AstraVerge Agent 2.1.0 — руководство по SDK и плагинам

Содержание

1. SDK для языков программирования
2. Разработка плагинов Agent: полное руководство
3. Карта API SDK
4. Локальная разработка и проверка
5. Жизненный цикл плагина
6. Обязательные методы плагина и read-only интерфейсы Setup
7. Установка плагина в общий Agent: контракт для любого продукта
8. Подпись сборки, лицензия покупателя и распространение
9. Подписи плагинов и независимые лицензии
10. Demo-ping 2.1.0: учебный плагин
11. Demo-active 2.1.0: учебный плагин
12. Service Health — проверки для балансировщика
13. Экспортируемые объявления SDK

Руководство рассчитано на разработчика плагина и администратора, подключающего Service Health к HAProxy. SDK 2.1.0 содержит Go-библиотеки и готовый хост IPC для примеров на C, C++, C# (.NET), Java, Python, PHP, Rust, JavaScript и TypeScript. Сборка и запуск описаны в главе SDK для языков программирования. Для Go-примеров используется Go 1.27.4. Публичный транспорт и IPC используют протокол версии 1.

1. Создайте IPC-плагин

1. В распакованном SDK начните с `examples/go`. Выполните `go test ./...` и `go vet ./...`. Пример использует каталог `source` из того же архива.
2. Задайте стабильный ID плагина. `Identity` сообщает ID, версию и операции, `Invoke` принимает запрос, `Close` завершает фоновые задачи и освобождает ресурсы.
3. Реализуйте `CheckPermissions` и `DescribeCompatibility`. Значения совместимости процесса и подписанного паспорта должны совпадать. Для нового плагина задайте минимальную версию Agent 2.1.0 и укажите только проверенный диапазон.
4. Запускайте IPC через `pluginhost.Main`. Используйте подтверждённую идентичность клиента из контекста исполнения и проверяйте доступ к предметному методу в обработчике.
5. Сначала проверьте обработчик отдельно, затем IPC-процесс с тестовым родителем. Интеграцию с работающей службой проверяйте на подписанной сборке с действующей лицензией и разрешениями клиента.

2. Учебные плагины

demo-ping проверяет пассивный маршрут. Авторизованный клиент вызывает `Ping` через REST или gRPC. Agent передаёт запрос в IPC, обработчик возвращает `{"reply":"Pong"}`. Проверьте отдельно, что клиент без разрешения получает отказ.

demo-active показывает исходящие события. Настройте отдельный приёмник и учётные данные. Сначала прочитайте `status`, затем выполните разрешённый `start`. По умолчанию плагин ставит событие `{"status":"ok"}` в очередь раз в пять секунд. Проверьте, что приёмник сохранил событие и подтвердил запись. Завершите проверку вызовом `stop`.

3. Контракты SDK

Пакет	Назначение
`pluginhost`, `pluginipc`	Запуск плагина, IPC, идентичность клиента и обработка запросов
`client`, `api`	Клиентские операции и типы публичного API
`active`	Брокер, активные команды и доставка событий

Пакет	Назначение
`admin`	Локальный административный клиент и типы операций

Для сырых входов реализуйте `pluginipc.IngressHandler` и настройте `socket_inputs` в Agent. Для потокового содержимого согласуйте поддержку transfer API и лимиты. Для staged activation подготовьте кандидата без внешних эффектов и отложите прикладные фоновые действия до активации или авторизованного вызова. При потере ответа после операции с побочным эффектом сверяйте предметное состояние по ID команды.

4. Подпись и установка

Соберите executable под целевые платформы, зафиксируйте SHA-256 и совместимость. Для выпуска по новой схеме нужны сертификат разработчика версии 2, подписанный паспорт экземпляра сборки и отметка времени AV-Guard. Закрытый ключ хранится у разработчика. Выдача лицензии пользователю и разрешения на вызов настраиваются отдельно. Сценарий установки и диагностика описаны в руководстве администратора.

5. Настройте Service Health для HAProxy

1. Получите подписанную сборку Service Health 1.0.0 для своей платформы и лицензию AstraVerge. Добавьте плагин в `plugins` Agent с ID `com.astraverge.service-health`.
2. В режиме `passive` или `both` привяжите вход `socket_inputs` к TCP-адресу, доступному HAProxy. Ограничьте разрешённые адреса отправителей и задайте SHA-256 ASCII-строки токена. Токен состоит из 64 шестнадцатеричных ASCII-символов.
3. В настройках плагина укажите те же `network` и `address`, ID клиента и продукта, HTTP-точку или процесс. При указании двух проверок готовность требует успешного ответа обеих.
4. На стороне HAProxy включите `agent-check`, укажите адрес и порт Agent и передайте исходный токен через `agent-send`. Пример ниже предполагает, что HAProxy и Agent работают на одном узле.

```
socket_inputs:
  - network: tcp
    address: 127.0.0.1:9230
    plugin: com.astraverge.service-health
    clients:
      - id: haproxy
        product: service-monitoring
        token_sha256: "<SHA-256 ASCII-строки токена>"
        allowed_networks: ["127.0.0.1/32"]
```

```
{
  "allowed_principals": ["haproxy"],
  "allowed_products": ["service-monitoring"],
  "checks": [
    {
      "network": "tcp",
      "address": "127.0.0.1:9230",
      "timeout_ms": 800,
      "http": {"url": "http://127.0.0.1:8080/healthz", "status": 200},
      "weight_degraded": 25
    }
  ]
}
```

Рабочая проверка возвращает `100% ready`. При отказе ответ содержит `drain fail` и причину, например `#http_503` или `#process_not_found`. Проверьте сценарий на тестовом backend: сначала убедитесь, что HAProxy видит готовность, затем измените ответ health endpoint и проверьте переход в состояние отказа. После восстановления проверьте возврат готовности.

SDK для языков программирования

Все примеры реализуют плагин `com.astraverge.demo-ping` с вызовом `Ping` и ответом `{"reply":"Pong"}`. Собирайте пример на целевой ОС и архитектуре.

Сборка

Выберите язык и установите инструменты из его строки. Для C, C++, .NET, Java, Python, PHP, Rust, JavaScript и TypeScript используется готовый `agent-plugin-build` из каталога `tools/<платформа>`. Go-компилятор и другие языки для этих примеров не требуются.

Аргумент	Пример	Дополнительные инструменты
`go`	`examples/go/main.go`	Go 1.27.4
`c`	`examples/c/ping.c`	GCC или совместимый C17-компилятор, `CC`
`cpp`	`examples/cpp/ping.cpp`	G++ или совместимый C++17-компилятор, `CXX`
`dotnet`	`examples/dotnet/Program.cs`	C# и SDK .NET 10 и среда выполнения .NET 10
`java`	`examples/java/Ping.java`	JDK 17+ для сборки JAR и Java 17+ для запуска
`python`	`examples/python/ping.py`	Python 3.12+
`php`	`examples/php/ping.php`	PHP CLI 8+
`rust`	`examples/rust/ping.rs`	`rustc` со штатным линкером
`javascript`	`examples/javascript/ping.mjs`	Node.js 24+
`typescript`	`examples/typescript/ping.mts`	Node.js 24+, встроенная обработка синтаксиса типов

Например, Java на Windows, из корня распакованного комплекта:

```
.\tools\windows-amd64\agent-plugin-build.exe -language java -output demo-ping.exe
```

Например, Rust на Linux:

```
./tools/linux-amd64/agent-plugin-build -language rust -output demo-ping
```

Для Go из `examples/go` выполните `go build -o demo-ping .`. На Windows используйте `demo-ping.exe`. Для первой сборки Go нужны публичные модули `go-winio` и `x/sys` или их локальный кэш.

Укажите `-id` для своего пространства имён в сертификате разработчика. Для Go измените `pluginID` в исходнике или передайте `-ldflags="-X main.pluginID=com.astraverge.example.ping"`. Сборщик фиксирует абсолютный путь доверенной среды выполнения. При необходимости укажите его параметром `-runtime`. Выберите каталог исходников параметром `-source`, если скопировали пример в другой проект. Для публикации по нескольким ОС подготовьте отдельные сборки и паспорта.

Как устроены примеры

У Go-примера обработчик находится в исполняемом файле. Для остальных языков готовый сборщик встраивает скрипт, JAR, сборку .NET или нативный обработчик в платформенный хост IPC. Выходной исполняемый файл уже содержит оба компонента. При разрешённом вызове адаптер извлекает встроенные байты в приватный временный каталог и запускает фиксированную команду. Пример обрабатывает один `Ping` за процесс. Результат возвращается из stdout как JSON. К входному запросу через командную строку или stdin обработчик примера доступа не получает. Путь к программе и её аргументы выбираются при сборке. Среду выполнения и её библиотеки устанавливает администратор как доверенные системные зависимости.

Общий хост реализует рукопожатие, lease, управление, staged-активацию и проверку доверенной идентичности. Обработчик запускается только после разрешения операции. Если агент разрывает lease или отменяет вызов, хост отменяет выполняющийся обработчик. Для своих долгоживущих обработчиков реализуйте управление состоянием и ресурсами внутри плагина.

Подпись и запуск

1. Скопируйте `build.example.json` в `build.json`. Укажите выданный `developer_id`, ID сборки, ОС и архитектуру. При изменении контракта одинаково обновите данные в исполняемом файле и паспорте. Для Linux используйте `linux`, для FreeBSD — `freebsd`, для macOS — `darwin`.

2. Выберите инструменты из `tools/<платформа>` для компьютера, на котором подписываете. Ключ разработчика храните вне архива и каталога проекта. Запускайте подпись из защищённого рабочего каталога. Закройте доступ посторонним к каталогу назначения и файлу ключа средствами ОС.

   ```
   .\tools\windows-amd64\make_sign.exe --key <путь-к-ключу> --request build.json --executable demo-ping.exe --output
   plugin.sig
   ```

   На Unix используйте `./tools/<платформа>/make_sign`. На macOS сборщик применяет системный `codesign` до выпуска паспорта. Любое изменение исполняемого файла после выпуска паспорта требует нового паспорта. Для сертификата версии 2 получите в AV-Guard отметку для точных байтов паспорта и сертификата. Передайте администратору бинарник, настройки, сертификат, паспорт, отметку и подписанный пакет доверия. Контракт доверия

3. Добавьте плагин в конфигурацию тестового Agent. Для сертификата версии 2 пример секции выглядит так:

   ```
   plugins:
     com.astraverge.demo-ping:
       enabled: true
       executable: plugins/demo-ping.exe
       settings_file: plugins/ping.settings.json
       license: plugins/plugin.avlic
       developer_certificate: plugins/developer.cert
       release_file: plugins/plugin.sig
       build_timestamp: plugins/plugin.timestamp
       build_trust:
         envelope_file: plugins/build-trust.json
         state_directory: state/demo-ping-trust
   ```

4. Разрешите ID плагина тестовому клиенту в конфигурации Agent. Укажите фактические ID клиента и продукта в копии `settings.example.json`. Запустите Agent с этой конфигурацией и вызовите:

   ```
   av-agent-cli call --config client.json com.astraverge.demo-ping Ping
   ```

   Успешный ответ — `{"reply":"Pong"}`. Для установки через административный API используйте management-контракт.

Проверка комплекта

Контрольные суммы всех файлов приведены в `SHA256SUMS`. Для IPC-приёмки разработчик может запустить интеграционный тест `TestNativePingIPC` на собранном бинарнике через переменную `AGENT_PLUGIN_TEST_EXECUTABLE`. Из `examples/go` выполните `go test -run TestNativePingIPC -count=1 .`. Тест использует локальный SDK-стенд, который проверяет авторизацию, Ping, staged-активацию и потерю lease. Регистрацию с сертификатом и паспортом проверяют отдельно через Agent.

