SASVA™ 3.0 IDE Plugin

User Guide

Software released on: September 25, 2025

Document updated on: September 29, 2025

Persistent

Legal notices

Warranty

The only warranties for products and services are set forth in the express license or service agreements accompanying such products and services. Nothing in this document should be considered as a promise of an additional warranty of any kind, implied, statutory, or in any communication between them, including without limitation, the implied warranties of merchantability, non-infringement, title, and fitness for a particular purpose. Persistent Systems shall not be liable for technical or editorial errors or omissions contained herein.

The information herein is subject to change anytime without notice.

Restricted rights legend

Copyright notices

Trademark notices

Revision history

Version	Published date	Description of change
1.0	September 25, 2025	Initial release

To ensure you are referring to the latest version of this document, refer to the following link:

https://support.accelerite.com/hc/en-us/categories/34222689823245

Release notes

The initial release of SASVA lays a strong foundation for scalable and flexible AI model serving and orchestration. This version introduces several key capabilities designed to enhance developer productivity and user experience:

- **Modern user interface**: A redesigned, user-centric interface that simplifies navigation and improves usability across workflows.
- **Multiple operational modes**: SASVA now supports various modes tailored to different development and deployment needs, offering flexibility for experimentation, staging, and production environments.
- **Enhanced LLM responses**: Optimized handling of large language model outputs for improved relevance, coherence, and contextual accuracy.
- **MCP integration**: Native support for Model Context Protocol (MCP) enables dynamic context switching and intelligent model routing.
- **@Context and #Hashtag**: Innovative tagging mechanisms to enrich prompts and improve model interpretability and response targeting.
- **Multi-model support**: SASVA supports a wide range of AI models including LLMs, vision models, and custom deployments, enabling diverse use cases.
- **Service provider integration**: Seamless connectivity with multiple model hosting providers for scalable inference and deployment.

Overview

SASVA is an AI code assistant specifically designed for direct integration into developers' Integrated Development Environments (IDEs). Its foundational objective is to amplify the capabilities of developers, providing powerful AI assistance that enhances productivity and streamlines workflows, rather than aiming for complete automation of the coding process.

The platform functions as a flexible framework, enabling developers to create, share, and utilize custom AI code assistants tailored to their specific needs. This is achieved through its extension for VS Code, complemented by the SASVA central server.

Get started

This section specifying system requirements for plugin installation.

System requirement

Minimum system requirments:

Hardware

- Processor: 1.6 GHz or faster
- RAM: 1 GB
- Hard Disk Space: 200 MB for installation, and additional space for projects and extensions.
- Display: 1024 x 768 or higher resolution.

Software

- VS Code 1.97 or later
- Node
- Python

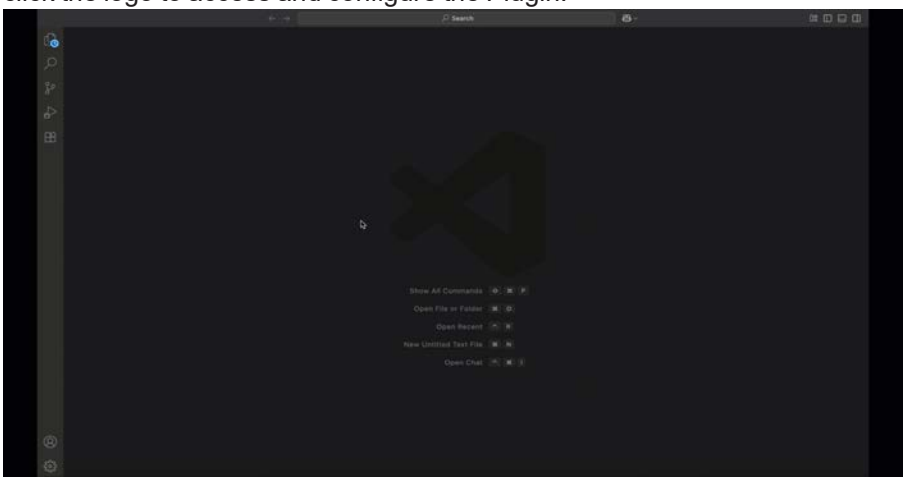
Supported operating systems

- Windows
- macOS
- Linux

Install

Obtain the server URL and authentication token from the SASVA administrator.

1. Open VS Code on your system.
2. In the left sidebar, click the **Extensions** icon.
3. In the search bar, type **SASVA** and select the **SASVA – Generative and Deterministic AI Platform**.
4. Click the **Install** button. Use the **Auto Update** checkbox to enable or disable automatic updates.
5. After installation, the SASVA Plugin logo will appear on the left sidebar in VS Code. You can click the logo to access and configure the Plugin.



First-time setup

Once the SASVA Plugin is installed, it must be configured to connect with the SASVA environment. This initial setup ensures secure access and enables you to run SASVA programs directly within VS Code.

1. Open the **SASVA Plugin** from the left sidebar.
2. Follow the steps mentioned in the Server Settings section.
3. Once saved, the **Models menu** will populate automatically with available models.

Uninstall

1. Open VS Code on your system.
2. In the left sidebar, click the **Extensions** icon.
3. In the search bar, type **SASVA** and select the **SASVA – Generative and Deterministic AI Platform**.

4. Click **Uninstall** and follow the steps to delete the user data:
   a. On Windows-
      - Go to c:\`user profile\ .vs code\ extensions` in local directory and delete the `persistent.sasva` and `.sasva` folder
      - Go to c:\`user profile\`in local directory and delete the `.sasva` folder
   b. On macOS/ Linux-
      - Open terminal and navigate to `.vs code\ extensions` in local directory and delete the `persistent.sasva` folder.
      - Open terminal and delete the `.sasva` folder from root directory.

Know your plugin

This section outlines the various components of the SASVA Plugin to help developers utilize them effectively and gain the maximum benefit.

Profiles

Profiles define the scope of user access and the operational capabilities available within the system. The profile specifies the permissions granted to the user, including which features, modes, and AI models can be accessed. This ensures that users interact only with the tools and resources appropriate to their role or authorization level. By managing access through profiles, the system maintains security, enforces compliance, and delivers a tailored user experience aligned with organizational policies and individual responsibilities.

Modes

Plugin offers distinct interaction modes to suit various development needs. These modes are accessible via a drop-down menu in the plugin interface and can be switched at any time without disrupting your workflow.

You can choose a mode based on the workflows described in the following sections.

