT PROTOCOL (MCP)

[MCP](#) standardizes how LLMs and agents discover and call APIs and data sources. It defines a consistent interface for exposing capabilities like tools, resources, and prompts.

3.8 AGENT2AGENT (A2A) PROTOCOL

[A2A](#) is an open protocol for agent-to-agent communication and interoperability. It defines common message formats and patterns so agents from different vendors, runtimes, or stacks can talk to each other, advertise capabilities, and collaborate across networks and organizations.

3.9 OPEN AGENTIC SCHEMA FRAMEWORK (OASF)

[OASF](#) provides a standard schema for defining agents—their identity, capabilities, tools, protocols, and configuration—as versioned artifacts. This makes agents discoverable, auditable, and versionable, and fits nicely alongside MCP and A2A in multi-agent ecosystems like CAIPE.

3.10 SECURE LOW-LATENCY INTERACTIVE MESSAGING (SLIM)

[SLIM](#) is a secure, low-latency messaging fabric for agents. It supports unicast, group, and pub/sub communication with built-in routing, retries, and session management. This enables conversational agent-to-agent and agent-group workflows without custom transport code.

3.11 AGNTCY IDENTITY

[AGNTCY Identity](#) defines how agents and services are uniquely identified, authenticated, and authorized. It underpins least-privilege access, scoped credentials, and verifiable “who did what,” forming the trust layer beneath agents and MCP servers.

3.11.1 AGNTCY OBSERVABILITY AND EVALUATION

AGNTCY Observability defines how to trace and measure agent behavior, messages, and tool calls end-to-end. In a MAS, many distributed agents interact in complex ways, so observability makes information flow and decision paths visible by recording what each agent did and what happened. On top of this factual view, evaluation assesses the quality of those behaviors, helping teams trust and improve large-scale, distributed multi-agent systems.

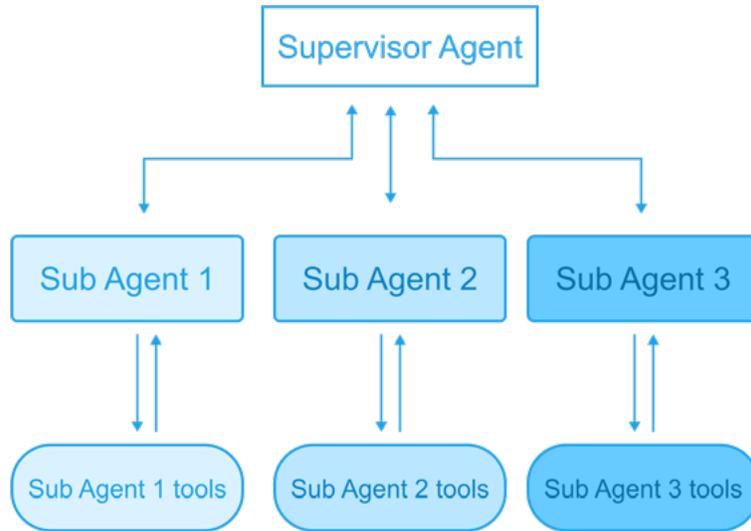
4 Common MAS architecture patterns

Distributed multi-agent systems (D-MAS) can be composed in several ways, each balancing simplicity, control, scalability, and collaboration. Rather than a single “perfect” design, real-world platforms blend patterns: supervisor agents that coordinate specialist sub-agents, planner-based [“deep” agents](#) that manage complex workflows and networked or swarm-style agents that collaborate via shared conversations or topics. The following sections outline common MAS and D-MAS architecture patterns that CAIPE builds on, and how these architecture patterns support distributed and large-scale automation in modern platform engineering.

4.1 SUPERVISOR AGENTS

A supervisor agent coordinates a set of specialized sub-agents within the same system or across the network. It decomposes high-level goals into tasks, assigns them to the appropriate agents, tracks progress, and aggregates results. This pattern centralizes control while

allowing each sub-agent to remain focused and independently evolvable.



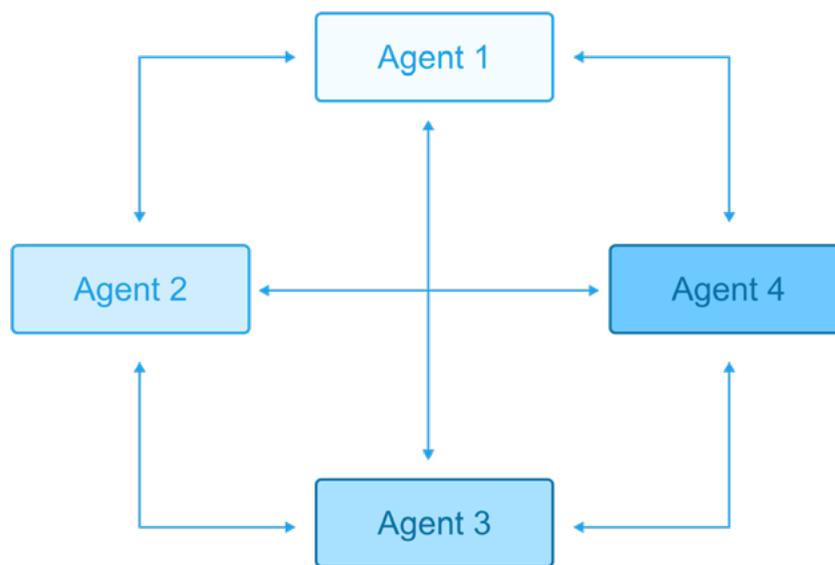
4.2 DEEP AGENTS / PLANNER-BASED MULTI-AGENT SYSTEMS

Simple ReAct agents are effective for short, bounded tasks but tend to become “shallow” and ineffective on longer-running, multi-step, or multi-turn workflows. [Planner-based, or “deep agents”](#) introduce an explicit planning layer that constructs an execution plan, invokes sub-agents and system tools in sequence or in parallel, and incorporates human-in-the-loop (HITL) checkpoints where needed.



4.3 CONVERSATIONAL DISTRIBUTED MULTI-AGENT SYSTEMS

In conversational MAS and D-MAS architectures, agents communicate over shared channels using patterns such as pub-sub, multicast, or broadcast. Each agent participates in a conversation, like a group chat, sharing findings, proposing new actions for other agents to perform, and converging on solutions to complex problems collaboratively. In practice, all the architectural patterns discussed above are fluid and are often combined together to enable coordinated, trustworthy problem solving within a defined agent trust boundary.



5 Agentic AI and distributed MAS adoption challenges and mitigations

Adopting agentic AI and distributed multi-agent systems (D-MAS) in platform engineering raises challenges that span both LLM reasoning behavior and distributed systems.

Here are key challenges platform engineering teams face when adopting agentic AI:

5.1 DETERMINISTIC WORKFLOWS:

- Platform workflows require predictable, auditable outcomes, while LLMs are probabilistic.
- One of the techniques use planner-based directives in MAS/D-MAS system prompts to constrain behavior and generate an explicit agent execution plan.
- Have agents follow that plan as explicit, deterministic tasks with scoped tools and parameters.
- Ensure multi-step instructions are repeatable through thorough agentic evaluations.