Разработка плагинов Agent: полное руководство

Руководство для Agent Plugin SDK 2.1.0. Полный справочник объявлений генерируется из дерева исходников. Он охватывает и публичные пакеты полного модуля Agent. Состав ZIP и подтверждённые проверки приведены в описании SDK.

1. Архитектура

Плагин — отдельный локальный executable. Agent аутентифицирует подключение клиента, выбирает разрешённый plugin ID, передаёт вызов через IPC и возвращает результат. Код плагина не загружается в адресное пространство ядра как DLL/.so. Процесс наследует учётную запись Agent, а не получает автоматически root/LocalSystem.

Методы, ресурсы, scopes, предметные права, задания и транзакции принадлежат плагину. Это позволяет выпускать независимые решения без добавления предметных collectors в ядро. Отдельный процесс не является sandbox для недоверенного кода: подпись устанавливает происхождение, не безопасность.

Для активного режима есть серверный SDK Broker/Exchange, durable inbox/outbox и доставка событий. Для общей установки — новый локальный административный контракт с OS-авторизацией. Это разные слои; административная операция не равна предметному job.

2. Пакеты

Пакет	Ответственность
pkg/pluginipc	Handler/Identity/Response, framing, клиент, lease, Execution, stream, ingress
pkg/pluginhost	Plugin, стандартный Main, Close, StagedPlugin, ConfigJournal
pkg/api	TransportVersion, конверты вызовов, DTO ресурсов/заданий/конфигурации
pkg/client	Ограниченный HTTP/HTTPS клиент вызовов и reload
pkg/active	REST/gRPC, Broker, Publisher, Inbox, transfer, credentials/failover
pkg/admin	Новый локальный контракт управления общей установкой
pkg/providerjobs	Сохраняемые задания внутри provider-процесса
pkg/providerpolicy	Точное сопоставление grants с доверенной identity
pkg/pluginipc/plugintest	Conformance-тесты; импортируется только из _test.go
pkg/keychain	Платформенный доступ к ключам, не универсальная библиотека секретов

Внешний плагин не импортирует internal ядра. Публичные пакеты SDK могут иметь внутренние зависимости внутри своего модуля; это учитывается при подготовке SDK-only дистрибутива. Исторические предметные DTO не означают, что соответствующие endpoints существуют в ядре.

3. Среда разработки

Используйте установленный Go 1.27.4 и `GOTOOLCHAIN=local`. Для native-функций дополнительно нужны API/SDK соответствующей ОС и зависимости плагина. Тестовый IPC-родитель запускается без установки службы.

Текущий путь импорта SDK — `astraverge.local/av-agent/v2`, версия Go-модуля — `v2.1.0`. Для сборки используется локальный checkout, исходники из SDK-архива или настроенный Go proxy. ID плагинов используют пространство `com.astraverge`.

SDK 2.1.0 включает публичные Go-компоненты, примеры, IPC-обвязку для обработчиков на других языках и инструменты подписи AV-Guard.

4. Новый проект

Публикуемые учебные исходники лежат в examples/demo-ping и examples/demo-active внутри этого комплекта. Они происходят из реальных одноимённых модулей каталога AV-Agent-Providers. Собственная module identity комплекта — `com.astraverge/av-agent-public-examples`.

В распакованном SDK каталог `examples/go` подключает `../../source`, который находится в том же архиве.

```
cd doc_market/examples
go mod tidy
go build ./demo-ping ./demo-active
```

Это команды для настоящих каталогов, не создание третьего произвольного demo вместо двух обязательных. Для собственного продукта скопируйте подходящую структуру, задайте свой module/ID и закрепите версию SDK.

5. Реализация плагина

Реальные базовые интерфейсы:

```
type Handler interface {
    Identity() Identity
    Invoke(context.Context, string, json.RawMessage) Response
}

type Plugin interface {
    pluginipc.Handler
```

```
    Close() error
}
```

Полный короткий исходник представлен в учебнике Ping/Pong. Его Main вызывает pluginhost.Main с фиксированным ID и фабрикой настроек. Invoke возвращает Body или Problem. Close обязан освободить фоновые ресурсы вашего плагина, даже если в минимальном примере он пустой.

ID в executable, Identity, launch config, конфигурации Agent и signed build должны совпадать. Переименование только YAML не превращает чужую коммерческую сборку в ваш плагин.

6. IPC

IPC v1 использует Unix socket либо Windows named pipe. Framing имеет 24-байтовый big-endian заголовок, magic AVPL, protocol version, kind, request ID, длину и reserved bytes. Обычные сообщения передают JSON. Авторизация private-соединения использует случайный launch token; он не берётся из публичного тела запроса.

pluginhost.Main принимает существующие флаги `-endpoint` и `-launch`. Agent создаёт закрытый launch JSON; токен не передаётся в аргументах и не включается в дистрибутив. Поддерживается parent lease, а при его потере plugin SDK прекращает обслуживание. Зарезервированные __lease/__shutdown/__activate не доступны обычному публичному вызову.

Реализация IPC на другом языке возможна по протоколу, но готового равнозначного pluginhost/plugintest для всех языков проект не предоставляет. Наличие REST/gRPC для клиентов не делает HTTP-приложение готовым IPC-плагином.

7. Объявление возможностей

Identity содержит ID, Version, SchemaVersion и Operations; SDK сообщает IPC protocol. Список Operations описывает функции для клиентов, но ядро не превращает его в предметный ACL и не блокирует все неизвестные методы автоматически. Обработчик сам возвращает unsupported_method.

Streaming capabilities SDK объявляет по фактическому наличию StreamHandler, а не только по строкам Identity. В новой staged-обёртке StagedActivation подтверждает поддержку подготовленного запуска; это не обещание безопасной миграции внешних данных.

В клиентских вызовах используйте точные ID плагинов, включая точки.

8. Запросы, identity и права

Публичный HTTP-конверт:

```
{"method":"Ping","body":null}
```

Клиент не задаёт principal через этот конверт. Agent передаёт Execution отдельно. Внутри Invoke используйте настоящий `pluginipc.ExecutionFromContext(ctx)`, затем проверьте PrincipalID/Product и предметные grants. Поля с такими же именами внутри body остаются недоверенными данными.

Метод — до 128 печатных ASCII-символов, без пробелов и префикса __. Схему body, пути, IDs ресурсов и допустимость операции проверяет плагин. Контекст содержит deadline/cancellation; он не откатывает уже выполненную мутацию. Сохранённый RequestID полезен для собственной идемпотентности, но не является автоматически job ID.

Минимальные demo намеренно не реализуют полный бизнес-ACL. Не переносите их отсутствие предметных проверок в плагин с чувствительными действиями.

9. Ответы и передача данных

```
return pluginipc.Response{Body: json.RawMessage(`{"reply":"Pong"}`)}
```

Это реальная строка из demo-ping. Ответ на выполненный вызов передаётся в pluginipc.Response; предметная ошибка может находиться в Error при HTTP 200. Go client.Call возвращает и response, и ошибку *pluginipc.Problem. HTTPError относится к транспортному HTTP-отказу.

Обычный IPC request ограничен 1 MiB после кодирования конверта, inline response — 256 MiB. Настраиваемый HTTP-предел может быть ниже. Не считайте эти значения лимитами собственного batch/body до сериализации. Для больших данных

используйте transfer/stream API, не увеличивайте один JSON до произвольного размера.

10. Streaming

Реальный дополнительный интерфейс:

```
type StreamHandler interface {
    InvokeStream(context.Context, string, io.Reader, io.Writer) error
}
```

Обрабатывайте input/output последовательно с ограниченным буфером; не вызывайте io.ReadAll на потенциально огромном входе. CallStreamSized заранее проверяет известный размер; превышение неизвестного потока может произойти уже после частичной предметной работы. Частичный output не становится успешным результатом при ошибке.

Для внешнего транспорта сначала создаётся и завершается проверенный transfer, затем StreamCall с method/transfer_id/request_id. Output сохраняется как новый transfer и скачивается отдельно. В SDK есть TransferStore, StoreTransfer, UploadTransfer/DownloadTransfer и AgentTransferClient. Их точные сигнатуры приведены в полном API reference, а каноническая семантика — docs/LARGE-TRANSFERS.md.

Максимум логического тела — конфигурируемый 1_000_000_000_000_000 байт, кадр/часть ограничены. Это не результат реальной передачи 1 ПБ и не гарантия throughput. Для VNC такая внешняя request/result-схема ещё не является интерактивным каналом.

В активном режиме резерв перед вызовом покрывает сохранение результата и переходы квитанции. Агент передаёт сохранённый результат и может завершить подтверждение независимо от очередного `Poll`. При неопределённом исходе используйте предметную сверку. Состояние потока и подтверждения.

Сокетный ingress

Реализуйте `pluginipc.IngressHandler`, если плагину нужны сырые входные данные из `socket_inputs`. Agent открывает слушатель, проверяет выбранный способ аутентификации и передаёт поток или датаграмму с `pluginipc.Ingress`. Подтверждённые ID клиента находятся в `pluginipc.Execution`. Для явно анонимного входа `Ingress.Anonymous` установлен, а ID и продукт пусты. UDP-ответ ограничен размером входной датаграммы. Для `unixgram` ответ отправителю отсутствует. Контракт socket ingress · Пример Service Health.

11. Длительные задания

Ядро не владеет предметными jobs. Если они нужны, используйте providerjobs внутри плагина. Queue.OpenWithOptions принимает Handler с Invoke и Authorized. Реальные внутренние методы обработчика — prepare (без побочных эффектов, возвращает нормализованный api.JobRequest) и execute (одна попытка, api.JobOutput).

Наружу provider-owned Queue предоставляет jobs.submit, jobs.lookup, jobs.status, jobs.cancel, jobs.result через Invoke. JobRequest включает Module, Operation, Scope, Resource, Parameters, TimeoutSeconds и IdempotencyKey. Авторизация выполняется по доверенной identity и проверяется повторно перед исполнением.

Один idempotency key с другим нормализованным запросом даёт conflict. После restart незавершённая работа не запускается вслепую заново; состояние отражает неизвестный исход. Payload результатов имеет retention/квоты, компактные свидетельства идемпотентности сохраняются. Close ожидает завершения принятых handlers, поэтому они обязаны соблюдать отмену.

Для точных grants есть providerpolicy.Authorized(ctx, policies, module, operation, scope, resource). Он не создаёт identity store и не заменяет проверку native-владения целевым объектом.

12. Ошибки и диагностика

Возвращайте стабильный Problem.Code, понятное безопасное Message и не включайте токены/ключи. Неизвестный метод — unsupported_method. Отказ аутентификации/доступа Agent и предметный отказ плагина — разные причины.

При потере IPC-ответа возможен outcome_unknown: вызов не повторяется автоматически. Не преобразуйте его в «не выполнялось». Для мутаций сначала сверяйте фактический предметный результат по своему ключу/идентификатору.

Agent предоставляет защищённые status/health/metrics, состояние процессов и статистику очередей. Они не гарантируют работоспособность внешнего устройства или R Runtime. Используйте собственную операцию диагностики, если она нужна

продукту. stderr должен содержать диагностику, а не секреты; stdout не является IPC-каналом.

13. Конфигурация

Основной YAML определяет transport/auth, principals, plugins и инфраструктурные лимиты. settings_file — отдельный JSON, смысл которого принадлежит плагину. Agent проверяет синтаксис конверта, не изобретает предметную валидацию.

Статическая регистрация требует `executable`, `developer_certificate`, `release_file` и `settings_file`. Для сертификата версии 2 добавьте `build_timestamp` и `build_trust`. Для политики `required` укажите `license`, а для лицензии, подписанной ключом разработчика, также `license_delegation`. В административном пути bundle копируется в защищённый staging, а его регистрация хранится отдельно от YAML. Нельзя одновременно считать один ID статическим и административно управляемым.