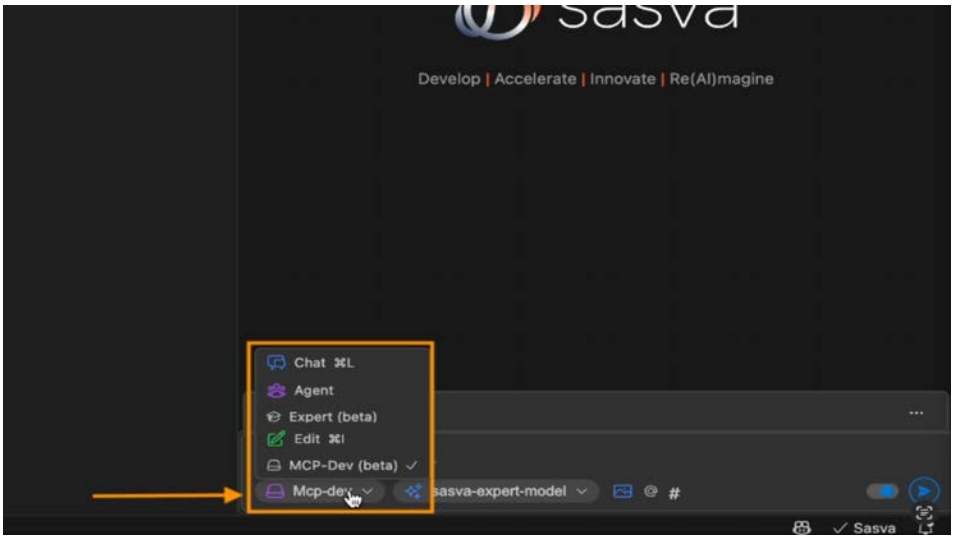
Mode	Purpose	Features	Use Cases	Benefits
Chat	Quick Q&A with the SASVA AI assistant using natural language This modes doesn't have Tools accessible in Chat	- Ask questions about your codebase or general programming concepts - Get explanations, code snippets, or suggestions without modifying code - Ideal for learning, debugging, or brainstorming	- “What does this function do?” - “How do I debounce an input in React?” - “Explain closures in JavaScript”	- fast and frictionless - No code changes - Great for quick insights and learning
Agent	Comprehensive codebase transformations with tool access. Handles	- SASVA plans and executes tasks across your project - Can create files,	- Build a login system with JWT authentication - Audit all	- High-level automation - Handles complex Interactive

Mode	Purpose	Features	Use Cases	Benefits
	complex tasks like generating entire programs, refactoring large sections, or interacting with external systems	run commands, debug, and iterate - Applies edits automatically, with safety checks for risky actions	server scripts for deprecated APIs - Create a dashboard with charts and filters - Implement Jira task SAS-2240	workflows
Expert	End-to end workflows and applications development.	- Resolve dependencies by its own - Integrates with external tools or API's - Can be configured for architectural discussions, performance tuning, or security audits	- Optimize this ML pipeline for GPU usage - Review this code for OWASP compliance - Generate a scalable microservices architecture - Build a React app that fetches user profiles from an API, displays them in a searchable table, and allows sorting by name or email	- Deep contextual understanding - Custom workflows - Ideal for senior developers, architects, or Casual developers - Ideal for rapid prototyping or large-scale refactoring
Edit	Works on selected codeblocks to streamlined code modification. Select code, provide instructions, and	- Select files or code blocks and describe the desired change - SASVA proposes inline edits for review before applying	- Refactor this to use async/await - Add error handling to this function - Convert this class to a	- You stay in control - Speeds up repetitive edits - Review changes before committing

Mode	Purpose	Features	Use Cases	Benefits
	get inline diffs for immediate application or rejection.	- Supports custom coding style instructions	functional component	
MCP-Dev	Specialized mode for creating and enhancing MCPs (Model Context Protocol)	- Automates MCP creation - Enhance automation for complex workflows	- Create an MCP to fetch figma design from URL - Enhance existing Jira MCP to comment directly on jira ticket. - Create an MCP which will fetch available component from storybook.	- Directly integrated withing SASVA - Create custom MCP as per recruitment - Bult-in testing feature for newly created MCP's
Deep agent	Orchestrate multi-agent workflows for distributed and complex tasks	- Coordinates multiple specialized agents - Plan and strategize complex tasks - Execute operations using various tools - Validate results for quality	- Create a web scraper for Amazon products - Research quantum computing and create a presentation - Analyze this codebase and suggest improvements	- Execute multi-step workflows - Work independently to completion

How to access modes

1. On the plugin chat window, click the **Mode Picker** menu.
2. From the drop-down list, select the desired mode.
3. Type your prompt in chat window and press **Enter** to view the result.



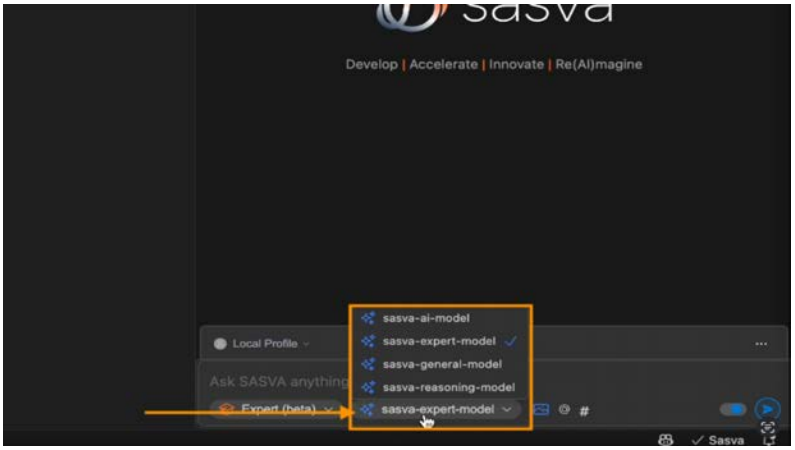
Models

SASVA offers extensive flexibility in AI model integration, allowing users to choose models based on performance needs, privacy preferences, and task complexity. Whether operating in a cloud-based environment, or within a secure enterprise setup, the plugin adapts seamlessly to diverse development workflows.

Additionally, model access is profile-aware, meaning users are granted access to models based on their assigned roles. This ensures that each team member from junior developers to senior architects or managers has the right level of capability and control, aligned with their responsibilities and skill level.

How to access models

1. On the plugin chat window click the **Model Picker** menu
2. From the drop-down list, select the desired model



Context

The plugin enhances SASVA's capabilities by delivering precise, relevant assistance through advanced context-awareness techniques. It leverages embedding and re-ranking models to efficiently index and identify similar code suggestions.

A key feature of the plugin is its ability to let you guide the SASVA assistant with accurate, context-rich information. Provide additional context by including workspace elements like files, terminal outputs, issues, images, or even your full codebase to make AI responses more accurate and relevant.