5.2 SPEC CONFORMANCE:

- Treat all LLM-generated code and configuration as proposals—not ground truth to immediately apply or modify infrastructure.
- Validate configuration against authoritative contracts like OpenAPI, JSON schema, Kubernetes CRDs, policy definitions.
- Iterate with LLM until results are fully spec-conformant and are valid before applying or merging the changes.

5.3 AGENT AND TOOL GROUNDING

- Ensure multi agent systems reliably select the correct sub-agents and tools, and tool parameters.
- Trace tool calls end-to-end so every decision and action is visible, auditable, and attributable.

5.4 ARTIFACT CONSISTENCY

- Version agents, MCP tools, prompts, LLM parameters and evaluations results as first-class artifacts
- Use schema-driven specifications like [AGNTCY OASF](#) to describe capabilities and contracts for defining agent protocols (MCP, A2A) and artifacts (prompts, LLM params, eval results).
- Capture LLM model version, parameters, and evaluation results as an artifact to benchmark and track any regression.

5.5 HUMAN-IN-THE-LOOP (HITL) GATING

- Define when agents must pause for human approval based on risk tier, blast radius, sensitivity, and environment.
- Use reversible mechanisms (e.g., Git-based pull requests, progressive rollouts) for all high-impact changes.

5.6 COST AND LATENCY MANAGEMENT

- Set token, call, and time budgets per agent and workflow.
- Limit ReAct recursion depth, especially for long-running tasks.
- Batch and cache repetitive calls; route simple tasks to smaller, faster models.

6 Architectural considerations

6.1 SINGLE AGENT VS. MULTI-AGENT ARCHITECTURE

Single-agent designs centralize reasoning and tools into one large agent. They are simple to start with but:

- Suffer from context bloat and overlapping responsibilities between tools.



- Do not map cleanly to team or domain ownership at scale.

Multi-agent architectures, as used in CAIPE:

- Enforce context isolation by assigning each agent a clear domain or service.
- Allow targeted policies, credentials, and guardrails per agent.
- Scale across organizations: Teams can own and evolve their agents independently while still collaborating through shared protocols.

6.2 DISTRIBUTED ORCHESTRATION

Distributed multi-agent systems are designed for fan-out/fan-in, parallel execution, and long-running tasks. Robust orchestration includes:

- Coordinating dependent and parallel steps across agents and services.
- Implementing sub-agent retries, exponential backoff, heartbeats or health checks, and timeouts to detect failures continuously to prevent agent communication stalls.
- Ensuring idempotency and well-defined compensation so that partial failures, duplicates, and restarts do not corrupt supervisor agent state.
- Providing end-to-end traceability across the entire execution graph for audit, debugging, and post-failure learning.

6.3 AGENT-TO-AGENT VS. AGENT GROUP CONVERSATION

In a distributed orchestration, agents can communicate both directly with each other or in collaborative groups.

Agent-to-Agent (Peer-to-Peer)

- Best for focused, low-complexity exchanges between two agents.
- Well-suited for smaller MAS deployments where a supervisor agent coordinates interactions.

Agent Group Conversations (via SLIM)

- Provide secure, low-latency channels where multiple agents share a common context (e.g., an incident or rollout).
- Enable swarm-style collaboration: specialists join, contribute data and tools, analyze options, and converge on actions.
- Support dynamic membership and long-running threads, essential for complex investigations.

6.4 CONTEXT ENGINEERING

Effective agentic systems are defined by how they manage context and provide context to the LLM, not just how they call tools:

- Maintain a conversational history as a short-term memory.
- Continuously prune and summarize short-term memory to fit within context window .
- Compress important facts and store as long-term memory.

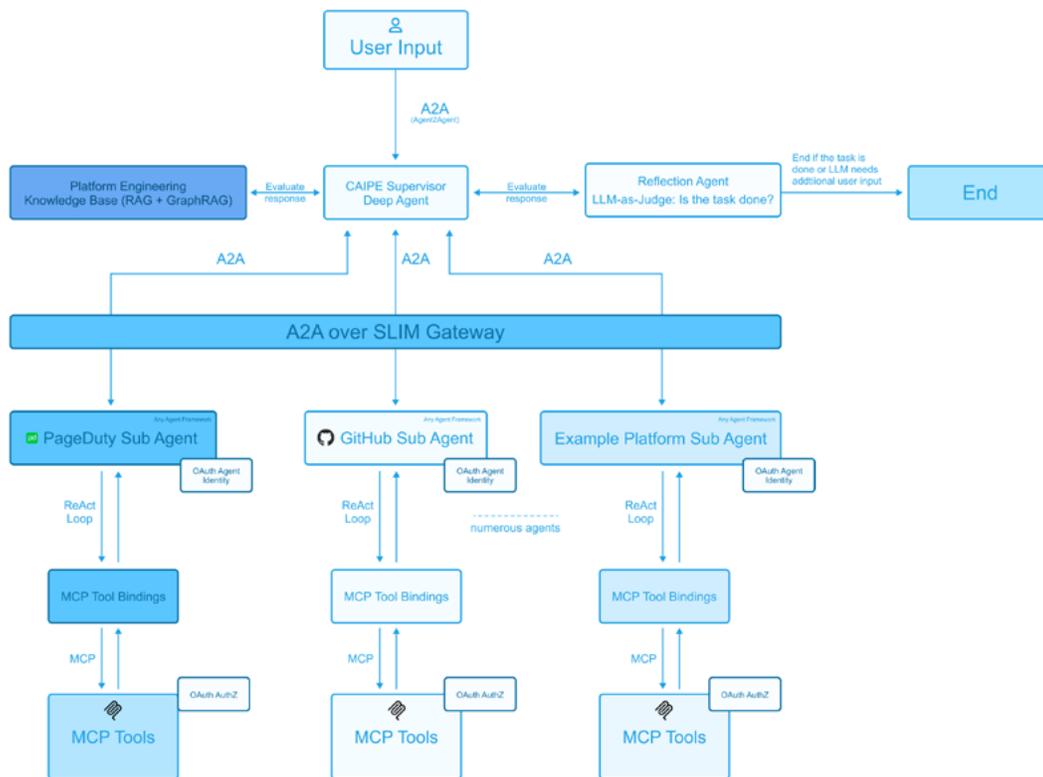


7 CAIPE architecture

At its core, the CAIPE supervisor agent is a deep agent. A deep agent is an AI agent built to handle long-running, multi-step work by planning, coordinating multiple tools/sub-agents, and verifying results before reporting back.

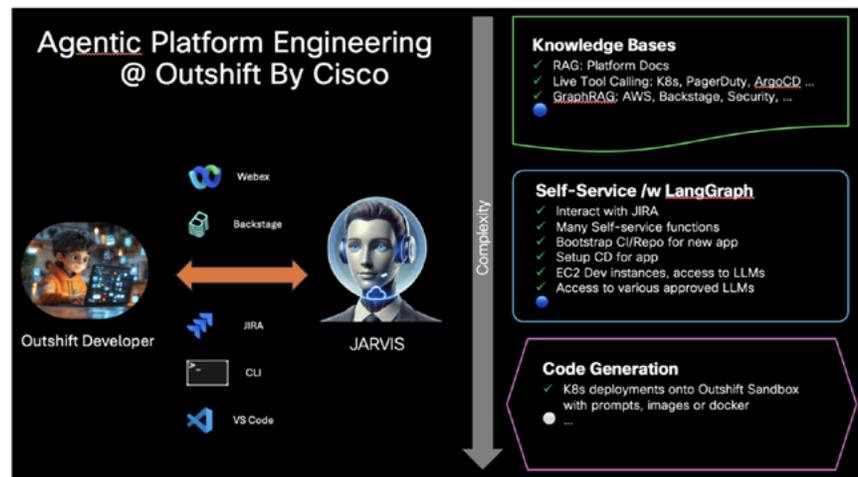
A deep agent architecture goes beyond a single ReAct step or one-shot tool call. This makes CAIPE especially effective for long-running investigations that coordinate multiple tools and sub-agents when troubleshooting platform engineering and SRE issues.