При строгой предметной схеме используйте DisallowUnknownFields и проверку единственного JSON-документа в фабрике. Demo-active исторически использует json.Unmarshal и не является примером полного hardened-конфига. JSON может содержать абсолютные пути к собственному state, но не должен включать реальный production-credential в публичный пример.

14. Запуск и остановка

В production executable запускает Agent после проверки доверия. Удалённая загрузка произвольного кода не выполняется. При старте проверяются Identity/IPC/schema; lease связывает срок жизни plugin-процесса с родителем.

Close останавливает scheduler, отменяет контексты, ожидает goroutines и закрывает exchange/outbox/native-ресурсы. Demo-active показывает эту последовательность. Нельзя объявлять остановку успешной только после отправки сигнала.

Supervisor может восстановить процесс для следующих запросов, но не повторяет неизвестную предметную операцию. Reload существующего статического процесса не следует считать транзакцией изменения его settings_file. Новый административный update использует отдельного staged-кандидата и поколения регистрации.

15. Локальные тесты

Начните с Invoke и фабрики, затем используйте plugintest.Run. Он проверяет identity, framing, credential, cancellation, disconnect, ошибки/panic и lease на реальном Handler в тестовом процессе. Его probes существуют только в decorator теста.

В данном комплекте добавлен private test parent, который собирает реальный executable, создаёт временные launch/token, проверяет describe, метод, неверный credential и shutdown/потерю lease. Это не запуск unsigned плагина через production Agent. Штатный Developer Mode пока отсутствует; инструкция не предлагает несуществующий флаг обхода.

```
cd doc_market/examples
go test -count=1 -timeout=120s ./...
```

Подробности среды и результатов — LOCAL-TESTING и EXAMPLES-VALIDATION. Полноценная production-регистрация примеров блокируется отсутствием выданных сертификатов нового формата; оба исправленных demo реализуют StagedPlugin, но это не заменяет проверку подписи.

16. Сборка разных ОС

Собирайте отдельный executable на пару GOOS/GOARCH. При CGO_ENABLED=0 чистый Go-путь переносится проще; native-библиотеки могут требовать собственную toolchain и проверку на целевой ОС. Windows использует .exe, Unix — исполняемый файл с нужными правами.

Пример целевых переменных для POSIX shell:

```
GOTOOLCHAIN=local CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -o /tmp/demo-ping-linux ./demo-ping
```

В PowerShell задавайте env-переменные процессу и восстанавливайте их после команды. Не считайте собранный Linux/FreeBSD/macOS файл нативно проверенным. Сопоставляйте платформу в подписанном паспорте с фактической сборкой.

17. Подпись release

Утверждённая Developer Program предусматривает персональный закрытый ключ от AstraVerge и публичный developer.cert. Перед распространением разработчик подписывает отдельный паспорт для каждой пары ОС и архитектуры и получает отметку

AV-Guard. Паспорт включает `format`, `plugin_id`, `developer_id`, `version`, `os`, `arch`, `ipc_protocol`, `compatibility`, `sha256` и `license_policy`.

SDK включает утилиты `make_sign` и `verify_sign` из AV-Guard. Для рабочего выпуска нужны выданный сертификат, штатная отметка времени и пакет доверия. Порядок подписания.

Закрытый ключ не входит в bundle, не пишется в argv, логи или примеры. Подписывается финальный executable после сборки; последующее изменение байтов требует новой подписи.

18. Проверка подписи Agent

Agent проверяет встроенным корнем сертификат, извлекает ключ разработчика, проверяет подписанный паспорт, права сертификата на пространство plugin/product ID, текущую платформу, IPC и хеш точных запускаемых байтов. Любая ошибка запрещает новый запуск.

Подписанная политика license_policy обязательна. Режим required требует отдельной действительной лицензии с точными привязками для каждого плагина. Изменение YAML не может ослабить подписанную политику. Лицензия и trust high-water state сохраняются, включая административные обновления; candidate использует частную копию state до commit.

Проверка локальная. Срок Developer Program не является online-switch для уже выпущенных сборок. При проверке Agent использует доверенные документы, установленные локально.

19. Совместимость

Различайте версию executable Agent, HTTP API v1, IPC v1, schema_version Identity, streaming_version и форматы сертификата/паспорта/admin. Совпадение номера Agent не означает наличия новой capability в старом артефакте.

Закрепляйте SDK и тестируйте своё поведение. Заявленная цель обратной бинарной/поведенческой совместимости не доказывает совместимость всех будущих сборок. Старые подписанные executable могут требовать нового паспорта/сертификата при переходе с прежней лицензионной модели без пересборки самого кода; staged-регистрация дополнительно требует поддержки SDK-интерфейса.

20. Распространение и обновление

Распространяйте executable, публичный сертификат, подписанный паспорт, документацию и разрешённые настройки; коммерческую лицензию заказчика передавайте отдельно и защищённо, если она нужна. Ваши договоры и цена независимы от Developer Program.

Для общей службы используйте новый административный сценарий. Сохраните operation ID до отправки, проверьте permissions/поколения, дождитесь client proof и конечного succeeded. Не редактируйте общий YAML из установщика продукта. Не удаляйте чужие плагины или общую службу при деинсталляции.

Обновление не сбрасывает jobs/cursors/state ради новой версии. Предметные миграции выполняются владельцем плагина с проверкой совместимости. Поддержку клиента, ОС, runtime и повторную диагностику после неизвестного исхода планируйте как часть собственного продукта.

Обязательный контракт SDK

Все плагины реализуют CheckPermissions и DescribeCompatibility из pluginipc.ManagementHandler. pluginhost.Plugin включает этот интерфейс: прежняя реализация без методов не компилируется с новым SDK. Полный контракт содержит точные поля compatibility, методы OS-peer API, команды CLI и правила denied/unknown. Нет исключений для бесплатных/customer-плагинов.

CheckPermissions должен использовать ту же политику, что предметные вызовы, без их исполнения. В обоих demo авторизованный start/Ping можно проверить без активации и без изменения outbox. DescribeCompatibility возвращает подписываемые требования к Agent/API/IPC/capabilities и схемам состояния/настроек. До exec проверяется паспорт v2, при handshake — фактические методы и совпадение report.

Карта API SDK

Далее приведена карта API SDK 2.1.0. Сигнатуры и типы описаны в справочнике объявлений.

Плагин и локальный IPC

`pluginipc.Handler`: Identity и Invoke. `pluginhost.Plugin` добавляет Close. `pluginhost.Main(id, open)` — стандартный executable entry point. Фабрика получает json.RawMessage настроек и возвращает Plugin/error.

`pluginipc.Identity`: стабильный ID, версия сборки, IPC Protocol, SchemaVersion, Operations, optional ResourceOperations, streaming caps и новый StagedActivation. Protocol и stream caps окончательно формируются SDK. Operations не является ACL ядра.

`pluginipc.Response`: Body и Error. `pluginipc.Problem`: Code/Message, реализует error. Body — raw JSON; ошибка может передаваться вместе с body.

`pluginipc.Config`: private ID/Token/Settings/Concurrency/StreamLimits и новый Staged. Это launch-файл родителя, не публичное тело запроса. `LoadConfig` читает его с проверками; `Endpoint` строит адрес локального транспорта; `SecureDirectory` защищает каталог.

`NewClient`/`NewClientWithStreamLimits`, `Client.Call`, `CallStream`, `CallStreamSized`, `Close` — private IPC-клиент. `ServePlugin`/`ServePluginWithStreamLimits` требуют parent lease; `Serve` предназначен для иной локальной модели без требования lease. Production-плагин использует pluginhost.Main, а не анонимный listener.

`ExecutionFromContext` возвращает RequestID, JobID, PrincipalID, Product. `WithExecution` нужен доверенной стороне/тестовой оснастке. Клиентское JSON-поле не становится доверенным через совпадение имени.

Подготовленный запуск

`pluginhost.StagedPlugin` добавляет Activate(context.Context) error. Плагин обязуется не выполнять внешние предметные действия в конструкторе/подготовке и изолировать state кандидата. Обёртка SDK закрывает предметные Invoke/stream до активации и не рекламирует streaming для обычного Handler.

Это новое требование административной регистрации, не обязательное изменение всех старых статических плагинов. Оно не является sandbox и не обеспечивает обратимость произвольного внешнего действия.

Публичные вызовы

`client.New` создаёт HTTPS/mTLS-клиент, `NewHTTP` — только явно выбранный HTTP/Bearer. `Client.Call(ctx, plugin, method, body)` делает один вызов, `Plugins` получает разрешённые ID, `ReloadPlugin` управляет отдельно разрешённым процессом, `Close` освобождает соединения. Ошибки: *pluginipc.Problem для предметного ответа, *client.HTTPError для HTTP-отказа, транспортная ошибка при недоступности ответа.

`api.PluginCall` содержит Method/Body; `api.StreamCall` — Method/TransferID/RequestID. `api.ValidPluginMethod` отклоняет __ и недопустимые методы. Новый `api.ValidPluginID` поддерживает точные dot-separated ID; нет wildcard или path separators.

Активный транспорт и события

`active.Exchange`: Hello/Poll/Complete/Publish/Heartbeat/PublishBatch/Close. `NewExchange(protocol, endpoint, security)` создаёт REST/gRPC-реализацию. Security описывает file paths к CA/certificate/private key/Bearer; credentials не кодируются в Event.Body.

`NewBroker`/`NewBrokerWithConfig`, Submit/Wait/Execute, Complete/Events/Close — SDK принимающей стороны. `RESTHandler` и `RegisterGRPC` подключают реальный transport. Это не готовый пользовательский портал.

`OpenPublisher`/`OpenPublisherWithConfig`, Publisher.Enqueue/Flush/QueueStats/Close — outbox внутри плагина. Flush передаёт накопленные события и обрабатывает ACK; это не повторное выполнение предметной операции. Demo-active использует именно этот механизм и собственный scheduler.

`QueueConfig` задаёт размеры/количество, batch, rate, jitter, concurrency, retention и maintenance. `QuotaError` означает непринятые/неподтверждённые данные, не разрешение отбросить их. `ErrOutcomeUnknown`, `ErrResultExpired` и tombstones имеют разные значения.

Большие тела

`TransferConfig`/`TransferStore` предоставляют Begin/Append/Status/Finish/Open/Confirm и квоты/retention. `TransferRequest` передаёт ограниченную часть, а не целое логическое тело. `TransferWriter` сохраняет output постепенно; незавершённый output не пересоздаётся повторным исполнением плагина.

`AgentTransferClient` выполняет Transfer/Stream/Call через аутентифицированный passive REST/gRPC. `TransferExchange` — опциональный transport interface; отсутствие возможности отклоняется до неподдерживаемого вызова. `Publisher.EnableTransfers`/`EnqueueStream` подключают большие event bodies. Прямой локальный IPC stream и внешний transfer-based вызов — разные уровни.

Задания и предметные grants

`providerjobs.Handler`: Invoke и Authorized. `Open`/`OpenWithOptions` создают provider-owned Queue; `Queue.Invoke` маршрутизирует jobs.submit/lookup/status/cancel/result. `Options` ограничивает records/state/concurrency/timeouts/retention/maintenance. Close ждёт принятых обработчиков.

`api.JobRequest`, JobSubmission, Job, JobOutput и ResultPage — DTO. Наличие этих DTO не означает ядровой HTTP `/jobs`. Внутренний handler Queue вызывает предметные prepare/execute; метод execute не должен повторяться после неопределённого исхода без предметной сверки.

`providerpolicy.Policy` связывает PrincipalID/Product со списком api.Grant. `Authorized` использует доверенный Execution, точные Module/Operation/Scope/Resource. Чужой объект не становится вашим только из-за прохождения grant matcher.