Context types

Here's a breakdown of the available context types and their use cases:

Context Type	Use Cases
@file	- Attaches a specific file from your workspace - Enables the assistant to read, analyze, and interact with the file contents
@code	- Attaches a code file for deeper analysis - Useful for asking questions, debugging, or requesting improvements
@terminal	- Includes terminal output or logs as context - Ideal for troubleshooting runtime errors or command-line workflows
@git diff	- Compares the current file with its version in Git - Helps identify changes, regressions, or missing updates
@problem	- References problems or diagnostics in the current file - Useful for resolving linting issues, warnings, or errors
@folder	- References problems or diagnostics in the current file - Useful for resolving linting issues, warnings, or errors
@project overview	- Generates a high-level analysis of the entire project - Useful for on-boarding, architecture review, or documentation
@fix	- Performs comprehensive codebase enhancements to fix defects - Can be combined with issues or code context for targeted fixes

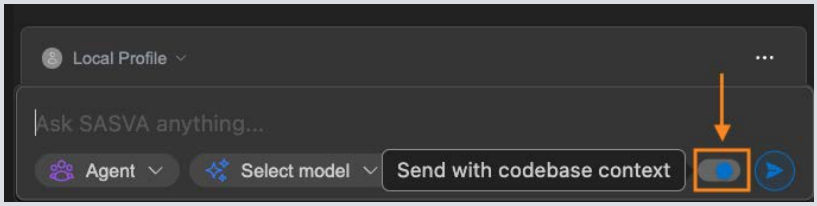
Context Type	Use Cases
@git issues	- Ensure that the **Git configuration** is completed - Enables the assistant to understand and help resolve specific issues - Attaches selected Git issues as context
@jira issues	- Ensure that the **Jira configuration** is completed - Attaches selected Jira issues as context - Useful for planning, fixing, or implementing features tied to tickets
@codebase	- Attaches the entire codebase as context - Can also be enabled via a **code context toggle** in the bottom bar  - Ideal for large-scale refactoring, audits, or feature additions

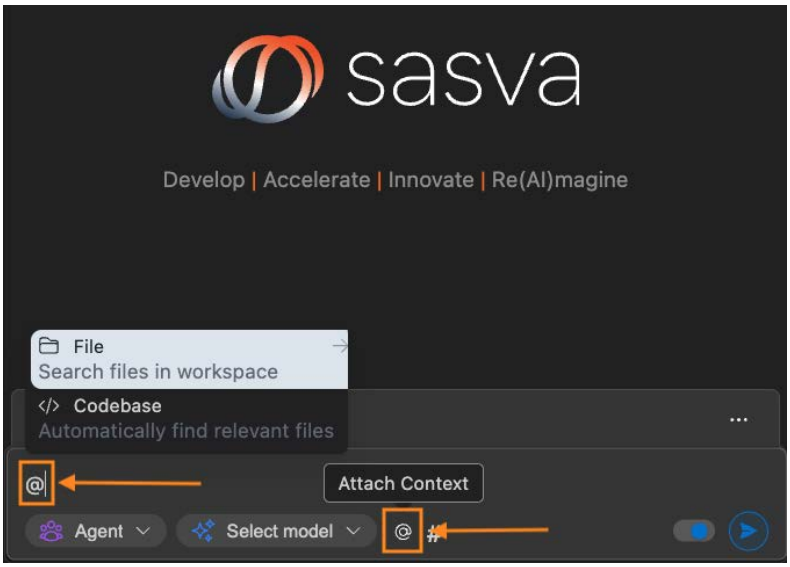
Image upload

- Upload images directly via the UI to use them as context
- Useful for UI mockups, screenshot, error messages, logs, diagrams or architecture visuals

How to add context

You can add context in two ways:

- **Typing @** directly in the chat bar
- **Clicking the @** icon in the plugin bottom bar



Once triggered, a context menu appears, allowing you to select one or more context types. You can **combine multiple contexts** to perform complex tasks more effectively.

Hashtag (Specialist)

When you select a hashtag, specialized system instructions are automatically appended to your prompt, enabling the SASVA AI assistant to tailor its responses to specific technical domains. This seamless integration enhances the precision and relevance of the AI's answers, ensuring you receive expert-level support for your queries.

Default hashtags

By default, the following four specialist roles are available:

#testing specialist: Focuses on test coverage, unit testing, integration testing, and test automation strategies

- Example prompt: #testing specialist create unit test case for following code

#security specialist: Adds instructions to consider security implications in every response.

- Example prompt: #security specialist Analyze this login flow for vulnerabilities

#performance specialist: Optimizes code for speed, memory usage, and scalability

- Example prompt: #performance specialist optimize the code for speed

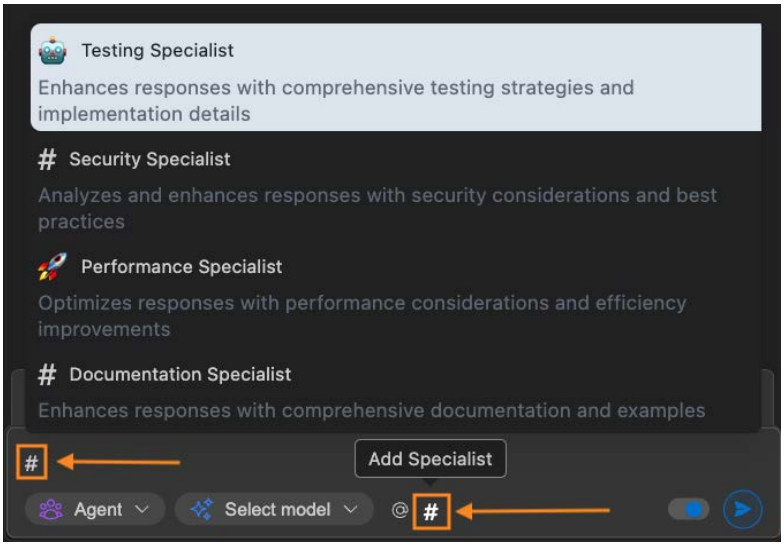
#documentation specialist: Generates clear, structured, and developer-friendly documentation

- Example prompt: #documentation specialist Generate API docs for this module

How to use hashtag

You can add hashtag in two ways:

- **Typing #** directly in the chat bar
- **Clicking the #** icon in the plugin bottom bar



Once triggered, a specialist menu appears, allowing you to select specialist.