CAIPE streamlines self-service, troubleshooting, and incident management—delivering up to 10x productivity on common tasks—and its extensible architecture accelerates automation rollouts and improves outcomes (shorter lead time, lower Mean Time To Recovery (MTTR), higher first-contact resolution).



CAIPE Architecture

8 How is CAIPE used in Outshift production



Outshift's platform engineering team has deployed CAIPE and integrated it with their [Backstage Internal Developer Portal](#) using the open-source [agent-forge plugin](#) that was also created for this project. This provided the team with a secure, familiar environment to access CAIPE's robust multi-agent capabilities.

The following sections highlight real-world Outshift use cases that demonstrate how CAIPE is actively used today:

8.1 USE CASE 1: KNOWLEDGE BASE RETRIEVAL

8.1.1 PROBLEM

Platform and SRE teams operate across fragmented knowledge sources including documentation systems, source repositories, tickets, incident records, and internal communications. Information is often outdated, duplicated, or difficult to correlate, leading to increased time spent searching for context and a higher risk of operational errors. One of the primary use cases of CAIPE at Outshift is retrieving information that's spread across multiple document stores and services. CAIPE uses Retrieval-Augmented Generation (RAG) and GraphRAG to build a map of structured and unstructured data and then performs a hybrid search to retrieve and summarize information—providing quick, relevant, and recent insights.

8.1.2 IMPLEMENTATION

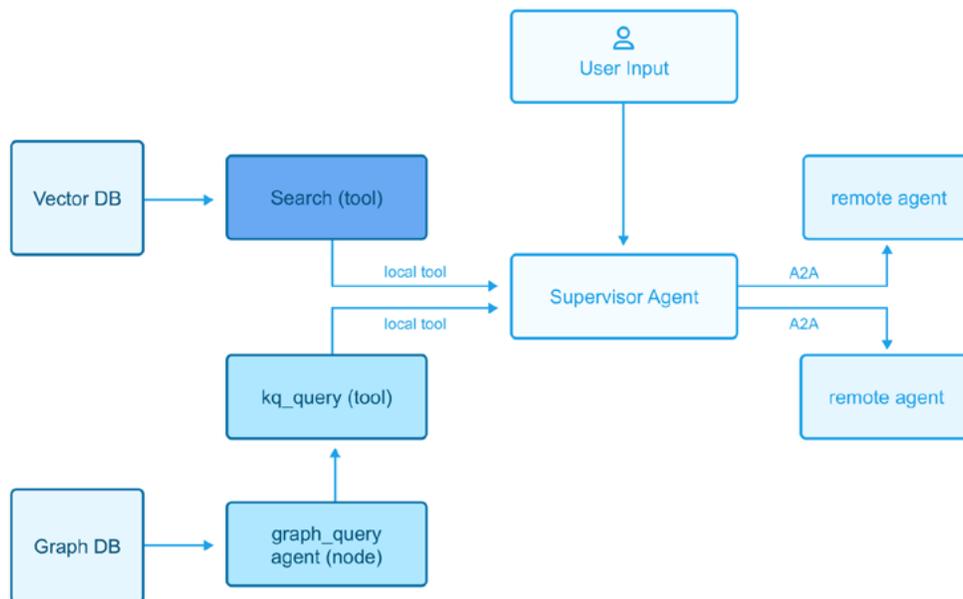
CAIPE implements a unified knowledge retrieval layer using Retrieval-Augmented Generation (RAG) and GraphRAG.

- Unstructured and structured data is continuously ingested from multiple systems.

- Vector-based semantic search is combined with keyword search for precision.
- GraphRAG models relationships between services, environments, incidents, ownership, and artifacts.
- Queries traverse both vector and graph layers to retrieve, correlate, and summarize relevant information.
- Results are grounded in source material and prioritize recent and authoritative data.

8.1.3 OUTCOME

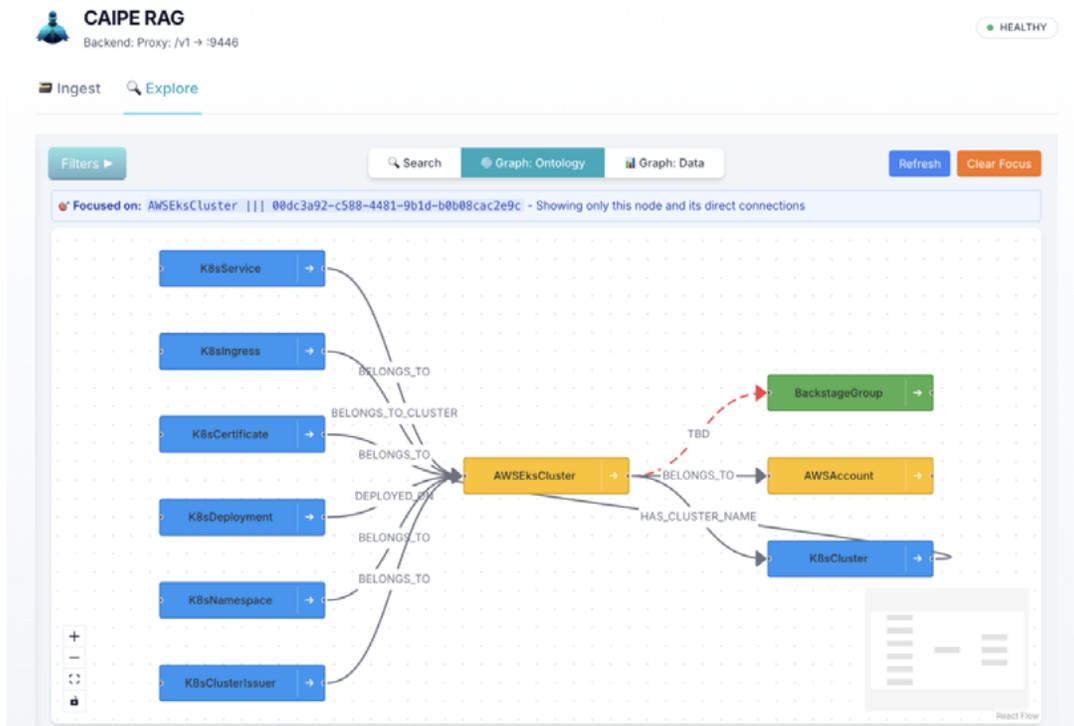
Engineers can retrieve accurate, contextual answers without manually searching multiple services. Operational knowledge becomes searchable and retrievable quickly, reducing time to provide valuable context in complex troubleshooting scenarios and platform architectural decisions.