Конфигурационные журналы

`pluginhost.ConfigJournal`, OpenConfigJournal/View/Update/Close и ConfigRecord/ConfigJournalData — средства хранения состояния внутри плагина. `ConfigTransition` и `ValidConfigResult` проверяют общую форму переходов. Ядро не исполняет предметную конфигурационную транзакцию за ваш Handler.

Административный SDK

`admin.Contract`, Installation/Endpoint/Request/Requirements/Files/LicenseBinding/Consumer/Credential/Call/Reply/Operation/Result/Failure/Registration — реальные DTO нового контракта. `ReadInstallation` читает descriptor, `Discover` проверяет живой OS peer, `Send` выполняет одну локальную административную команду. `DecodeRequest` строго проверяет JSON-запрос; ответы допускают дополнительные информационные поля.

Административные методы: discover, inspect, prepare, apply, status. Права проверяются сервером по OS-идентичности, а не наличию экземпляра struct Request. Client proof выполняется обычным приложением через отдельный заголовок публичного HTTP/HTTPS-вызова, не Send от имени установщика.

Тестовая оснастка

`plugintest.Run(t, factory, Options)` проверяет Handler через реальный SDK. Options содержит ID/Method/Body/Execution/Check. Пакет должен оставаться в _test.go. SDK `plugintest.StartExecutable(t, ExecutableOptions)` собирает и запускает локальный executable с private launch/token/lease; `CheckIdentityAndAuth` и `Stop` проверяют handshake и завершение. Это тестовая оснастка, не Developer Mode Agent.

Платформенные варианты

Справочник включает объявления из файлов разных ОС. Их нельзя одновременно скопировать в один исходник; выбор делает Go build system. Native transport, peer credentials и filesystem guarantees требуют проверки соответствующей ОС. Для полного перечня типов/полей и точных сигнатур используйте сгенерированный справочник и ссылки на реальные исходные файлы.

Обязательный контракт SDK

Все плагины реализуют CheckPermissions и DescribeCompatibility из pluginipc.ManagementHandler. pluginhost.Plugin включает этот интерфейс: прежняя реализация без методов не компилируется с новым SDK. Полный контракт содержит точные поля compatibility, методы OS-peer API, команды CLI и правила denied/unknown. Нет исключений для бесплатных/customer-плагинов.

CheckPermissions должен использовать ту же политику, что предметные вызовы, без их исполнения. В обоих demo авторизованный start/Ping можно проверить без активации и без изменения outbox. DescribeCompatibility возвращает подписываемые требования к Agent/API/IPC/capabilities и схемам состояния/настроек. До exec проверяется паспорт v2, при handshake — фактические методы и совпадение report.

Локальная разработка и проверка

Что доступно сейчас

Unit-тесты Handler, conformance через plugintest и запуск отдельного demo-executable private тестовым родителем. Это позволяет разрабатывать функцию без покупки подписки и без выдачи рабочего developer certificate.

Проверяйте обработчик и IPC в тестовом процессе. Для запуска плагина в службе Agent требуется подписанная сборка.

Воспроизведение данного комплекта

В архиве SDK 2.1.0 перейдите в `examples/go` и выполните `go test ./...` и `go vet ./...`. Пример использует распакованный `source/`.

Компиляция двух main-пакетов без -o в одной команде проверяет сборку, не размещая executable в каталоге исходников. Тестовый родитель собирает отдельные executable во временных каталогах.

Что проверяют тесты

Demo-ping: фактический Ping/Pong, unsupported_method, SDK conformance, настоящий executable, describe/ID/schema/protocol, неверный IPC credential, корректный __shutdown и завершение по потере parent lease.

Demo-active: отказ неверным настройкам, настоящий executable и status, неизвестный метод, доставка события с постоянным ID на loopback-приёмник с явным Bearer, завершение scheduler и освобождение outbox lock. Секреты создаются crypto/rand во временном каталоге и не выводятся в пример.

Запуск активного события происходит самим плагином через active.Exchange/Publisher, а не через скрытый предметный scheduler в ядре. Тестовый приёмник — SDK Broker с реальным HTTP-handler, не подмена успешного ACK строкой.

Что эти тесты не доказывают

Они не регистрируют неподписанный demo в production Agent, не выпускают developer.cert от AstraVerge, не устанавливают службу и не проверяют полноту прикладных grants. Describe-регистрация с тестовым IPC-родителем — проверка handshake, а не административная registration в системной установке.

Полноценный административный сценарий самого Agent отдельно проверен в internal/agent на Windows с временными подписанными test fixtures. Их ephemeral root подставлен только внутри _test.go процесса, не в пользовательской конфигурации executable. Это не публичный signing service.

Платформенная матрица

Native-тесты выполняются только на фактической ОС запуска. Дополнительные GOOS/GOARCH build/cross-test-compile проверяют сборку, не поведение. Для публикации заявленной совместимости вашего плагина нужен отдельный finite native-план по используемым ОС/API.

В Windows private IPC/ACL-тесты могут требовать запуска тестовой кампании вне ограниченной песочницы. Полученный Access is denied сначала классифицируется как инфраструктурный отказ; production-проверки безопасности ради него не отключаются.

План будущего Developer Mode

Нужны явное opt-in включение, отдельный technical instance, только локальный доступ, разграничение state/endpoint от основной службы, запрет выдавать dev-экземпляр за production и отсутствие распространения unsigned production-пакетов. Режим не должен изменять встроенные корни системного Agent или позволять сетевому клиенту включить обход.

Это требования, не реализованные флаги. Пока используйте описанные тесты и не переименовывайте их в успешную production-установку.

Результаты текущей конечной кампании — EXAMPLES-VALIDATION.

Жизненный цикл плагина

Статический запуск

Администратор включает доверенный executable в YAML. Перед стартом Agent проверяет сертификат разработчика, подписанный паспорт, пространство ID, хеш и требуемую независимую лицензию. Затем копирует executable в private staging, проверяет точные байты повторно, создаёт launch config и запускает процесс с endpoint/launch path.

pluginhost.Main читает launch config, сверяет ID, создаёт Handler и начинает SDK IPC-сервер. Agent проверяет describe/Identity и открывает parent lease. Неудачная инициализация не становится успешным запуском. Коды предметных методов не передаются через shell.

Служебные methods describe/__lease/__shutdown обрабатываются транспортом; __activate обслуживает новую staged-обёртку. Public API отклоняет префикс __.

Принятый вызов

Agent проверяет подключение/доступ к plugin ID и инфраструктурные лимиты, передаёт доверенный Execution и тело. Плагин проверяет свои предметные права и выполняет одну попытку. При разрыве IPC операция автоматически не повторяется; отсутствие ответа означает возможный неопределённый результат.

Количество принятых вызовов, draining, deadlines и ограничение очереди влияют на lifecycle. Отмена контекста — сигнал, который обработчик должен соблюдать. Она не отменяет внешний эффект задним числом.

Подготовленный административный кандидат

Все новые managed-регистрации требуют StagedPlugin. Его конструктор и Activate не должны выполнять внешние предметные изменения; state кандидата должен быть изолирован от работающего поколения. Agent проверяет фактическую Identity.Version по подписанному паспорту.

До активации SDK-обёртка разрешает служебную проверку describe, но закрывает предметные вызовы. Подключаемое приложение проходит client proof со своими реальными credentials. Наличие owner ID не даёт права пройти этот proof вместо приложения.

После подтверждений Agent проверяет поколения, повторно проверяет доверие/лицензию, активирует кандидата, дренирует старые принятые вызовы и фиксирует новую регистрацию/ACL. Immutable routing snapshot позволяет другим плагинам обслуживаться во время административной подготовки. Старый процесс закрывается, результат становится succeeded после обязательных этапов.

Обратимость ограничена собственным административным состоянием. Фоновый плагин, который начинает внешние отправки уже в конструкторе, не соответствует staged-контракту. Нельзя просто добавить пустой Activate и объявить такой плагин безопасным для транзакционного обновления. Исправленный demo-active 2.1.0 не открывает outbox при подготовке: отдельный авторизованный start запускает работу после завершения установки.

Восстановление

Supervisor может создать новый процесс после аварии, предварительно проверив подпись и право нового запуска. Для ранее активированного staged-процесса новая генерация снова проходит локальную активацию. Оборванные предметные Invoke при этом не переисполняются.

Потеря parent lease завершает SDK-сервер; Close должен остановить и собственные goroutines. Плагин не должен оставлять фоновые workers без управляемого жизненного цикла.

Административный журнал переживает restart. Prepared сохраняется; незавершённый candidate без commit после restart получает отказ вместо автоматического повторения подготовки. Зафиксированный commit восстанавливает новую регистрацию. Неопределённая ошибка storage обозначается recovery_required: нельзя выдавать успех либо гарантированный откат без reconciliation.

Остановка и данные

Close должен быть повторно безопасным по контракту вашего ресурса и не скрывать ошибки завершения. SDK вызывает Close после прекращения обслуживания. Для active-плагина порядок: отменить контекст, дождаться goroutines, закрыть Exchange и Publisher.

Обновление/удаление registration не является purge рабочих данных. Cursors, поколения событий, jobs и результаты не сбрасываются ради новой сборки. Миграции форматов принадлежат плагину. На Unix есть directory fsync в используемом state path; Windows имеет собственные ограничения durability, поэтому внезапное отключение питания не считается доказанным одним atomic rename.

Истечение коммерческой лицензии запрещает новые старты/reload/recovery, но не завершает уже работающий процесс по таймеру. После окончания подписки AV-Guard прекращает выдачу отметок для новых сборок. Ранее выданная отметка по-прежнему подтверждает выпуск в пределах срока сертификата.

Канонические источники: docs/PLUGIN-ADMIN.md, internal/plugins/process.go, pkg/pluginhost/main.go и activation.go, pkg/pluginipc/server.go. Полные ссылки — в матрице доказательств.

Перед каждым запуском обязательны паспорт v2 и management handshake: контракт. Проверки permissions/compatibility доступны до активации и не запускают фоновые задачи.

Обязательные методы плагина и read-only интерфейсы Setup

API транспорта и IPC остаются v1; контракт управления плагином имеет собственную версию 1. Паспорт сборки — v2.

Обязательный интерфейс SDK

`pluginhost.Plugin` теперь включает `pluginipc.ManagementHandler`:

```
CheckPermissions(context.Context, pluginipc.PermissionQuery) (pluginipc.PermissionDecision, error)
DescribeCompatibility(context.Context) (pluginipc.Compatibility, error)
```

Методы обязательны у всех плагинов независимо от цены, владельца и режима установки. В private IPC им соответствуют `permissions.check` и `compatibility.describe`. SDK обрабатывает эти запросы отдельно от Invoke и позволяет их до staged-активации. Capability identity `management_version: 1` формируется по наличию реального Go-интерфейса, а не по строке разработчика.

CheckPermissions проверяет ту же политику, что Invoke/InvokeStream. Субъект берётся из доверенного IPC Execution; query содержит только operation, scope, resource. Поля principal/product в query запрещены. Допустимы allowed, denied, unknown. Отсутствующая идентичность обязана давать denied. Unknown и ошибки никогда не означают разрешение.

Оба метода не создают задания, не открывают изменяемое состояние, не запускают предметные операции, не отправляют внешние команды и не выполняют ленивую инициализацию. Они отвечают и до Activate. Это обязательство доверенного кода, а не sandbox или математическое доказательство отсутствия побочных эффектов.