How to add new hashtag

You can create and add specialist

1. Navigate to `C:\users\<username>\.sasva\data` in local directory.
2. Open the `specialist.json` file in editor.
3. The file contains a list of previously added specialists. You can:
   - **Edit** existing specialist
   - **Disable** a specialist

- **Add a new** specialist by inserting a new JSON object using the format below:

```
    {
      "name": "Use Common Components",
      "description": "Ensures implementation uses only existing common
components from the project without modifying them",
      "icon": "   ",
      "systemPrompt": "You are a common components specialist. Your primary role
is to guide implementations to use ONLY the existing common components available
in the project. NEVER suggest modifying existing common components. You may
suggest creating new components when needed, but always prefer using existing
ones. Always analyze the project's component library first and recommend the
most appropriate existing components for any given task.",
      "responseProcessor": {
        "type": "prefix",
        "instructions": "Before providing the main response, add a \"Common
Components Usage\" section that includes: 1) List of existing common components
from the project that should be used for the implementation, 2) Specific usage
examples showing how to properly import and use these components, 3) Clear
warnings against modifying existing components with alternative approaches if
customization is needed, 4) If new components are required, provide clear
justification and ensure they complement existing ones without duplication, 5)
Component composition patterns showing how to combine existing components
effectively."
      },
      "enabled": true
    }
```

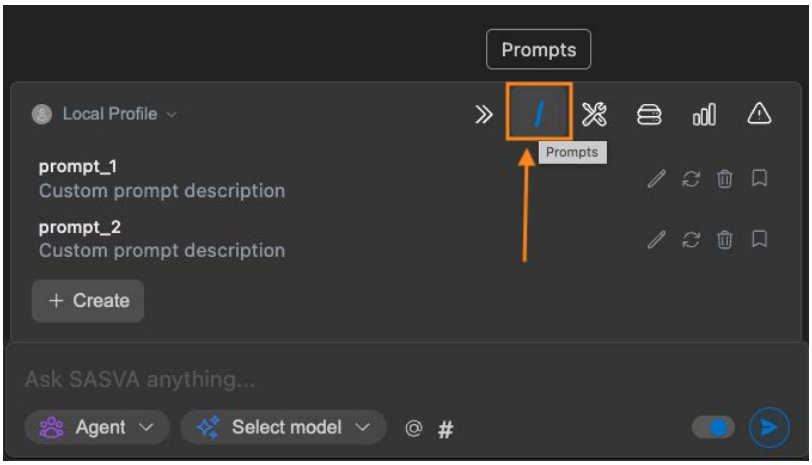
4. **Save** the file to apply your changes. Type # in the chat window to test new specialist.

Prompts

The prompts feature allows users to save and reuse frequently used instructions, making interactions with the AI assistant faster, more consistent, and tailored to specific workflows.

How to access saved prompts

1. Type / in the chat bar to bring up a list of saved prompts.
2. Select any prompt from the list to instantly populate it in the chat window.



This is ideal for recurring tasks like code reviews, documentation generation, or bug analysis.

How to create a new prompt

To create a new prompt:

1. Click the **Expand Toolbar** in the plugin UI.
2. Click the **/** icon.
3. Click **+Create** to add a new prompt.
4. A new **YAML file** opens where you can define:
   - **Name**: A short, descriptive title
   - **Description**: What the prompt is intended to do
   - **Prompt**: The actual instruction or query to be reused
5. **Save** the file to make the prompt available for future use.

How to manage prompts

Plugin provides a simple and intuitive interface to manage saved prompts, allowing you to edit, organize, and reuse them efficiently. To manage your saved prompts:

1. Click the **Expand Toolbar** in the plugin UI.
2. Click the **/** icon.
3. Browse the list of saved prompts. For each prompt in the list, you can perform the following operations:
   - **Edit**: Modify the name, description, or content of the prompt. This opens the YAML file for direct editing
   - **Refresh configuration**: Reloads the prompt configuration to reflect any recent changes made in the YAML file
   - **Delete**: Permanently removes the prompt from your list

- **Bookmark the prompt**: Mark frequently used prompts for quick access. Bookmarked prompts appear at the top of the list

MCP servers

Overview

An MCP Server is a service that implements the Model Context Protocol (MCP)—an open standard designed to connect AI systems with external data sources and services. The MCP Server acts as a bridge between AI applications (called MCP Clients) and the real world, enabling secure and standardized communication.

Key responsibilities of an MCP Server:

- Expose Tools and Resources: It provides a set of tools and resources that the MCP Client can use to perform actions or retrieve data.
- Standardized Interface: By following MCP specifications, it ensures consistent integration across different AI models and applications.
- Secure and Controlled Access: It manages permissions and ensures that only authorized actions are performed.

MCP Tools overview

An MCP Tool is a function or capability exposed by an MCP Server. These tools are part of the MCP, which provides a common interface for integrating AI systems with external services and data source. MCP Tools enable AI applications to:

- Access databases
- Call external APIs
- Read or write files
- Automate workflows

Default MCP servers

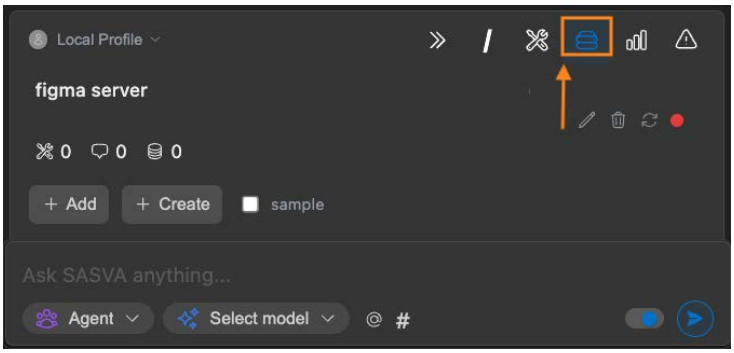
SASVA comes pre-configured with three MCP servers:

- Memory MCP – Handles session memory and context retention
- Web Navigation MCP – Enables browsing and interaction with web content
- GitHub MCP – Integrates with GitHub for issue tracking, repository access, and commit analysis

How to access MCP server panel

To manage MCP servers:

1. Click the **Expand Toolbar**.
2. Click the **MCP Server** icon.



3. The **MCP Management Panel** opens, displaying all configured MCP servers.

How to add MCP server

1. Click the **Expand Toolbar**.
2. Click the **MCP Server** icon.
3. Click the **+Add** button in the MCP panel.
4. A new **YAML file** opens where you can define:
   - **Name**: A unique identifier for the MCP server. For example:`mcp-github` or `mcp-custom-analytics`
   - **Command**: Specifies the executable used to run the MCP server. For example: `node`, `python`,`npx`
   - **Arguments**: Arguments passed to the command during execution. These arguments vary depending on the technology stack or the server's requirements. They may include flags, script names, or configuration paths. For example:

     ```
     args:
       - -y
       - mcp-serve
     ```

   - **Environment**: Environment variables to be set when the MCP server runs. This allows you to pass secure credentials, toggle debug modes, or configure runtime behavior. For

example:

```
env:
  API_KEY: "your-api-key"
  DEBUG: "true"
```

5. **Save** the file to register the new MCP server.