CAIPE knowledge ingestion and retrieval architecture

The screenshot shows the CAIPE RAG web interface. At the top, it says "CAIPE RAG" with a "Backend: Proxy: /v1 -> :9446" and a "HEALTHY" status indicator. Below are "Ingest" and "Explore" tabs. The "Ingest URL" section has a text input field containing "https://docs.example.com" and an "Ingest" button. A checkbox for "Check for sitemap" is checked. An "Optional Configuration" dropdown is visible. The "Data Sources" section features a table with columns: PATH, DOCUMENTS, LAST UPDATED, and ACTIONS. The table lists a source for "https://cnoe-io.github.io/ai-platform-engineering/" with a document count of 5 and a last update time of 11/11/2025, 10:54:47 PM. Below the table, detailed ingestion statistics are shown, including ID, Default Chunk Size (10000), Created time, Documents Processed (5), Ingestion Status (4% progress bar), and Job ID. A "Stop" button is present next to the progress bar.

CAIPE unstructured data ingestion web UI



Visualizing CAIPE GraphRAG knowledge base relationship summary

 You
How can I publish agent record using dirctl?
08:11 PM

 CAIPE
Response

To publish an agent record using `dirctl`, follow these steps:

1. **Prepare the Agent Record:** Ensure that your agent record is in JSON format and conforms to the Outshift Agent Directory requirements. The record should include necessary metadata such as the agent's name, version, description, locators, skills, and any previous record CID if applicable.
2. **Install `dirctl`:** Make sure you have the `dirctl` command-line tool installed. You can find installation options on GitHub, including container images, Homebrew, and binary packages.
3. **Authenticate to the Outshift Agent Directory:** You can authenticate using either interactive login or API key-based login.
 - **Interactive Login:** Run `dirctl hub login` to open a login page in your browser where you can enter your credentials.
 - **API Key Login:** Generate API key credentials using `dirctl hub apikey create` and store them as environment variables or in a JSON file.
4. **Push the Agent Record:** Use the following command to push your agent record to the desired organization:


```
dirctl hub push <organization> <record.json>
```

If using API key authentication, include the `--apikey-file` option with your credentials file:

```
dirctl hub push <organization> <record.json> --apikey-file api-key-creds.json
```
5. **Sign the Agent Record:** After pushing the record, sign it using the default provider Sigstore:


```
dirctl hub sign <organization> <record CID>
```

The CID of the record can be found in the output of the push command or in the record details page in the Hub UI.

Retrieving knowledge base questions using CAIPE

8.2 USE CASE 2A: DEVELOPER SELF-SERVICE (GOLDEN PATH) – LLM KEY CREATION

8.2.1 PROBLEM

Developers require access to multiple LLM providers to build and test AI-powered services, but obtaining API keys is often a manual, ticket-driven process that lacks agility, clear budget standards and governance. This results in delayed access, inconsistent usage, and increased security and compliance risk.

8.2.2 IMPLEMENTATION

CAIPE enables developer self-service by encoding platform “golden paths” as agent-driven workflows coordinated by a supervisor agent. In this workflow, CAIPE provides a golden path for LLM key access through an agent-orchestrated workflow.

The supervisor agent validates approved providers, models, and usage budget policies. The user is validated against the project in Backstage. The agentic workflow provisions a scoped

LLM key and is securely transmitted to the user.

8.2.3 OUTCOME

Developers receive fast, standardized access to approved LLM services without bypassing security controls. Platform teams maintain visibility, auditability, and policy enforcement while eliminating manual approval workflows and unsafe credential sharing to multiple LLM providers.

The screenshot shows a chat window with a user 'You Jarvis' and a bot 'CAIPE'. The bot's response asks for LLM provider, model, and project name details. Below the response is an 'Execution Plan' section and a form titled 'Input Required'. The form contains three dropdown menus: 'Provider Name' (selected: mistral), 'Model' (selected: amazon.titan-text-express-v1), and 'Project Name' (selected: action-engine). Each dropdown has a 'Required' label. A 'SUBMIT' button is at the bottom of the form.

You Jarvis
Get LLM Access
08:09 PM

CAIPE
Response

To request LLM access, I need to know the specific LLM provider, model, and project name you are interested in. Please provide these details so I can proceed with your request.

Execution Plan

08:09 PM

Input Required

To request LLM access, I need to know the specific LLM provider, model, and project name you are interested in. Please provide these details so I can proceed with your request.

Provider Name
mistral **Required**
The name of the LLM provider.

Model
amazon.titan-text-express-v1 **Required**
The name of the LLM model.

Project Name
action-engine **Required**
The name of the project that the user is part of.

SUBMIT

8.3 USE CASE 2B: DEPLOY AN APPLICATION USING ARGOCD

8.3.1 PROBLEM

As platform environments grow in complexity, deploying a new application or microservice consistently requires deep knowledge of repositories, CI/CD pipelines, Kubernetes configuration, security controls, and organizational standards. This knowledge often resides with a small set of platform experts, creating bottlenecks, inconsistent deployments, and increased

onboarding time for developers.

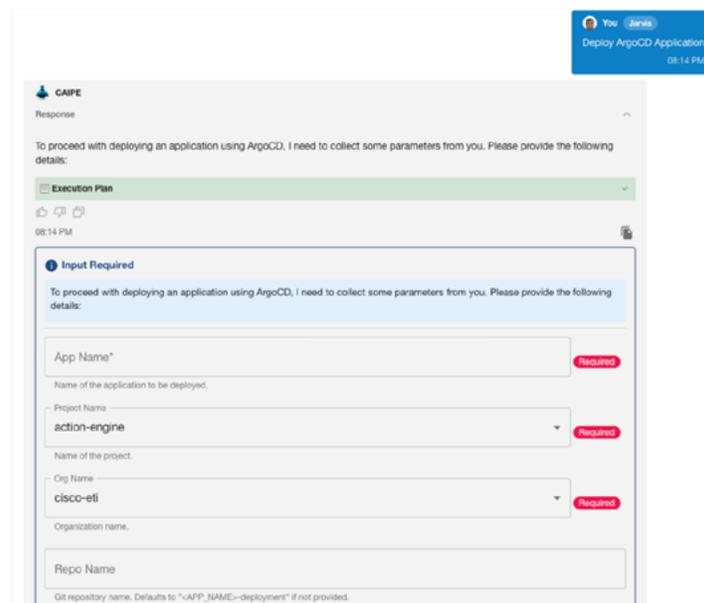
8.3.2 IMPLEMENTATION

CAIPE enables developer self-service by encoding platform “golden paths” as agent-driven workflows coordinated by a supervisor agent.

- The user interacts with CAIPE through a platform engineer or developer persona.
- The supervisor agent orchestrates multiple specialized agents:
 - **GitHub agent** for repository creation, configuration, and pull requests
 - **ArgoCD agent** for application and deployment configuration
 - **Jarvis agent** for enforcing Outshift enterprise standards and templates
 - **Kubernetes agent** for cluster and environment validation
- Each step follows predefined templates and policies, ensuring consistency and compliance.
- Human-in-the-loop approvals are introduced at key decision points before changes are applied.