Требования совместимости

DescribeCompatibility возвращает те же значения, что подписанное поле `compatibility` паспорта `com.astraverge.plugin-build/2`:

```
{
  "management_version":1,
  "min_agent_version":"2.1.0",
  "max_agent_version_exclusive":"3.0.0",
  "api_version":1,
  "ipc_version":1,
  "admin_contract":0,
  "capabilities":[],
  "settings_schema_version":1,
  "state_schema_version":"1"
}
```

Нижняя граница включительна, верхняя исключительна. Версии Agent — три числовых компонента без ведущих нулей и prerelease. `admin_contract: 0` означает отсутствие зависимости от административного listener; это не отключение обязательных management-методов плагина. Значение 1 требует включённого административного контракта. Дополнительные capabilities должны присутствовать у целевого Agent. Схемы settings/state — объявления разработчика; совпадение номера не доказывает безопасность миграции прикладных данных.

Запрет запуска неполного плагина

1. До exec Agent проверяет подпись паспорта v2 и наличие валидных требований. Старый паспорт v1, пропущенный management_version и некорректные требования не допускают запуск executable. Подпись и коммерческая лицензия независимы.
2. До exec проверяется совместимость подписанных требований с текущим Agent.
3. При handshake Agent требует management_version=1, получает фактический compatibility report, сравнивает его с паспортом и проверяет read-only permission probe без идентичности: ответ обязан быть denied.
4. Отсутствующий, ошибочный или несовпадающий ответ не допускает процесс в ready. Кандидат завершается до активации/выдачи бизнес-маршрута.
5. Те же проверки действуют при restart и reload. Нет fallback-флага или исключения для бесплатных, локальных либо customer-плагинов.

Доказать наличие метода в произвольном native executable без его исполнения невозможно. До exec проверяется подписанная декларация; её достоверность сверяется при handshake. SDK дополнительно проверяет собственный интерфейс до активации и открытия listener. Подпись удостоверяет происхождение и целостность сборки. Безопасность кода оценивают отдельно.

Административная проверка эффективных прав

Метод локального OS-peer API: `permissions.check`. Требуется отдельное action `permissions.check` в grants для OS identity, owner ID и namespace плагина. Наличие install/status или consumer.create само по себе его не даёт.

```
{
  "method":"permissions.check",
  "owner_id":"com.astraverge.example-product",
  "plugin_id":"com.astraverge.demo-ping",
  "permissions":{
    "principal_id":"com.astraverge.demo-client",
    "client_address":"127.0.0.1",
    "query":{"operation":"Ping","scope":"","resource":""}
  }
}
```

Agent выбирает principal/product из текущей подтверждённой конфигурации, проверяет routing, manage_plugins и CIDR, затем передаёт разрешённый запрос плагину. Запрос не принимает credentials, не выдаёт grants, не меняет реестр, не создаёт запись операции, не запускает недоступный процесс и не повторяет IPC. Таймаут read-only запроса не завершает работающий плагин.

Ответ содержит decision и отдельные routing, management, network, domain. Management описывает право управления процессом, не предметное разрешение. Общий allowed требует разрешённых routing/network/domain. Неизвестный IP, недоступный плагин, ошибка/таймаут либо смена регистрации дают unknown; явный отказ — denied. Отсутствующий principal даёт denied.

`authentication_verified: false` возвращается всегда: администратор проверил настройки прав субъекта, но не предъявление клиентом credential. Setup обязан отдельно подтвердить реальное клиентское подключение. Client proof к describe не заменяет CheckPermissions для конкретной операции.

```
av-agent permissions check --descriptor ABSOLUTE_DESCRIPTOR --owner-id com.astraverge.example-product --plugin-id
com.astraverge.demo-ping --principal-id com.astraverge.demo-client --client-address 127.0.0.1 --method Ping --json
```

SDK: `admin.CheckPermissions(ctx, installation, owner, plugin, query)`. CLI печатает полный результат; exit 0 только при decision=allowed, exit 3 при denied/unknown. Exit 0 не является доказательством аутентификации клиента.

Полный состав плагинов для обновления

Метод `plugins.inventory` требует отдельного action `plugins.inventory`. Это явно выданное право читать весь состав установки, включая чужих владельцев; оно не выводится из namespace/install/status и не даёт права их изменять.

Без target возвращаются требования к текущему Agent. С target можно проверить будущую версию до изменения службы:

```
{
  "method":"plugins.inventory",
  "target":{
    "agent_version":"2.1.0",
```

```
    "api_version":1,
    "ipc_version":1,
    "admin_contract":1,
    "capabilities":["plugin.management.v1","permissions.check.v1","plugins.inventory.v1"]
  }
}
```

Ответ включает ВСЕ static-конфигурации (включая disabled/unavailable) и managed регистрации; для каждого — ID, источник, известный owner, generation, enabled, версию, состояние, подписанные requirements, результат их проверки и совместимость с target. У static-плагина owner не придумывается. Settings, credentials, приватные пути и байты лицензий не возвращаются.

Requirements читаются локально из подписанного паспорта и сверяются с SHA-256 executable, без запуска плагина и без записи лицензионного state. Для работающего процесса дополнительно сверяется его принятый report и версия. Неизвестные или неподтверждённые данные включаются в blockers, а не исключаются из списка.

`complete` относится к полноте снимка; `upgrade_ready` — к подтверждённой совместимости. Они различаются: полный список с неизвестными требованиями имеет complete=true, upgrade_ready=false. Незавершённые операции, recovery, неизвестные требования или несовместимость блокируют готовность обновления. При смене managed-регистрации во время чтения complete=false. Snapshot SHA-256 не является lease или резервированием; Setup перепроверяет его перед заменой службы. Ответ сверх транспортного лимита — явная ошибка, не усечённый успех.

```
av-agent plugin inventory --descriptor ABSOLUTE_DESCRIPTOR --request TARGET_JSON --json
```

SDK: `admin.Inventory(ctx, installation, target)`. CLI exit 0 требует обоих complete/upgrade_ready; иначе exit 3 и полный ответ с blockers. Возможность использовать ранее настроенную службу сохраняется при блокировке обновления.

Установка плагина в общий Agent: контракт для любого продукта

Команды описаны в главе Работа с общей установкой. Типы и клиент SDK находятся в пакете `pkg/admin`.

Роли

Установщик подтверждает локальную OS-идентичность и делегированные owner/namespace/actions. Владелец управляет registration плагина. Потребитель имеет отдельную клиентскую identity и право invoke. Издатель executable подтверждается developer certificate. Строковые owner/product/developer ID не создают полномочий.

Если Agent не обнаружен

Вызовите `av-agent installation discover --descriptor PATH --json`. Ответ not_discovered означает отсутствие descriptor, а не доказанное отсутствие старой службы. AV-Setup должен сверить свой реестр установки. Продукт не сканирует порты, не переписывает чужой YAML и не создаёт вторую службу.

Если AV-Setup подтвердил отсутствие установки, он размещает совместимый executable, подготавливает аккаунт/права/конфигурацию службы и запускает её. Эта автоматизация принадлежит AV-Setup; в данной задаче внешний проект не изменён. Административный listener и descriptor публикует включённый Agent.

Если Agent работает

1. Получите доверенный путь CLI/descriptor от установки.
2. Выполните discovery, проверьте installation ID, API/IPC/admin contract и capabilities.
3. Прочитайте `plugin inspect --plugin-id ID --owner-id OWNER`; для нового ID используется generation 0.
4. Сохраните operation UUID до отправки. Подготовьте admin.Request и bundle без приватных ключей.
5. Выполните `plugin prepare --request FILE`; проверьте prepared, план и digest.
6. Выполните `plugin apply --operation-id UUID`. Нетерминальный ответ не является успехом.
7. Для каждого потребителя выполните настоящий authenticated describe probe с заголовком AV-Agent-Install-Probe из приложения, а не с credentials установщика.
8. Прочитайте `operation wait/status` до succeeded либо конечного отказа. После потери ответа используйте тот же UUID.
9. Проверьте прикладную готовность собственного runtime отдельно, если она обязательна для продукта.

Процесс работает через локальный OS-authenticated IPC; внешнего административного HTTP API нет. Публичный HTTP/HTTPS нужен только для обычных клиентских вызовов и proof. Для проверки доступа клиента используйте режим passive или both и проверяйте capability client.proof.v1.

Атомарность и обновление

Agent хранит managed registrations отдельно от static YAML. Он копирует и проверяет bundle, запускает staged-кандидата, проверяет доступ, дренирует старые вызовы и фиксирует новое поколение. Прежняя работающая registration и права сохраняются при подтверждённой ошибке до commit. Неопределённая ошибка хранения возвращает recovery_required; повтор с новым ID не разрешает «продавить» установку.

Совпадающий ID с другим содержимым — operation_conflict. Журнал переживает restart. Подтверждённый commit восстанавливается из реестра; незавершённый кандидат не получает скрытого повторного запуска предметных действий. Квота outcomes может запретить новые операции вместо удаления свидетельств и повторного исполнения старых.

Admin-путь требует StagedPlugin. Обновление произвольного executable с необратимой работой в конструкторе не обещается. Миграции native state и внешних БД — ответственность плагина. Инфраструктурный succeeded не равен «R/ERP/БД полностью работоспособны», если такая проверка не проводилась.

Удаление

Уполномоченный owner сначала отсоединяет свои consumer relationships через action detach_consumer. Удаление используемого другими потребителями плагина получает plugin_in_use. Затем remove_registration удаляет только собственную registration и маршрут. Общая служба и чужие плагины сохраняются; source bundle, результаты и working state не подвергаются purge.

Смена credentials общего principal, перенос ownership, принятие static YAML-плагина и автоматическая смена license/customer binding не выполняются молча. При конфликте нужна явная административная миграция.

Проверено и ещё требуется

Реальный локальный administrative normal/race сценарий проверен на Windows, включая commit storage fault и restart reconciliation. Новые Linux/FreeBSD/macOS peer credentials пока подтверждены кросс-сборкой. Требуется отдельная native-матрица и интеграция реального AV-Setup; старые системные отчёты этого не заменяют.

Обязательный контракт SDK

Все плагины реализуют CheckPermissions и DescribeCompatibility из pluginipc.ManagementHandler. pluginhost.Plugin включает этот интерфейс: прежняя реализация без методов не компилируется с новым SDK. Полный контракт содержит точные поля compatibility, методы OS-peer API, команды CLI и правила denied/unknown. Нет исключений для бесплатных/customer-плагинов.

CheckPermissions должен использовать ту же политику, что предметные вызовы, без их исполнения. В обоих demo авторизованный start/Ping можно проверить без активации и без изменения outbox. DescribeCompatibility возвращает подписываемые требования к Agent/API/IPC/capabilities и схемам состояния/настроек. До exec проверяется паспорт v2, при handshake — фактические методы и совпадение report.

Подпись сборки, лицензия покупателя и распространение

Три независимых слоя

1. Developer certificate и подпись паспорта подтверждают происхождение и целостность сборки.
2. Коммерческая лицензия, если её требует signed policy, подтверждает право запуска этого product/plugin у заказчика.
3. Авторизация подключения и предметные grants определяют право вызова/управления.

Для запуска каждого плагина обязательны сертификат разработчика, подписанный паспорт и действующая лицензия покупателя.

Утверждённая Developer Program

Модель программы за $99/год предусматривает персональную пару Ed25519 и сертификат разработчика. AstraVerge передаёт закрытый ключ разработчику один раз по защищённому каналу. Разработчик хранит его у себя и подписывает паспорт каждой сборки.

Для сертификата версии 2 AV-Guard выдаёт доверенную отметку времени на SHA-256 точных байтов сертификата и подписанного паспорта. Выпуск отметки возможен в период действия сертификата. Agent проверяет её автономно при каждом

новом запуске. Отметка, выданная в этот период, подтверждает ранее выпущенную сборку и после окончания подписки.