Best Practices

- Use descriptive names for clarity, especially when managing multiple MCPs
- Ensure the command and args match the server's tech stack and entry point
- Keep environment variables minimal, secure and avoid hardcoding sensitive data

How to create MCP server

1. Click the **Expand Toolbar**.
2. Click the **MCP Server** icon.
3. Click the **+Create** button in the MCP panel. You can check the **sample box** to get the sample math MCP server.
4. A new **YAML file** opens where you can define:
   - **Name**: A unique identifier for the MCP server. For example:`mcp-github` or `mcp-custom-analytics`
   - **Command**: Specifies the executable used to run the MCP server. For example: `node,python,npx`
   - **Arguments**: Arguments passed to the command during execution. These arguments vary depending on the technology stack or the server's requirements. They may include flags, script names, or configuration paths. For example:

     ```
     args:
       - -y
       - mcp-serve
     ```

   - **Environment**: Environment variables to be set when the MCP server runs. This allows you to pass secure credentials, toggle debug modes, or configure runtime behavior. For

example:

```
env:
  API_KEY: "your-api-key"
  DEBUG: "true"
```

5. **Save** the file to register the new MCP server.

How to manage MCP server

The MCP Management Panel allows you to view, configure, and control all MCP servers integrated with the SASVA plugin. To manage MCP server:

1. Click the **Expand Toolbar**.
2. Click the **MCP Server** icon.
3. Browse the list of saved MCP. Each MCP server include a set of:
   - **Tools**: Functional modules that automate tasks (e.g., file creation, Git operations)
   - **Prompts**: Predefined instructions or templates tailored to the server's domain
   - **Resources**: External data or connectors (e.g., APIs, file paths, memory blocks)
4. . For each MCP in the list, you can perform the following operations:
   - **Edit**: Modify the name, command, argument or environment of the MCP. This opens the YAML file for direct editing
   - **Delete**: Permanently removes the MCP from your list
   - **Refresh configuration**: Reloads the MCP configuration to reflect any recent changes made in the YAML file
   - **Status Monitor**: Displays current status of MCP: **Green**- Active and functioning, **Amber**- Warning or partial functionality, and **Red**- Inactive or error state

Built-in tools

Built-in tools are integrated utilities within the plugin that automate repetitive and system-level development tasks. They are designed to simplify complex workflows by reducing manual effort during operations such as component creation and configuration. Availability of these tools depends on the profile assigned to you.

Note: Enabling tools within a plugin increases the prompt's context length because each tool contributes metadata such as its name, description, and input schema to the prompt. This additional information is included every time the prompt is constructed, leading to higher token utilization. As a result, enabling multiple tools may impact performance in environments with limited context windows.

Key capabilities include:

- Generate and organize multiple files as part of component scaffolding
- Perform file system and Git operations for seamless version control
- Manage in-memory state to maintain context across tasks
- Execute predefined actions without requiring user input

Available tools

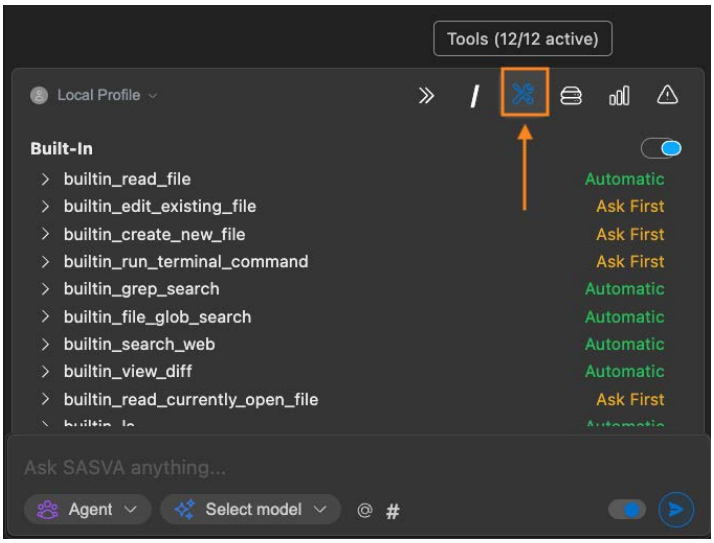
The number and type of tools available in the menu depend on the MCP Server configuration. You can expand the toolset by configuring additional MCP Servers via the MCP Server settings. By default, the following tools are included:

- Memory Tools: Handle session memory, context retention, and prompt history
- File System Tools: Manage file creation, editing, and organization
- Web Navigation: Browse, extract, and manipulate information from websites, making it useful for tasks like research, automation, and data collection
- Git Tools: Perform Git operations like diffing, issue linking, and commit analysis

How to access built-in tools

To access and manage built-in tools:

1. Click the **Expand Toolbar** in the plugin UI.
2. Click **Tools** icon. A panel opens showing all available tools.



3. Use the **toggle switch** to enable or disable all tools globally.
4. You can expand each tool in the UI to view:
   - Description: What the tool does
   - Arguments: Parameters or settings used during execution
5. Each tool offers three configurable operation modes:

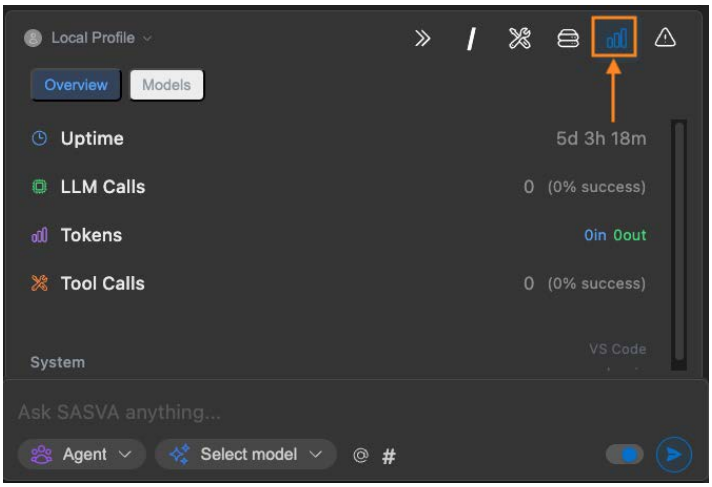
- **Automatic**: The tool runs without user intervention whenever needed
- **Ask First**: The plugin prompts before executing the tool
- **Exclude**: The tool is disabled and won't be used during operations

This gives fine-grained control over how each tool behaves during execution.

Statistics

The Session Statistics panel provides real-time insights into plugin activity, model usage, token usage, and tool performance. This helps users monitor system behavior, optimize resource usage, and troubleshoot issues effectively.