8.3.3 OUTCOME

Developers can deploy new services independently while adhering to organizational standards. Platform teams reduce manual intervention, environments remain consistent, and onboarding time is significantly reduced without sacrificing governance or control.



The screenshot displays a chat interface for CAIPE. At the top right, a blue notification bubble says "You Jarvis Deploy ArgoCD Application 08:14 PM". The main chat area shows a "Response" from CAIPE: "To proceed with deploying an application using ArgoCD, I need to collect some parameters from you. Please provide the following details:". Below this is a green "Execution Plan" header. A blue "Input Required" box contains the following form fields:

- App Name*** (text input, required): "Name of the application to be deployed."
- Project Name** (dropdown menu, required): "action-engine", "Name of the project."
- Org Name** (dropdown menu, required): "cisco-etl", "Organization name."
- Repo Name** (text input): "Git repository name. Defaults to 'APP_NAME--deployment' if not provided."

App Namespace
Kubernetes namespace for deployment. Defaults to "<APP_NAME>-<ENVIRONMENT>" if not provided.

App Helm Chart Version
Helm chart version for the application.

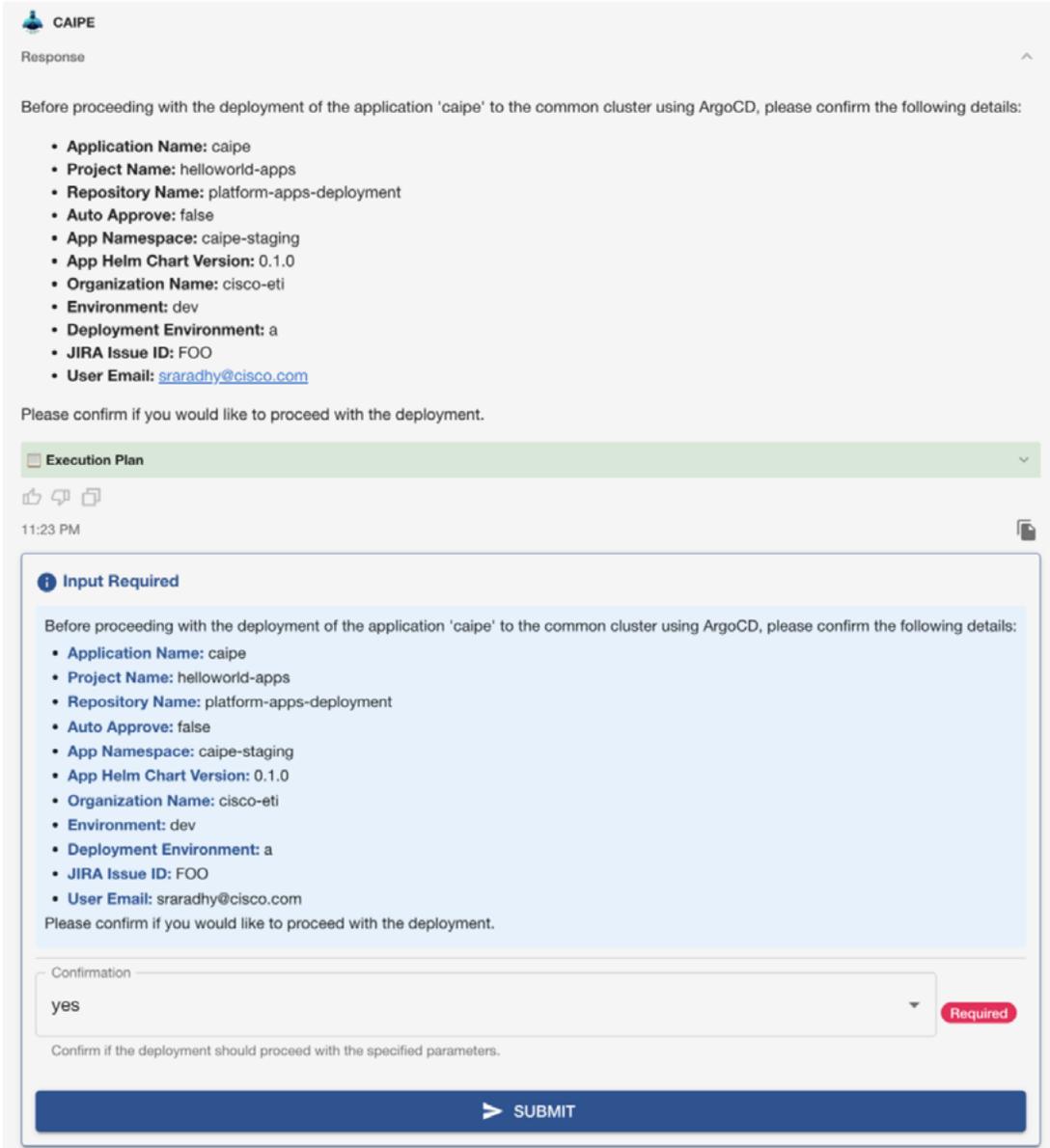
Environment
Deployment environment.
dev

Deployment Env
Deployment variant.
a

Auto Approve
Whether to auto-merge to git.
true

Jira Issue Id
The JIRA issue ID for the request.

> SUBMIT



CAIPE

Response

Before proceeding with the deployment of the application 'caipe' to the common cluster using ArgoCD, please confirm the following details:

- **Application Name:** caipe
- **Project Name:** helloworld-apps
- **Repository Name:** platform-apps-deployment
- **Auto Approve:** false
- **App Namespace:** caipe-staging
- **App Helm Chart Version:** 0.1.0
- **Organization Name:** cisco-eti
- **Environment:** dev
- **Deployment Environment:** a
- **JIRA Issue ID:** FOO
- **User Email:** sraradhy@cisco.com

Please confirm if you would like to proceed with the deployment.

Execution Plan

11:23 PM

Input Required

Before proceeding with the deployment of the application 'caipe' to the common cluster using ArgoCD, please confirm the following details:

- **Application Name:** caipe
- **Project Name:** helloworld-apps
- **Repository Name:** platform-apps-deployment
- **Auto Approve:** false
- **App Namespace:** caipe-staging
- **App Helm Chart Version:** 0.1.0
- **Organization Name:** cisco-eti
- **Environment:** dev
- **Deployment Environment:** a
- **JIRA Issue ID:** FOO
- **User Email:** sraradhy@cisco.com

Please confirm if you would like to proceed with the deployment.

Confirmation

yes

Required

Confirm if the deployment should proceed with the specified parameters.

SUBMIT

Human-in-the-loop before creating ArgoCD deployment

8.4 USE CASE 3: TROUBLESHOOTING PLATFORM ISSUES

8.4.1 PROBLEM

Platform incidents and failed deployments often require engineers to manually correlate signals across multiple systems, including ArgoCD controllers, Kubernetes resources, application logs, and recent code changes. This fragmented troubleshooting process increases Mean Time To Resolution (MTTR) and places a high cognitive load on developers and on-call SREs, especially during time-sensitive incidents.