Состав поставки

```
plugin.exe                    # либо executable нужной Unix-платформы
developer.cert                # публичный сертификат разработчика
plugin.sig                    # подписанный паспорт финальной сборки
plugin.timestamp              # отметка AV-Guard для сертификата версии 2
build-trust.json              # подписанный пакет доверия для отметки
settings.json                 # безопасные начальные настройки
документация
```

Для установки плагина передаётся `plugin.avlic`. Для лицензии, подписанной ключом разработчика, дополнительно передаётся `plugin.delegation`. Agent проверяет делегирование по `build-trust.json` для точных организации, разработчика и продукта. Закрытый developer.key, private TLS keys, рабочие tokens и state заказчика в публичную поставку не входят.

Сертификат и паспорт

Сертификат `com.astraverge.plugin-developer/2` содержит `issuer_id`, `principal_id`, `developer_id`, публичный ключ Ed25519, `plugin_namespaces`, `product_namespaces` и интервал `[not_before, expires_at)` в секундах Unix. Сертификаты версии 1 продолжают проверяться по прежнему контракту. Он подписан корнем AstraVerge. Пространство разрешает точный ID и потомков через точку, а не похожий строковый префикс. Manifest не может расширить этот scope.

Паспорт содержит format `com.astraverge.plugin-build/2`, plugin_id, developer_id, version, os, arch, ipc_protocol, sha256 и обязательную license_policy. В текущем административном пути Identity.Version executable сверяется с версией паспорта. Для каждой платформенной сборки нужен собственный хеш и паспорт.

Конверт документов имеет key_id/payload/signature; payload и signature — строгий Base64URL без padding. Ed25519 подписывает точные байты:

```
FORMAT + LF + KEY_ID + LF + BASE64URL_PAYLOAD
```

Нет завершающего LF. Тип документа и key ID входят в подпись. Версии форматов и идентификаторы ключей предусмотрены для последующей ротации. Канонический подробный формат остаётся в PLUGIN-TRUST; не поддерживайте отличающуюся вторую схему в продукте.

Политика запуска

```
{"mode":"required","product_id":"av-example-product"}
```

В required нужна действительная лицензия с точными product/plugin/customer/license привязками и допустимой revision. Product namespace также проверяется по сертификату разработчика. Отсутствующая политика, неизвестный mode или противоречивое поле запрещают запуск. Локальный YAML не может снять required.

Лицензионный профиль schema 1.0.0 содержит purpose `av_agent_plugin`, launch_allowed и plugin_id. Модель допускает бесплатные и коммерческие плагины разных издателей, но текущий встроенный механизм проверки коммерческой лицензии доверяет системе лицензирования AstraVerge. Произвольный сторонний формат лицензий ядро не интерпретирует. Собственные договоры и проверки внутри плагина остаются предметной ответственностью автора.

Проверка Agent

Перед стартом/restart/reload Agent проверяет сертификат встроенным корнем, ключом разработчика — паспорт, затем пространства ID, платформу, IPC и SHA-256 executable. Для сертификата версии 2 Agent также проверяет отметку и подписанный пакет доверия. В private staging повторно проверяются точные копируемые байты. Также проверяется локальная лицензия каждого плагина, доверие к её делегированным ключам закрепляется подписанным trust-пакетом.

Сеть в этом пути не нужна. Истечение лицензии запрещает новые процессы, но не останавливает работающий процесс по таймеру и не блокирует его IPC только по сроку. Developer Program и лицензия покупателя — разные сроки и разные смыслы.

Отзыв developer certificates ещё не имеет законченного рабочего процесса. До его реализации нельзя обещать оперативное глобальное отключение скомпрометированного ключа. Подпись не является проверкой безопасности исходного кода.

Обновление и каналы распространения

Разработчик выбирает цену и свой канал поставки, включая 0 ₽. Для каталога нужны подтверждённые метаданные и действующие ссылки автора.

Новый installer использует контракт общей установки. Старые signed documents не переписываются автоматически; предшествующая общая лицензия av_agent не заменяет новую цепочку. Рабочие настройки, jobs, cursors и state сохраняются. Для staged-update нужны безопасная конструкция плагина и проверка совместимости его собственного состояния.

Обязательное поле compatibility и реализация CheckPermissions/DescribeCompatibility описаны в новом контракте. Отсутствие любого метода или расхождение с подписанным паспортом запрещает готовность процесса.

Подписи плагинов и независимые лицензии

Этот раздел описывает проверку доверия к плагинам. Для новых подписей используйте порядок выпуска сборок.

Проверка запуска

Перед каждым новым запуском, восстановлением supervisor и reload Agent локально проверяет цепочку Ed25519: встроенный корень AstraVerge → `developer.cert` → подписанный паспорт `plugin.sig` → SHA-256 executable. При staging executable копируется в закрытый каталог; повторная проверка политики и хеша относится к скопированным байтам. Ошибка не разрешает запуск и не включает запасной неподписанный режим. В частности, `customer_*` теперь также требует подпись.

При reload предварительный отказ сохраняет работающий процесс. Истечение коммерческой лицензии запрещает новые запуски и восстановление, но само по себе не останавливает процесс и не блокирует IPC. Оборванные вызовы не повторяются автоматически.

Проверка запуска не вызывает `Ensure`, discovery, login или refresh и не обращается к серверу. Получение/замена подписанных документов выполняются отдельным процессом. Замена документов должна выполняться атомарно; во время неполной поставки новый запуск может быть отклонён.

Подписанные документы

Сертификат и паспорт — JSON-конверты до 64 КиБ, с тремя обязательными строками:

```
{"key_id":"KEY_ID","payload":"BASE64URL_PAYLOAD","signature":"BASE64URL_SIGNATURE"}
```

Base64URL без padding, строгое каноническое кодирование. JSON payload кодируется в UTF-8; подписываются точные байты, а не повторная сериализация. Вход Ed25519:

```
FORMAT + "\n" + KEY_ID + "\n" + BASE64URL_PAYLOAD
```

Новые строки — одиночные байты LF, без завершающего LF. У сертификата `FORMAT` равен `com.astraverge.plugin-developer/1`, у паспорта — `com.astraverge.plugin-build/2`. Тип документа, key ID и payload связаны одной подписью. Алгоритм зафиксирован как Ed25519 и не выбирается входным документом. Версия формата и key ID предусматривают ротацию через последующие выпуски Agent; динамическое обновление корневого ключа здесь не добавлено.

Payload сертификата:

```
{
  "format":"com.astraverge.plugin-developer/1",
  "developer_id":"com.astraverge.example-developer",
  "public_key":"BASE64URL_ED25519_PUBLIC_KEY",
  "plugin_namespaces":["com.astraverge.example"],
  "product_namespaces":["av-example"]
}
```

Конверт сертификата подписывает AstraVerge, `key_id` выбирает встроенный публичный корень. `public_key` имеет ровно 32 байта. Закрытый `developer.key` разработчик хранит у себя, в поставку он не входит.

Пространство разрешает точный ID и потомков через точку: `com.astraverge.example` разрешает `com.astraverge.example.reader`, но не `com.astraverge.example-other`. Wildcard, пустые сегменты и пустой ID не допускаются. Право разработчика определяется сертификатом, не manifest. Для новых собственных ID применяется `com.astraverge`; исторические ID конфигурации могут сохраняться только при явном разрешении в подписанном сертификате.

Payload паспорта:

```
{
  "format":"com.astraverge.plugin-build/2",
  "plugin_id":"com.astraverge.example.reader",
  "developer_id":"com.astraverge.example-developer",
  "version":"1.0.0",
  "os":"windows",
  "arch":"amd64",
  "ipc_protocol":1,
  "compatibility":
{"management_version":1,"min_agent_version":"2.1.0","max_agent_version_exclusive":"3.0.0","api_version":1,"ipc_version":1,"ad
min_contract":0,"capabilities":[],"settings_schema_version":1,"state_schema_version":"1"},
  "sha256":"LOWERCASE_SHA256_HEX",
  "license_policy":{"mode":"required","product_id":"av-example"}
}
```

Паспорт подписывает разработчик; `key_id` конверта совпадает с `developer_id` сертификата. Agent проверяет ID, ОС/архитектуру текущего процесса, версию IPC и SHA-256. `version` обозначает версию сборки, но текущий профиль независимой лицензии не вводит диапазонов версий или права на обновления.

`license_policy` обязательна. Единственный допустимый режим — `required`. Он требует непустого `product_id`, разрешённого `product_namespaces` сертификата, и действительной лицензии на точный продукт и плагин. Для стороннего разработчика Product ID совпадает с полным DNS-идентификатором плагина, например `io.acme.demo-active`, и записывается в сертификат как точное значение. Сертификат версии 2 выдаётся AV-Guard после нового DNS TXT-подтверждения при каждом выпуске. Сегмент `astraverge` между точками зарезервирован за AstraVerge. Неизвестные режимы и поля, дубликаты ключей, некорректные типы и противоречия отвергаются. Локальная конфигурация и manifest не меняют подписанную политику.

Для сертификатов версии 1 срок подписки при проверке подписи не используется. Сертификаты версии 2 проверяются по доверенной отметке времени, описанной ниже. Отзыв самого сертификата разработчика и его онлайн-проверка этим контрактом не предусмотрены.

Конфигурация

```
plugins:
  com.astraverge.example.reader:
    enabled: true
    executable: plugins/reader.exe
    settings_file: plugins/reader.settings.json
    developer_certificate: plugins/developer.cert
    release_file: plugins/reader.sig
    license:
      file: licenses/reader.avlic
      issuer_id: db804b8f-f073-4421-85ae-5a92bd94365e
      product: av-example
      license_id: REPLACE_WITH_LICENSE_UUID
      customer_id: REPLACE_WITH_CUSTOMER_UUID
      minimum_revision: 1
      state_directory: state/licenses/reader
      # При лицензии, подписанной делегированным ключом AV-Guard:
      trust_package:
        envelope_file: licenses/trust.json
        state_directory: state/trust/reader
```

Пути из YAML разрешаются относительно конфигурации; для программно переданной конфигурации задаются абсолютные пути. Блок `license` обязателен для каждого включённого плагина. `trust_package.roots` и верхний `license_trust` для новых

запусков не принимаются: корни берутся из Agent, а не входной поставки или YAML. Без `trust_package` допустима лицензия, подписанная непосредственно встроенным корнем. Общие старые `licensing.roots` не меняют доверие новому пути запуска.

Встроенный публичный ключ используется из опубликованного материала AstraVerge (`astraverge_trust_root_2026`); приватный ключ в Agent отсутствует. Для коммерческих лицензий делегированные ключи из подписанного trust-пакета проверяются SDK AV-Guard. Подписанные сертификаты разработчиков нового формата должны быть выпущены владельцем корня отдельно; Agent их не выпускает и не подменяет тестовыми.

Независимая коммерческая лицензия

Прямая подпись AstraVerge

Для собственных плагинов AV-Guard подписывает паспорт формата `com.astraverge.plugin-build/2` через `POST /api/v1/astraverge/plugin-releases`. Agent проверяет издателя AstraVerge, область ключа `release/1/astraverge-plugin-build`, подписанное время выпуска и привязку паспорта к ID, платформе, IPC и SHA-256 бинарника. Плагин и разработчик используют пространство `com.astraverge.`, продукт лицензии использует ID `av-*`. Лицензия по-прежнему обязательна.

В статической конфигурации укажите `release_file` и `build_trust`. В административном запросе передайте конверт в `files.release` и подписанный пакет доверия в `files.build_trust`. Поля сертификата разработчика, отдельной отметки времени и делегирования лицензии оставьте пустыми.