1. Click the **Expand Toolbar** in the plugin UI.
2. Click the **Session Statistics** icon.



3. The Session Statistics panel opens, displaying two sections:
   - **Overview** : The Overview section displays key metrics for the current session:
     - **Uptime**- Shows how long the plugin has been running since its last start
     - **LLM Calls**- Displays the total number of requests made to the large language model (LLM)
     - **Tokens**- Tracks tokens sent (in) and received (out)
     - **Tool Calls**- Number of tool calls with success rate (%)
   - **Models**: The Models section provides visibility into model usage:
     - Lists all models used in the current session
     - Shows the number of calls made to each model
     - Useful for understanding which models are most active and evaluating performance across different tasks

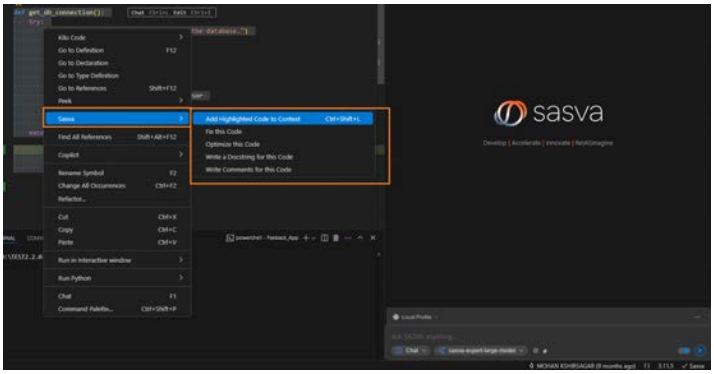
Context menu tools

SASVA provides a dedicated context menu within Visual Studio Code, allowing developers to access AI-powered coding assistance directly from the editor. The following options are available:

- **Add Highlighted Code to Context**: Adds the selected code snippet to SASVA's working context, improving the relevance of AI suggestions
- **Fix This Code**: Suggests fixes for syntax errors, logical bugs, or structural issues in the selected code.
- **Optimize This Code**: Recommends performance improvements or readability enhancements
- **Write a Docstring for This Code**: Automatically generates structured documentation for functions, classes, or methods
- **Write Comments for This Code**: Adds inline comments to explain logic and improve code clarity

How to use context menu

1. Select the code snippet you want to modify in the editor window.
2. **Right-click** to open the context menu.



3. In the context menu, click **Sasva** option.
4. Choose the appropriate action based on your task. SASVA suggest changes accordingly.
5. Accept or rejct the changes suggested by SASVA.

Keyboard shortcuts

SASVA provides a set of keyboard shortcuts to help developers work faster and more efficiently within Visual Studio Code. These shortcuts are available on both Windows and macOS, with platform-specific key combinations where applicable.

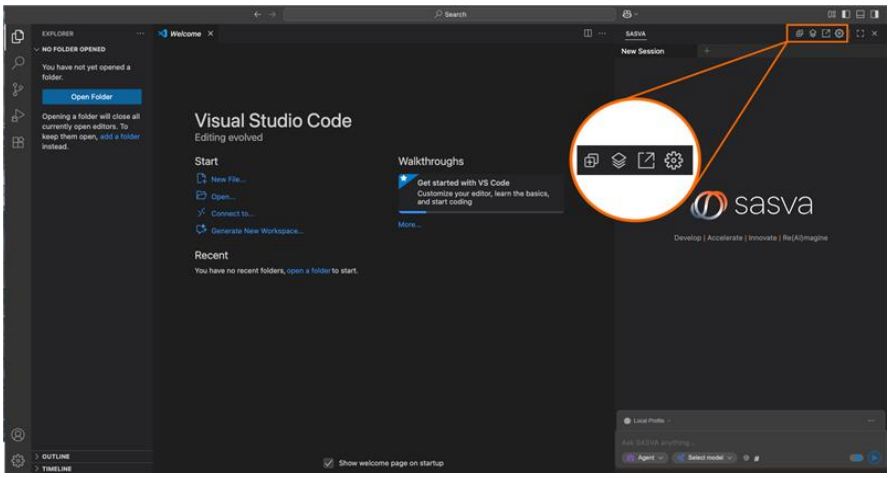
You can explore list of available shortcuts by entering >sasva in VS code search window.

Action	Windows	macOS	Description
Add highlighted code to context	Ctrl + Shift + L	Shift + Cmd + L	Adds the selected code snippet to SASVA's working context for better suggestions
Open saved prompts	/	/	Opens the list of saved prompts for quick selection and reuse
Accept diff	Ctrl+ Shift + Enter	Shift + Cmd + Enter	Accepts the suggested diff changes
Reject diff	Ctrl + Z	Cmd + Z	Rejects the suggested diff changes
Accept vertical diff block	Ctrl +Alt +Y	Option+Cmd+Y	Accepts a vertical diff block suggestion
Reject vertical diff block	Ctrl +Alt +N	Option + Cmd + N	Rejects a vertical diff block suggestion
Apply code from chat	Alt + A	Option + A	Applies code generated in the chat window to the editor
Debug terminal	Ctrl + Shift + R	Shift + Cmd + R	Opens the debug terminal
Detach window	Ctrl +K Ctrl + M	Cmd + K Cmd + M	Snaps the plugin window out of the IDE layout for flexible positioning
Exit edit mode	Escape	Escape	Exits the current edit mode
Focus SASVA chat	Ctrl + L	Cmd + L	Focuses the chat input field
Generate code	Ctrl + I	Cmd + I	Triggers code generation based on current context or prompt
Toggle autocomplete enable	Ctrl+ K Ctrl+ A	Cmd + K Cmd + A	Triggers code generation based on current context or prompt

Manage sessions

The plugin provides intuitive session management features that allows user to start fresh conversations, revisit previous work, and customize their workspace layout for better productivity.

Located at the top of the plugin interface, next to the **Preferences (gear)** icon, you'll find three key session management buttons:



New session

- Starts a fresh session with the SASVA AI assistant
- Ideal for switching tasks or beginning a new workflow

Saved session

- Opens a list of previous sessions
- Allows user to resume tasks from where they left off
- Useful for long-running projects or revisiting past interactions
- You can rename and delete past session.

Detach Window

The Detach Window feature lets you pop out the AI assistant chat panel from the IDE layout, giving you more flexibility to work across screens or alongside other tools without distraction..

Key features

- Independent Chat Window: Seamlessly detach the assistant chat panel from the plugin canvas, allowing it to float as a standalone window outside the IDE interface.
- Flexible Positioning & Resizing: Users can freely move and resize the chat window, placing it wherever it best supports their workflow.
- Enhanced Multitasking :Ideal for developers who prefer to keep the assistant visible while working across multiple panels or tools.

Cross-IDE Collaboration

This feature also supports multi-IDE workflows. Developers can open the same project in two different IDEs such as VS Code and another preferred platform and position the detached chat window next to one IDE while actively coding in the other. This setup enables real-time assistance without switching contexts, making it a powerful tool for productivity.