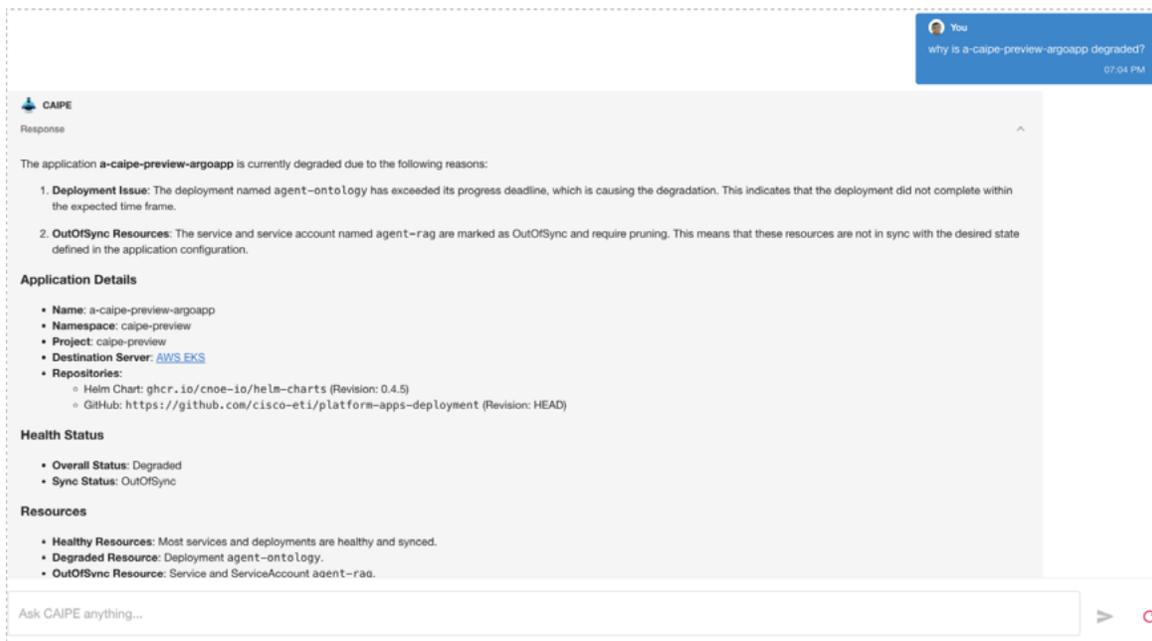
8.4.2 IMPLEMENTATION

CAIPE implements troubleshooting as a coordinated, agent-driven workflow led by a supervisor agent.

- The user initiates the workflow with a natural-language question (e.g., “Why did this service deploy in a degraded state?”).
- The supervisor agent gathers evidence by routing tasks to specialized agents:
 - **ArgoCD agent** to retrieve application sync status, rollout history, and deployment events
 - **Kubernetes agent** to inspect pod state, logs, resource constraints, and RBAC configuration
 - **GitHub agent** (optional) to analyze recent pull requests, commits, and configuration changes
- Collected signals are correlated across time, service, and environment.
- The supervisor synthesizes the findings and determines the most likely root cause.
- When remediation is clear, CAIPE generates a proposed fix , (such as a Helm), values change, and opens a remediation pull request with required approvals.

8.4.3 OUTCOME

Engineers receive a structured, evidence-backed explanation of deployment failures along with a recommended remediation. Troubleshooting time is reduced, corrective actions are consistent and auditable, and platform reliability improves without sacrificing governance or control.



The screenshot displays a chat interface for CAIPE. At the top right, a blue header shows a user profile icon labeled 'You' and the question: 'why is a-caipe-preview-argoapp degraded?' with a timestamp of '07:04 PM'. The main content area, titled 'CAIPE Response', provides a detailed explanation of the degradation. It states that the application 'a-caipe-preview-argoapp' is degraded due to two reasons: 1. **Deployment Issue:** The deployment 'agent-ontology' has exceeded its progress deadline. 2. **OutOfSync Resources:** The service and service account 'agent-rag' are marked as OutOfSync and require pruning. Below this, 'Application Details' lists: Name: a-caipe-preview-argoapp, Namespace: caipe-preview, Project: caipe-preview, Destination Server: AWS EKS, and Repositories: Helm Chart (ghcr.io/cnoe-io/helm-charts) and GitHub (https://github.com/cisco-eti/platform-apps-deployment). 'Health Status' shows Overall Status: Degraded and Sync Status: OutOfSync. 'Resources' lists Healthy Resources, Degraded Resource (agent-ontology), and OutOfSync Resource (agent-rag). At the bottom, there is a text input field with the placeholder 'Ask CAIPE anything...' and a send button.

8.5 USE CASE 4: SERVICE DESK MANAGEMENT

8.5.1 PROBLEM

Service desk operations rely heavily on structured processes, but interaction with service desk systems remains largely manual and fragmented. Engineers often need to switch contexts to create tickets, retrieve historical issues, track status, or generate reports. This slows response times, introduces inconsistency in ticket quality, and reduces visibility into operational trends.

8.5.2 IMPLEMENTATION

CAIPE implements service desk management as an agent-driven interaction layer over existing service desk systems.

- The user initiates actions such as creating tickets, searching issues, or generating reports using natural language.
- The supervisor agent validates intent and routes requests to the Service Desk Agent.
- The Service Desk Agent performs:
 - Ticket creation with structured fields and standardized templates
 - Retrieval of existing issues based on project, status, labels, or time range
 - Aggregation and summarization of ticket data for reporting
- Retrieved data is normalized and summarized before being returned to the user.
- All actions respect existing access controls, project boundaries, and approval rules.

8.5.3 OUTCOME

Engineers can interact with service desk systems directly from CAIPE without manual navigation or context switching. Ticket creation is consistent, retrieval is faster, and operational reporting becomes accessible on demand. Platform and support teams gain improved visibility into issue trends while maintaining governance and auditability.

NO MORE MESSAGES

You
show all OPENS ticket related to caipe in last 7 days
11:13 PM

CAIPE
Response

Date used for Jira query is 2025-11-24.

Found 20 issues using JQL: (project = "OPENS" OR text ~ "CAIPE") AND (created >= "2025-11-18" OR updated >= "2025-11-18") AND (created <= "2025-11-25" OR updated <= "2025-11-25")

Here are the details of the issues:

Issue	Title	Assignee	Reporter	Created	Resolved	Days to Resolve

8.6 USE CASE 5: INCIDENT MANAGEMENT

8.6.1 PROBLEM

Incident response requires rapid detection, accurate signal correlation, and decisive action under pressure. Signals are often distributed across alerting systems, deployment tools, cloud infrastructure, and cluster state, forcing responders to manually piece together context. This increases mean time to resolution (MTTR) and raises the risk of incomplete or incorrect remediation during high-severity incidents.