Сборки по подписке разработчика

Сертификат `com.astraverge.plugin-developer/2` дополнительно связывает `issuer_id`, `principal_id` и интервал `[not_before, expires_at)` в секундах Unix. Для каждой платформенной сборки AV-Guard выдаёт отметку на точные байты сертификата и подписанного паспорта. При запуске проверяется время выпуска отметки внутри интервала сертификата. Уже выпущенная сборка может запускаться после окончания подписки при соблюдении остальных условий доверия.

В статической конфигурации укажите `build_timestamp` и `build_trust` с полями `envelope_file` и `state_directory`. Пакет доверия проверяется по встроенному корню организации. Состояние пакета хранится локально для защиты от отката. Административная установка принимает файлы `build_timestamp` и `build_trust` в `files` и сохраняет состояние доверия при обновлении.

Если коммерческую лицензию подписал разработчик, добавьте `license_delegation`. Его подпись проверяется по пакету `build_trust`, а ключ разработчика применяется к лицензии только при совпадении организации, разработчика и точного продукта. Для этого варианта `license.trust_package` следует опустить. Сертификаты версии 1 сохраняют существующий путь проверки без отметки.

Лицензия `.avlic v1` должна содержать подписанные непустые строки `issuer_display_name`, `customer_display_name` и `customer_email`. SDK AV-Guard проверяет их вместе с подписью и остальными полями. Agent связывает лицензию с покупателем по `customer_id` и дополнительно проверяет схему `1.0.0` и entitlements:

```
{"purpose":"av_agent_plugin","launch_allowed":true,"plugin_id":"com.astraverge.example.reader"}
```

Продукт лицензии точно совпадает с `license_policy.product_id`, ID плагина — с паспортом и конфигурацией. Пропущенные/неправильно типизированные права запрещают запуск. Старые поля привязки к сборке (`executable_sha256`, `os`, `arch`, `ipc_protocol`) в этом профиле отвергаются: они требуют явной миграции, а не молчаливого расширения лицензии.

Успешно принятые revision и SHA-256 подписанного документа сохраняются атомарно под эксклюзивным lock в `<state_directory>/launch-rights/entitlement.json`. Если `state_directory` не задан, используется `<license.file>.state`. Более старая revision, подмена байтов той же revision, повреждённое state и смена привязок запрещают запуск. Trust sequence и ограничения делегированных ключей сохраняются отдельно средствами существующего Trust SDK. Configuration/state/logs не удаляются для обновления. Учётной записи службы нужны права записи в указанные state-каталоги.

Разрешение запуска не заменяет право вызова или управления. `principals[].plugins` и `manage_plugins` остаются независимыми списками, предметные права проверяет плагин.

Demo-ping 2.1.0: учебный плагин

Полный код публичного примера на основе `agent_plugins/demo-ping`. ID учебного плагина — `com.astraverge.demo-ping`, версия — 2.1.0. Обвязка SDK может переопределить ID и обработчик Ping для примеров на других языках. Настройки `allowed_principals` и `allowed_products` проверяются по доверенному IPC-контексту. Неизвестные поля, пустые списки и несколько JSON-значений отклоняются.

Структура проекта

```
examples/
  go.mod
  go.sum
  demo-ping/main.go
  demo-ping/main_test.go
```

Полный исходный код

Каноническая публикуемая копия: main.go. Блок обновляется из этого файла инструментом tools/render_demo_docs.py, чтобы учебник не расходился с компилируемым кодом.

```
// demo-ping is a trusted, explicitly authorized Ping/Pong SDK example.
package main

import (
        "bytes"
        "context"
        "encoding/json"
        "errors"
        "io"
        "slices"
        "sync/atomic"

        "astraverge.local/av-agent/v2/pkg/pluginhost"
        "astraverge.local/av-agent/v2/pkg/pluginipc"
)

var pluginID = "com.astraverge.demo-ping"

const pluginVersion = "2.1.0"

var ping = func(context.Context) pluginipc.Response {
        return pluginipc.Response{Body: json.RawMessage(`{"reply":"Pong"}`)}
}

type settings struct {
        AllowedPrincipals []string `json:"allowed_principals"`
        AllowedProducts   []string `json:"allowed_products"`
}
type plugin struct {
        settings settings
        ready    atomic.Bool
}

func open(raw json.RawMessage) (pluginhost.Plugin, error) {
        var c settings
        decoder := json.NewDecoder(bytes.NewReader(raw))
        decoder.DisallowUnknownFields()
        if decoder.Decode(&c) != nil || decoder.Decode(new(any)) != io.EOF {
                return nil, errors.New("one valid settings object required")
        }
        if len(c.AllowedPrincipals) == 0 || len(c.AllowedProducts) == 0 || slices.Contains(c.AllowedPrincipals, "") ||
slices.Contains(c.AllowedProducts, "") {
                return nil, errors.New("explicit principal and product allowlists required")
        }
        return &plugin{settings: c}, nil
}
func (*plugin) Identity() pluginipc.Identity {
        return pluginipc.Identity{ID: pluginID, Version: pluginVersion, SchemaVersion: 1, Operations: []string{"Ping"}}
}
func (p *plugin) Activate(ctx context.Context) error {
```

```
        if err := ctx.Err(); err != nil {
                return err
        }
        p.ready.Store(true)
        return nil
}
func (p *plugin) Invoke(ctx context.Context, method string, _ json.RawMessage) pluginipc.Response {
        decision, checkErr := p.CheckPermissions(ctx, pluginipc.PermissionQuery{Operation: method})
        if checkErr != nil {
                return pluginipc.Response{Error: &pluginipc.Problem{Code: "cancelled", Message: "request cancelled"}}
        }
        if decision.Decision != "allowed" && decision.Reason != "unsupported_operation" {
                return pluginipc.Response{Error: &pluginipc.Problem{Code: "forbidden", Message: "principal, product or scope
not allowed"}}
        }
        if !p.ready.Load() {
                return pluginipc.Response{Error: &pluginipc.Problem{Code: "not_ready", Message: "plugin not activated"}}
        }
        if err := ctx.Err(); err != nil {
                return pluginipc.Response{Error: &pluginipc.Problem{Code: "cancelled", Message: "request cancelled"}}
        }
        if method != "Ping" {
                return pluginipc.Response{Error: &pluginipc.Problem{Code: "unsupported_method", Message: "demo supports
Ping"}}
        }
        return ping(ctx)
}
func (p *plugin) Close() error { p.ready.Store(false); return nil }
func main()                    { pluginhost.Main(pluginID, open) }

func (p *plugin) DescribeCompatibility(context.Context) (pluginipc.Compatibility, error) {
        return pluginipc.Compatibility{ManagementVersion: 1, MinAgentVersion: "2.1.0", MaxAgentVersionExclusive: "3.0.0",
APIVersion: 1, IPCVersion: 1, SettingsSchemaVersion: 1, StateSchemaVersion: "1"}, nil
}
func (p *plugin) CheckPermissions(ctx context.Context, q pluginipc.PermissionQuery) (pluginipc.PermissionDecision, error) {
        if err := ctx.Err(); err != nil {
                return pluginipc.PermissionDecision{}, err
        }
        who := pluginipc.ExecutionFromContext(ctx)
        if !slices.Contains(p.settings.AllowedPrincipals, who.PrincipalID) || !slices.Contains(p.settings.AllowedProducts,
who.Product) {
                return pluginipc.PermissionDecision{Decision: "denied", Reason: "identity_not_allowed"}, nil
        }
        if q.Scope != "" || q.Resource != "" {
                return pluginipc.PermissionDecision{Decision: "denied", Reason: "unsupported_scope_or_resource"}, nil
        }
        if !slices.Contains([]string{"Ping"}, q.Operation) {
                return pluginipc.PermissionDecision{Decision: "denied", Reason: "unsupported_operation"}, nil
        }
        return pluginipc.PermissionDecision{Decision: "allowed"}, nil
}
```

Как работает код

Identity объявляет точный ID, версию 2.1.0, schema 1 и операцию Ping. Поле Protocol окончательно выставляется SDK. Invoke принимает context/method/raw JSON; для Ping возвращает `{"reply":"Pong"}`, для остальных методов — Problem с кодом unsupported_method. Close сбрасывает готовность; повторный вызов безопасен.

Main передаёт фиксированный ID и фабрику в pluginhost.Main. Settings обязательны: `{"allowed_principals":["com.astraverge.demo-client"],"allowed_products":["com.astraverge.demo-product"]}`. Launch config с ID/token/endpoint формирует родитель. Самостоятельный запуск без private launch-файла не является корректным способом подключения.

Сборка

Из корня examples:

```
go mod tidy
go build -o demo-ping.exe ./demo-ping
```

Для Unix задайте имя выходного файла без `.exe`. Сборку выполняйте установленным Go 1.27.4. В примере из архива SDK 2.1.0 локальный `replace` указывает на `source/` из комплекта.

Локальный запуск и реальные вызовы

```
go test -count=1 -timeout=120s ./demo-ping
```

Тест запускает реальный собранный executable с private launch/endpoint, открывает parent lease, вызывает describe, затем фактический `Client.Call(ctx, "Ping", nil)`. Проверяется body `{"reply":"Pong"}`. Для `unknown` ожидается Error.Code=unsupported_method и Message=demo supports Ping. Неверный private IPC credential отклоняется до предметного вызова.

Отдельный conformance-набор проверяет framing, ошибки, отмену и SDK lifecycle. Тестовый родитель затем выполняет __shutdown либо закрывает lease и ждёт успешного завершения executable. Это native private IPC-проверка, **не unsigned Developer Mode Agent**.

Production-конфигурация после получения подписей

```
plugins:
  com.astraverge.demo-ping:
    enabled: true
    executable: /absolute/path/demo-ping
    settings_file: /absolute/path/settings.json
    developer_certificate: /absolute/path/developer.cert
    release_file: /absolute/path/plugin.sig
```

Principal отдельно получает этот ID в plugins. Это фрагмент существующей защищённой конфигурации, не полный файл и не обход сертификатов. При signed policy none коммерческая лицензия не нужна. Без реального developer.cert и подписи подходящего корня проверка должна отказать.

После легитимной production-настройки реальный HTTP-вызов:

```
POST /api/v1/plugins/com.astraverge.demo-ping/call
Authorization: Bearer <CLIENT_TOKEN>

{"method":"Ping","body":null}
```

Здесь Bearer используется только в явно настроенном профиле; в HTTPS/mTLS нужен клиентский сертификат. Ожидаемый ответ `{"body":{"reply":"Pong"}}`. Для unknown — `{"error":{"code":"unsupported_method","message":"demo supports Ping"}}` при выполненном IPC-вызове. В Local Net (Test) подписанный `demo-ping` проверен через статическую конфигурацию на шести BM и Mac mini Intel. Для сертификата версии 2 в конфигурацию добавляются `build_timestamp` и `build_trust`, а для лицензии, подписанной разработчиком, — `license_delegation`.

Переход к подписанному release

Получите персональный ключ/сертификат по будущему готовому процессу Developer Program, соберите окончательные байты, вычислите SHA-256 и подпишите паспорт для точного ID/версии/OS/arch/IPC. Закрытый ключ не включайте в поставку. Процедура ещё не имеет готовой публичной команды sign.

StagedPlugin реализован: Activate подтверждает локальную готовность без внешних действий. До активации предметный вызов отклоняется с not_activated. Статический SDK-запуск вызывает Activate до обслуживания.

Полные результаты компиляции, tests и cross-build — отчёт.

Обязательный контракт SDK

Все плагины реализуют CheckPermissions и DescribeCompatibility из pluginipc.ManagementHandler. pluginhost.Plugin включает этот интерфейс: прежняя реализация без методов не компилируется с новым SDK. Полный контракт содержит точные поля compatibility, методы OS-peer API, команды CLI и правила denied/unknown. Нет исключений для бесплатных/customer-плагинов.