Use Cases

- Pair the assistant with your favorite IDE while keeping your main workspace clutter-free.
- Use the chat window on a second screen for uninterrupted guidance.
- Collaborate across IDEs with shared project views and centralized SASVA support.

Docking plugin/ Floating window / Arrange plugin position

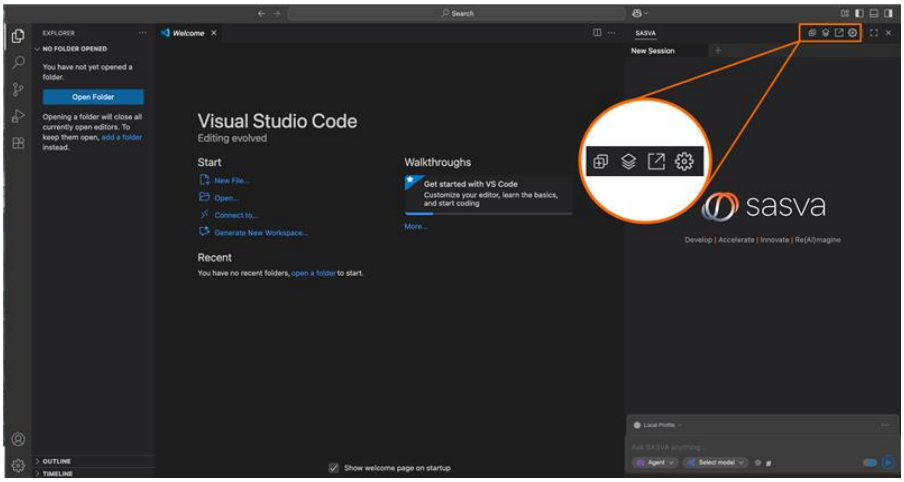
In addition to detaching the chat window, the plugin supports full UI repositioning:

1. **Click and hold** the plugin icon in the IDE
2. **Drag** the plugin to your **preferred location** within the IDE layout
3. In case plugin logo is not visible, press **Cmd+L** for macOS and **Ctrl+L** for windows to open the chat window

This allows you to dock the plugin in a way that best suits your workflow side-by-side with code, terminal, or file explorer.

Preferences

The preferences section allows you to configure and personalize the SASVA Plugin, streamlining integrations and optimizing the development experience. To access the settings click the **Preferences (Gear icon)** on top of the plugin interface.



Codebase Scan

Codebase scanning helps build a deeper understanding of the project currently open in your IDE workspace. By extracting knowledge from the codebase, it enables the model to construct a richer context, which enhances its ability to identify issues, assess code quality, generate insights, and support various development tasks. This foundation also facilitates the addition of new features and improvements with greater accuracy and relevance.

1. Once the project is opened in workspace.
2. he plugin will initiate a codebase scan automatically.To disable automatic scan set the **Enable Code Base Indexing** toggle to **Off** in "Preferences" above section.
3. To manually trigger a rescan, click the **Rescan** button in the Codebase Scan panel. **Note**: Performing a rescan will delete all previously extracted data and rebuild the context from scratch.

Server Settings

Connect the plugin to a SASVA server to automatically fetch and apply project settings. Contact administrator to obtain server URL and authentication token.

Follow the steps to configure sever settings:

1. Click the **Preferences (Gear icon)** on the top-right of the chat panel.
2. Open the **Configuration Server** drop-down.
3. Enter the **Server URL**.

4. Enter the **Authentication Token**.
5. Save to apply the configuration.

Update Server

Keeps the plugin up to date by connecting to a server for the latest features and fixes for on-prem deployments.

Follow the steps to update server configuration:

1. Open the **Update Server** drop-down.
2. Enter the **Update Server URL**.
3. Enter the **Update Authentication Token**.
4. Save to enable automatic updates.

GitHub Settings

Connect your GitHub account to sync repositories, track changes, and manage code. This setting is required to use the @git context feature. Follow the steps to configure GitHub repository:

1. Open the **GitHub Settings** drop-down
2. Enter your **GitHub Token**
3. Enter the **GitHub Owner**
4. Enter the **GitHub Repository**
5. Save to activate GitHub integration.

Jira Settings

Integrates Jira to allow issue creation, updates, and tracking directly from the plugin. This setting is required to use the @jira context feature. Follow the steps to configure Jira:

1. Open the **Jira Settings** drop-down
2. Enter your **Jira Email**
3. Enter your **Jira Domain**
4. Enter your **Jira Token**
5. Save to complete the integration

Display Settings

Customize the plugin’s appearance and layout to suit your workflow.

- **Show Session Tabs**: Toggle the setting to enable or disable the session tabs on the main window

- **Wrap Codeblocks**: Toggle to wrap long code lines within the chat window for better readability
- **Show Chat Scrollbar**: Toggle helps to show or hide chat scrollbar
- **Font Size Control**: Enter a value manually or use arrow buttons to increase or decrease font size

Code Assistance

Enable intelligent tools to help write, analyze, and improve code.

- **Enable Codebase Indexing**: Toggle to enable or disable automatic indexing of the codebase. Disabling this stops automated scans for the repository
- **Format Markdown**: Toggle to automatically format markdown content generated by the assistant
- **Show Session Titles**: Toggle to display titles for each session, helping you quickly identify and organize your work

Auto-suggest Settings

Configure how code suggestions appear and behave while typing. Follow the steps to configure:

1. Open the **Auto-suggest Settings** drop-down.
2. Choose behavior for **Multiline Autocompletions**:
   - Auto: Suggest when appropriate
   - Always: Suggest when appropriate
   - Never: Suggest when appropriate
3. Set **Autocomplete Timeout** (in microseconds): Defines how long the plugin waits before showing suggestions.
4. Set **Autocomplete Debounce** (in microseconds): Controls the delay between keystrokes and suggestion generation.
5. **Disable Autocomplete in Files**: Enter a **comma-separated list of file paths** where autocomplete must be **disabled**.

Administrators

The following sections contains the information IDE deployment, configuration and manage administrative tasks for IDE plugin.

Deployment architecture

Config Server (Port 8025)

The Config Server acts as the central configuration provider for SASVA3 services. It supplies runtime parameters, environment settings, and service-specific configurations to other components like the Gateway and Embedding servers.

Responsibilities:

- Serve configuration files and metadata
- Support production and development modes
- Provide a REST API or file-based access to configurations

Gateway Server (Port 8027)

The Gateway Server is the entry point for client requests. It routes incoming traffic to appropriate backend services such as the Embedding Server, LLM Servers, or MCP Server based on request type and context.

Responsibilities:

- Handle API requests
- Perform request validation and logging
- Route to Embedding or LLM servers
- Interface with MCP Server for context-aware routing

Embedding Server (Port 8026)

The Embedding Server processes textual data and converts it into vector representations using transformer-based models. These embeddings are used for downstream tasks like semantic search, classification, or LLM input preparation.