8.6.2 IMPLEMENTATION

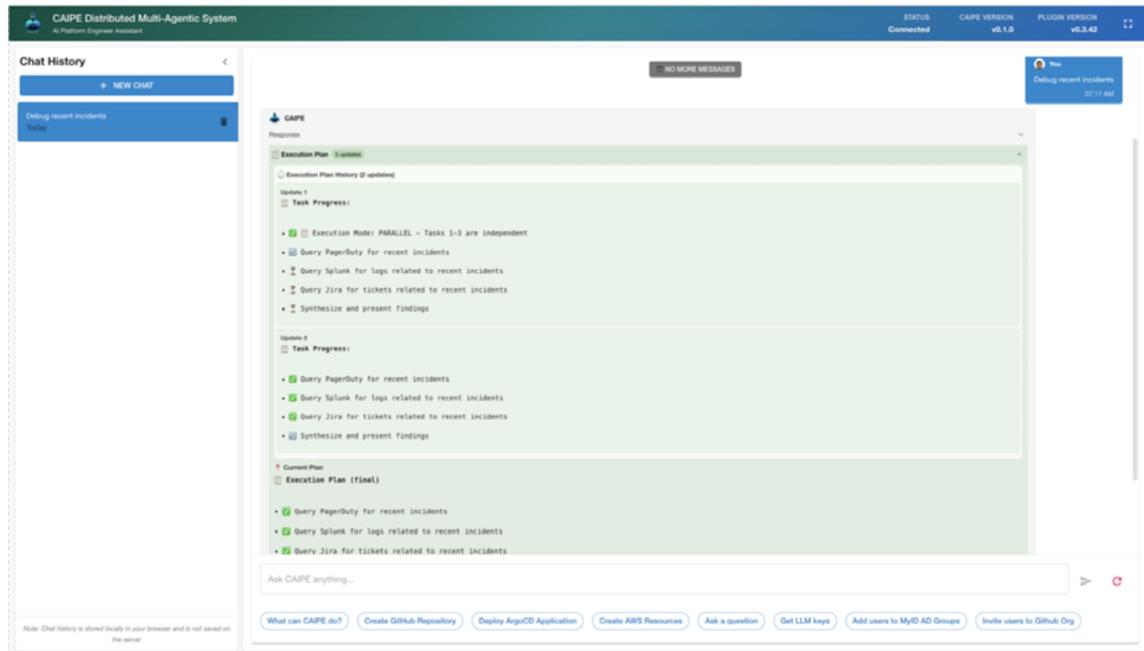
CAIPE implements incident management as a coordinated, multi-agent workflow led by a supervisor agent.

- The user or alert initiates an incident workflow (e.g., "Service X is down").
- The supervisor agent orchestrates collaboration across specialized agents:
 - **PagerDuty agent** to retrieve alerts, escalation policies, and page the correct service owner
 - **ArgoCD agent** to evaluate recent deployments, rollouts, and synchronization state
 - **Kubernetes agent** to inspect cluster health, pod status, logs, and resource constraints
 - **AWS agent** to retrieve cloud resource state, including infrastructure health, capacity, networking, and managed service status
- Signals are correlated across alerts, deployments, cloud infrastructure, and runtime state.
- The supervisor deep agent evaluates whether the collected evidence is complete and identifies gaps.
- The supervisor synthesizes findings and proposes a remediation plan, such as rollback of a recent deployment, cloud-side configuration changes, or a remediation pull request with required

approvals.

8.6.3 OUTCOME

Incident responders receive a unified, end-to-end view spanning application, cluster, and cloud infrastructure layers. The remediation plan is derived from platform team playbooks and established best practices retrieved from the knowledge base, ensuring responses are consistent, validated, and aligned with organizational standards.



9 Impact at Outshift



Thanks to CAIPE, the Outshift platform engineering team resolves issues faster, self-serves more safely, and offloads repetitive work to free engineers to focus on higher-value tasks while improving Mean Time To Recovery (MTTR) and overall developer experience.

- Augmented capacity: CAIPE-powered support desk now supplements the effort of ~3 engineers.
- Faster answers: Typical query response time dropped from hours to seconds.
- Workflow acceleration: ~30% of a day's tasks can be completed in minutes.

10 How is CAIPE evaluated?

CAIPE evaluates agents using trace-driven methods across both offline and online runs. Every request to the supervisor (platform engineer) agent yields a single-turn multi-agent trace (supervisor → sub-agents → tools) capturing inputs, outputs, and outcomes at each hop. Offline evaluations run against curated datasets and sample datasets in pre-production, while online evaluations validate behavior in shadow or safety-scoped production conditions.

10.1 TRACING FOUNDATION

- The supervisor issues a trace ID that propagates in A2A messages to sub-agents.
- All delegations and tool calls are correlated into a single-turn multi-agent trace for reproducibility, audit, and forensics.
- Traces record who acted, with what inputs, which tools, and what results.

10.2 EVALUATION METHOD (OFFLINE AND ONLINE)

Offline (pre-prod): Runs on datasets and sample datasets to measure correctness, determinism, and latency without production risk—processed by the [AGNTCY metrics compute engine \(MCE\)](#) to produce normalized metrics and scorecards.

Online (prod shadow/safety-scoped): Mirrors real usage, collecting the same metrics from live traces to confirm results under production variability.

A lightweight webhook evaluation service executes custom evaluators on captured traces:

1. An evaluation runner issues a representative task to the supervisor.
2. The resulting trace (including sub-agent/tool spans) is extracted.
3. Evaluators score trajectory, tool use, and answer quality against expectations.

10.3 EVALUATORS, METRICS AND RUBRICS

- **Trajectory-match:** Confirms the supervisor routing to the correct sub-agent(s) for a given task.
- **Tool-match (API):** Verifies the selected sub-agent invoked the correct tool.
- **LLM-as-a-judge (Rubric-graded):** Scores response on:
 - **Correctness:** How accurately does the response match the expected response.
 - **Helpfulness:** How effectively the response helps user complete the task

Captured per run and trace (offline and online):

- **Latency** (end-to-end and per step)
- **Cost** (token/compute per run)
- **Trajectory-match score (API)**
- **Tool-match score (API)**
- **LLM-as-a-judge scores (correctness, helpfulness)**
- **Trace links** for drill-down and aggregate stats (mean, p95) for run-to-run comparison.

CAIPE uses LLM tracing based metrics to make agent behavior measurable, reproducible, and governable. It pairs offline evaluations on curated datasets and sample datasets which are processed by the AGNTCY MCE along with online evaluations from measuring online production runs. Together, these checks continuously test changes to system prompts, routing logic to sub-agents and tools. This approach helps in catching regressions early, quantifying improvements, and driving steady iteration.

cone / new-trace / Traces

Trace **ai-platform-engineer: 2b69622e5246f1d7677ce1c9dd9438e9**

2025-09-24 23:05:55.948
 Latency: 1m 31s Env: default \$0.029876

Preview Scores Formatted JSON

Input: null

Output: null

Metadata

Path	Value
resourceAttributes	4 items
telemetry.sdk.language	"python"
telemetry.sdk.name	"opentelemetry"
telemetry.sdk.version	"1.33.1"
service.name	"unknown_service"
scope	3 items
name	"langfuse-sdk"
version	"3.3.4"
attributes	1 items

LLM-as-a-Judge Evaluators

Current default model: gpt4.1 / gpt-4.1 [+ Set up evaluator](#)

Running Evaluators Evaluator Library

Search (Name) Filters Columns 11/11

Generated Score Name	Status	Result	Logs	Referenced Evaluator	Created At	Updated At	Tar
Correctness	active	21	View	Correctness	9/14/2025, 12:34:45 PM	9/24/2025, 11:06:16 PM	dat
Correctness	active	48	View	Correctness	9/14/2025, 11:57:41 AM	9/14/2025, 11:57:41 AM	dat
Correctness	inactive		View	Correctness	9/14/2025, 9:36:02 AM	9/14/2025, 9:36:08 AM	dat
Helpfulness	active	101	View	Helpfulness	9/11/2025, 10:59:47 AM	9/11/2025, 11:03:10 AM	dat
Correctness	inactive	5	View	Correctness	9/11/2025, 10:56:11 AM	9/11/2025, 10:58:55 AM	dat

cone / new-trace / Datasets / single_agent_tests / Runs

Dataset run **single_agent_tests_0924_1404** Compare

Columns 12/12

Dataset Item	Trace	Latency	Cost	# Correctness (eval)	# routing_score (api)	# tool_match_score (api)
komodor_get_clusters		25.36s	\$0.083318	0.0000	1.0000	1.0000
slack_channel_info		22.16s	\$0.029174	0.0000	1.0000	1.0000
slack_list_channels		17.41s	\$0.02865	0.0000	1.0000	1.0000
pagerduty_get_incidents		1m 48s	\$0.335298	0.0000	1.0000	1.0000
pagerduty_list_services		39.36s	\$0.065618	0.0000	1.0000	1.0000
jira_search_issues		21.34s	\$0.032292	0.0000	1.0000	1.0000
jira_list_projects		12.05s	\$0.021586	0.0000	1.0000	1.0000
confluence_search_content		20.71s	\$0.04047	0.2000	1.0000	1.0000
confluence_list_spaces		45.41s	\$0.10508	0.0000	1.0000	1.0000
backstage_get_entity_deta...		6.54s	\$0.014066	0.0000	1.0000	1.0000
backstage_list_entities		46.50s	\$0.015942	0.0000	1.0000	1.0000
argocd_list_applications		18.04s	\$0.037934	0.0000	1.0000	1.0000
argocd_version		26.77s	\$0.03813	0.0000	1.0000	0.0000
github_get_commit_history		13.28s	\$0.039094	0.0000	1.0000	1.0000
github_list_repositories		36.20s	\$0.10183	0.0000	1.0000	1.0000
github_repo_description		19.61s	\$0.068416	0.0000	1.0000	1.0000

This routing decision was correct. The supervisor routed the request to the komodor_tools_agent, which is the expected agent to handle such requests related to Kubernetes clusters in Komodor.

11 Security

- **Identity and auth:** Use A2A authentication with [AGNTCY Identity](#) so every agent call is verified, scoped, and attributable.
- **Least privilege:** Capability- and environment-scoped tokens (e.g., read vs. mutate; dev/stage/prod).
- **Approvals:** Risk-tiered HITL gates; prefer reversible change paths (PRs, progressive rollouts).
- **Secure messaging:** SLIM for trusted, low-latency agent messaging in group channels.
- **Data and secrets:** Short-lived tokens, vault-managed secrets, and standard microservice security practices.
- **Audit and observability:** End-to-end traces for compliance and continuous evaluation.
- **Supply chain:** Trusted sources, pinned versions, signature verification, software bill of materials (SBOMs), and staged rollouts.

12 Lessons learned

12.1 DISTRIBUTED MULTI-AGENT ORCHESTRATION IS COMPLEX

- Design for fan-out/fan-in, long-running tasks, retries/backoff, sub-agent timeouts, health checks.
- Special consideration needs to be taken to adapt protocols and supervisor agent to stream



sub-agent responses and tool notifications to delivery responsive MAS system.

- Apply learnings learned from distributed microservices in cloud native ecosystems

12.2 VERSION EVERYTHING

- Treat agents, tools, prompts, LLM params and eval suites as versioned, contract-backed artifacts (MCP/A2A/OASF).

12.3 CONTEXT ENGINEERING AND DEEP AGENTS

- Supervisor and sub-agent context sharing and usage of short-term and long-term memory results in best results.
- Use planner-based deep agents to keep multi-step work on plan.

12.4 PLAN FOR DETAILED AGENT EVALUATION

- Have a detailed offline and online evaluation strategy and incrementally add it.

12.5 LLMs ARE NOT STATIC

- LLMs are constantly improving and getting better.
- It's important to pin LLM versions and params.
- Use agent evaluations to re-tune system prompts as models evolve.

12.6 TREAT THE AGENT UI AS A FIRST-CLASS CITIZEN

- Treat agent UX as “Core Feature” of agentic experience so users have the best experience interfacing with MAS systems.
- Provide schema-driven input forms, action widgets, visual execution plans, streaming output, and rich markdown so chat feels like a guided app.

12.7 PREFER STANDARD PROTOCOL-COMPATIBLE AGENTS

- Adopt industry standard protocols like MCP/A2A/SLIM to maximize reuse and future compatibility with MAS systems.

12.8 PROPAGATE USER CONTEXT IDENTITY

- Propagate user context end-to-end (on-behalf-of flows), enforce agent identity for agent↔agent, use scoped tokens for agent↔tool interactions.

13 CAIPE roadmap

CAIPE's next phase centers on scalable multi-agent collaboration, strong identity and on-behalf-of execution, auto-generated agents, and rigorous evaluation.

Here are some roadmap themes:

Multi-agent conversation (SLIM): Advance group-chat swarming for incidents/rollouts (dynamic join/leave, policies, transcripts → traces).

Identity and zero trust: Zero-trust agent identity plus user-context transfer (on-behalf-of) for scoped, auditable actions across sub-agents.

Agent Directory and artifacts: Generate new agents from OASF specs; auto-register tools/scopes; track prompts/tools/evals as versioned artifacts.

MCE and evaluations: Expand AGNTCY metrics compute engine (MCE) and offline/online evals (trajectory-match, tool-match, correctness/helpfulness) with dashboards and regression alerts.

More info is available in the Community Project board:

<https://github.com/orgs/cnoe-io/projects/9>

14 Get involved in CAIPE open source community

CAIPE is an open source project under [Cloude Native Operational Excellence \(CNOE\)](#) community's agentic AI SIG.

Please join the [community office hours](#) weekly to learn and improve CAIPE.

15 References

- CAIPE GitHub Repo: <https://github.com/cnoe-io/ai-platform-engineering>
- CAIPE Docs: <https://cnoe-io.github.io/ai-platform-engineering>
- CAIPE Community Meetings: <https://www.youtube.com/@cnoe-community>
- AGNTCY docs:
 - <https://docs.agntcy.org>
 - OASF: <https://docs.agntcy.org/oasf/open-agentic-schema-framework>
 - SLIM: <https://docs.agntcy.org/messaging/slim-core/>
 - Identity: <https://docs.agntcy.org/identity/identity/>
 - MCE: <https://docs.agntcy.org/obs-and-eval/evaluation/>
- Evaluations:
 - <https://github.com/agntcy/telemetry-hub>
 - <https://github.com/langchain-ai/agentevals>
- Outshift blogs (Original JARVIS Intro):
 - <https://outshift.cisco.com/blog/jarvis-agentic-platform-engineering-outshift>
 - <https://outshift.cisco.com/blog/architecting-jarvis-technical-deep-dive-into-its-multi-agent-system-design>
- Deep Agents: <https://github.com/langchain-ai/deepagents>
- Context Engineering: <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>



outshift.com ↗

@ outshiftbycisco