Responsibilities:

- Accept batch text input
- Generate embeddings using pre-trained models
- Optimize performance via batching and multi-worker support

HAProxy (Load Balancer)

HAProxy is used to distribute incoming traffic across multiple backend services, ensuring high availability and load balancing. It can also be used to expose a unified endpoint for Gateway and LLM servers.

Responsibilities:

- Load balance requests to Gateway and LLM servers
- Provide failover and health checks
- Support SSL termination if needed

Private LLM Servers

These servers host large language models (LLMs) that perform inference tasks such as summarization, question answering, or code generation. They are private to the SASVA deployment and not exposed publicly. Refer to "Models" on page 36 section to know more information about models.

Responsibilities:

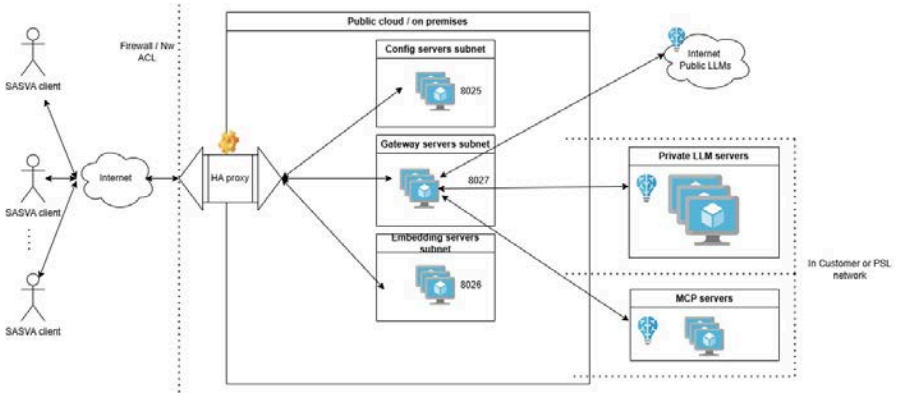
- Serve LLM inference APIs
- Handle context-aware requests from Gateway or MCP
- Maintain model isolation and performance

MCP Server (Model Context Protocol)

The MCP Server orchestrates model context switching and routing logic. It ensures that the correct model is selected based on request metadata, user session, or business rules.

Responsibilities:

- Manage model registry and context rules
- Interface with Gateway for routing decisions
- Provide APIs for dynamic context switching



Prerequisites

System requirements

- Ubuntu 20.04+ or compatible Linux distribution
- Python 3.12 installed

- pip, pyinstaller, and virtualenv available
- Sufficient CPU/RAM for embedding and LLM servers

User and permissions

- Deployment user: sasvauser (except embedding server and MCP which may use root)
- Ensure write access to `/data/<component-folder>` and `/etc/systemd/system/`

Component deployment

Common binary build steps

```
python3.12 -m venv env3.12
source env3.12/bin/activate
pip install -r requirements.txt --no-cache-dir
pip install pyinstaller
```

Config server (Port 8025)

Build

```
cd /data/config-server
source env3.12/bin/activate
pip install -r requirements.txt --no-cache-dir
pip install pyinstaller

pyinstaller --onefile --clean \
  --hidden-import=flask \
  --hidden-import=yaml \
  --hidden-import=gunicorn.glogging \
  --collect-submodules gunicorn \
  serverNew.py
```

Run manually

```
./serverNew --port 8025 --config . --production
```

Gateway server (Port 8027)

Build

cd /data/sasva-gateway-new

```
cd /data/sasva-gateway-new
source env3.12/bin/activate
pip install -r requirements.txt --no-cache-dir
pip install pyinstaller

pyinstaller --onefile --clean \
  --hidden-import=fastapi \
  --hidden-import=uvicorn \
  --hidden-import=yaml \
  --hidden-import=structlog \
  --hidden-import=_posixshmem \
  gatewaymain.py
```

Run manually

```
./gatewaymain
```

Embedding Server (Port 8026)

Build

```
cd /data/embedding-server
python3.12 -m venv env
source env/bin/activate
pip install -r requirements.txt --no-cache-dir
pip install pyinstaller

pyinstaller --onefile --clean \
  --collect-all transformers \
  --collect-all numpy \
  --hidden-import=sklearn \
  --collect-submodules sklearn \
```

```
  --hidden-import=fastapi \
  --hidden-import=starlette \
  --hidden-import=pydantic \
  --collect-submodules fastapi \
  --collect-submodules starlette \
  --collect-submodules pydantic \
  --collect-all transformers \
  --hidden-import=uvicorn \
  --collect-submodules uvicorn \
  --hidden-import=embedding_server \
  --collect-submodules embedding_server \
  embedding_server.py
```

Run manually

```
./embedding_server --port 8026 --workers 3 --batch-size 256 --max-texts 2000
```

HAProxy (Load balancer)

<add content>

Private LLM servers

<add content>

Configure MCP server

<add content>

Service configuration

<add content>

Post-deployment checks

Start and enables services

```
sudo systemctl daemon-reload
sudo systemctl enable <service-name>
sudo systemctl start <service-name>
```

Verify status

```
sudo systemctl status <service-name>
```

Log monitoring

```
tail -f /data/<component>/logs/*-access.log
tail -f /data/<component>/logs/*-error.log
```

Health check URLs

- Config Server: http://<vm-ip>:8025/health
- Gateway Server: http://<vm-ip>:8027/health
- Embedding Server: http://<vm-ip>:8026/health
- LLM Server: http://<vm-ip>:<port>/health (Add actual port)
- MCP Server: http://<vm-ip>:<port>/context-status (Add actual port and endpoint)

Models

SASVA supports a variety of models and service providers. Please refer to the section below for detailed information.

Self-hosted models

For advanced users or organizations, the plugin supports custom model deployments on private infrastructure. They also offer multiple benefit such as custom tuning and optimization, full control over model behavior , and enterprise grade scalability.

Features:

- Deploy your own LLMs using Docker, Kubernetes, or bare-metal setups.
- Integrate with internal APIs or model servers.
- Configure model routing and fallback strategies.

SASVA offers the following models for the self hosted deployments:

- sasva-ai-model
- sasva-expert model
- sasva-general-model
- sasva-reasoning-model
- sasva-expert-model

Local models

Run directly on the user's machine or local server, these models offer privacy and offline capabilities. These models are ideal for secure or air-gapped environment.

Supported Engines:

- Ollama (supports models like LLaMA, Mistral, Code Llama)
- Jan AI
- ggml (optimized for CPU inference)

Cloud services

These models are hosted by third-party providers and accessed via APIs. They offer high performance and are ideal for general-purpose tasks.

Supported Provider:

- OpenAI
- Anthropic
- Mistral