CheckPermissions должен использовать ту же политику, что предметные вызовы, без их исполнения. В обоих demo авторизованный start/Ping можно проверить без активации и без изменения outbox. DescribeCompatibility возвращает подписываемые требования к Agent/API/IPC/capabilities и схемам состояния/настроек. До exec проверяется паспорт v2, при handshake — фактические методы и совпадение report.

Demo-active 2.1.0: учебный плагин

Полный код учебного плагина из agent_plugins/demo-active. ID com.astraverge.demo-active. Версия 2.1.0. Обязательные allowed_principals/allowed_products проверяются по доверенному IPC-контексту; неверная идентичность получает forbidden. Неизвестные поля, пустые allowlists и несколько JSON-значений настроек отклоняются.

Структура

```
examples/
  go.mod
  go.sum
  demo-active/main.go
  demo-active/main_test.go
```

Полный исходный код

Каноническая публикуемая копия: main.go.

```
// demo-active publishes only after an explicitly authorized start operation.
package main

import (
        "bytes"
        "context"
        "encoding/json"
        "errors"
        "io"
        "log"
        "path/filepath"
        "slices"
        "sync"
        "time"

        "astraverge.local/av-agent/v2/pkg/active"
        "astraverge.local/av-agent/v2/pkg/pluginhost"
        "astraverge.local/av-agent/v2/pkg/pluginipc"
)

const pluginID = "com.astraverge.demo-active"
const pluginVersion = "2.1.0"

type settings struct {
        AllowedPrincipals    []string              `json:"allowed_principals"`
        AllowedProducts      []string              `json:"allowed_products"`
        Transfers            active.TransferConfig `json:"transfers"`
        IntervalSeconds      int                   `json:"interval_seconds"`
        FlushIntervalSeconds int                   `json:"flush_interval_seconds"`
        PayloadBytes         int64                 `json:"payload_bytes"`
        Queue                active.QueueConfig    `json:"queue"`
        Endpoint             string                `json:"endpoint"`
        Protocol             string                `json:"protocol"`
        AgentID              string                `json:"agent_id"`
        StateDirectory       string                `json:"state_directory"`
        Security             active.Security       `json:"security"`
}
type runState struct {
        cancel context.CancelFunc
        wg     sync.WaitGroup
        outbox *active.Publisher
}
type plugin struct {
        control       sync.Mutex
        mu            sync.Mutex
        config        settings
        exchange      active.Exchange
```

```
        run           *runState
        ready, closed bool
        stopping      bool
        closeErr      error
}

func open(raw json.RawMessage) (pluginhost.Plugin, error) {
        var c settings
        decoder := json.NewDecoder(bytes.NewReader(raw))
        decoder.DisallowUnknownFields()
        if decoder.Decode(&c) != nil || decoder.Decode(new(any)) != io.EOF {
                return nil, errors.New("one valid settings object required")
        }
        if c.AgentID == "" || !filepath.IsAbs(c.StateDirectory) {
                return nil, errors.New("agent_id and absolute state_directory required")
        }
        if len(c.AllowedPrincipals) == 0 || len(c.AllowedProducts) == 0 || slices.Contains(c.AllowedPrincipals, "") ||
slices.Contains(c.AllowedProducts, "") {
                return nil, errors.New("explicit principal and product allowlists required")
        }
        if c.IntervalSeconds == 0 {
                c.IntervalSeconds = 5
        }
        if c.FlushIntervalSeconds == 0 {
                c.FlushIntervalSeconds = 1
        }
        if c.IntervalSeconds < 1 || c.FlushIntervalSeconds < 1 || int64(c.IntervalSeconds) > int64((1<<63-1)/time.Second) ||
int64(c.FlushIntervalSeconds) > int64((1<<63-1)/time.Second) || c.PayloadBytes < 0 || c.PayloadBytes > active.Petabyte {
                return nil, errors.New("invalid interval or payload size")
        }
        if err := c.Transfers.Defaults(); err != nil {
                return nil, err
        }
        if err := c.Queue.Defaults(); err != nil {
                return nil, err
        }
        if c.PayloadBytes > 0 && (!c.Transfers.Enabled || c.PayloadBytes > c.Transfers.MaxBatchBytes) {
                return nil, errors.New("stream payload requires transfers and sufficient max_batch_bytes")
        }
        if c.Protocol == "" {
                c.Protocol = "rest"
        }
        exchange, err := active.NewExchange(c.Protocol, c.Endpoint, c.Security)
        if err != nil {
                return nil, err
        }
        // Neither construction nor activation opens an outbox or starts network I/O.
        // A running old generation may still own the same persistent outbox.
        return &plugin{config: c, exchange: exchange}, nil
}
func (*plugin) Identity() pluginipc.Identity {
        return pluginipc.Identity{ID: pluginID, Version: pluginVersion, SchemaVersion: 1, Operations: []string{"status",
"start", "stop"}}
}
func (p *plugin) Activate(ctx context.Context) error {
        if err := ctx.Err(); err != nil {
                return err
        }
        p.mu.Lock()
        defer p.mu.Unlock()
        if p.closed {
                return errors.New("plugin closed")
        }
        p.ready = true
        return nil
}
func (p *plugin) Invoke(ctx context.Context, method string, _ json.RawMessage) pluginipc.Response {
        decision, checkErr := p.CheckPermissions(ctx, pluginipc.PermissionQuery{Operation: method})
        if checkErr != nil {
                return pluginipc.Response{Error: &pluginipc.Problem{Code: "cancelled", Message: "request cancelled"}}
        }
        if decision.Decision != "allowed" && decision.Reason != "unsupported_operation" {
                return pluginipc.Response{Error: &pluginipc.Problem{Code: "forbidden", Message: "principal, product or scope
not allowed"}}
        }
        if ctx.Err() != nil {
                return problem("cancelled", "request cancelled")
```

```
        }
        switch method {
        case "status":
                return p.status()
        case "start":
                if err := p.start(ctx); err != nil {
                        return problem("start_failed", "publisher could not start; inspect configuration and state")
                }
                return p.status()
        case "stop":
                if err := p.stop(); err != nil {
                        return problem("stop_failed", "publisher stop was not confirmed")
                }
                return p.status()
        default:
                return problem("unsupported_method", "demo supports status, start and stop")
        }
}
func problem(code, message string) pluginipc.Response {
        return pluginipc.Response{Error: &pluginipc.Problem{Code: code, Message: message}}
}
func (p *plugin) status() pluginipc.Response {
        p.mu.Lock()
        defer p.mu.Unlock()
        state := "prepared"
        if p.closed {
                state = "closed"
        } else if p.stopping {
                state = "stopping"
        } else if p.run != nil {
                state = "running"
        } else if p.ready {
                state = "stopped"
        }
        var queue any
        if p.run != nil {
                queue = p.run.outbox.QueueStats()
        }
        body, _ := json.Marshal(map[string]any{"status": "ok", "state": state, "interval_seconds": p.config.IntervalSeconds,
"queue_open": p.run != nil, "queue": queue})
        return pluginipc.Response{Body: body}
}
func (p *plugin) start(ctx context.Context) error {
        p.control.Lock()
        defer p.control.Unlock()
        p.mu.Lock()
        closed, ready, running := p.closed, p.ready, p.run != nil
        p.mu.Unlock()
        if closed || !ready {
                return errors.New("plugin not ready")
        }
        if running {
                return nil
        }
        if err := ctx.Err(); err != nil {
                return err
        }
        outbox, err := active.OpenPublisherWithConfig(p.config.StateDirectory, p.config.Queue)
        if err != nil {
                return err
        }
        if p.config.Transfers.Enabled {
                if err = outbox.EnableTransfers(p.config.Transfers); err != nil {
                        _ = outbox.Close()
                        return err
                }
        }
        runCtx, cancel := context.WithCancel(context.Background())
        r := &runState{cancel: cancel, outbox: outbox}
        c := p.config
        r.wg.Add(2)
        if c.Transfers.Enabled {
                r.wg.Add(1)
                go func() { defer r.wg.Done(); outbox.RunMaintenance(runCtx) }()
        }
        go func() {
                defer r.wg.Done()
```

```
                ticker := time.NewTicker(time.Duration(c.IntervalSeconds) * time.Second)
                defer ticker.Stop()
                for {
                        select {
                        case <-runCtx.Done():
                                return
                        case <-ticker.C:
                                var err error
                                if c.PayloadBytes > 0 {
                                        _, err = outbox.EnqueueStream(runCtx, c.AgentID, pluginID,
io.LimitReader(testByte('x'), c.PayloadBytes))
                                } else {
                                        _, err = outbox.Enqueue(pluginID, json.RawMessage(`{"status":"ok"}`))
                                }
                                if err != nil && runCtx.Err() == nil {
                                        log.Print("demo-active: enqueue failed; outbox retained")
                                }
                        }
                }
        }()
        go func() {
                defer r.wg.Done()
                ticker := time.NewTicker(time.Duration(c.FlushIntervalSeconds) * time.Second)
                defer ticker.Stop()
                for {
                        if err := outbox.Flush(runCtx, p.exchange, c.AgentID); err != nil && runCtx.Err() == nil {
                                log.Print("demo-active: delivery pending; outbox retained")
                        }
                        select {
                        case <-runCtx.Done():
                                return
                        case <-ticker.C:
                        }
                }
        }()
        p.mu.Lock()
        p.run = r
        p.mu.Unlock()
        return nil
}
func (p *plugin) stopRun() error {
        p.mu.Lock()
        r := p.run
        p.stopping = r != nil
        p.mu.Unlock()
        if r == nil {
                return nil
        }
        r.cancel()
        r.wg.Wait()
        err := r.outbox.Close()
        p.mu.Lock()
        p.run = nil
        p.stopping = false
        p.mu.Unlock()
        return err
}
func (p *plugin) stop() error { p.control.Lock(); defer p.control.Unlock(); return p.stopRun() }
func (p *plugin) Close() error {
        p.control.Lock()
        defer p.control.Unlock()
        p.mu.Lock()
        if p.closed {
                err := p.closeErr
                p.mu.Unlock()
                return err
        }
        p.closed = true
        p.ready = false
        p.mu.Unlock()
        err := errors.Join(p.stopRun(), p.exchange.Close())
        p.mu.Lock()
        p.closeErr = err
        p.mu.Unlock()
        return err
}
```

```
type testByte byte

func (b testByte) Read(p []byte) (int, error) {
        for n := range p {
                p[n] = byte(b)
        }
        return len(p), nil
}
func main() { pluginhost.Main(pluginID, open) }

func (p *plugin) DescribeCompatibility(context.Context) (pluginipc.Compatibility, error) {
        return pluginipc.Compatibility{ManagementVersion: 1, MinAgentVersion: "2.1.0", MaxAgentVersionExclusive: "3.0.0",
APIVersion: 1, IPCVersion: 1, SettingsSchemaVersion: 1, StateSchemaVersion: "1"}, nil
}
func (p *plugin) CheckPermissions(ctx context.Context, q pluginipc.PermissionQuery) (pluginipc.PermissionDecision, error) {
        if err := ctx.Err(); err != nil {
                return pluginipc.PermissionDecision{}, err
        }
        who := pluginipc.ExecutionFromContext(ctx)
        if !slices.Contains(p.config.AllowedPrincipals, who.PrincipalID) || !slices.Contains(p.config.AllowedProducts,
who.Product) {
                return pluginipc.PermissionDecision{Decision: "denied", Reason: "identity_not_allowed"}, nil
        }
        if q.Scope != "" || q.Resource != "" {
                return pluginipc.PermissionDecision{Decision: "denied", Reason: "unsupported_scope_or_resource"}, nil
        }
        if !slices.Contains([]string{"status", "start", "stop"}, q.Operation) {
                return pluginipc.PermissionDecision{Decision: "denied", Reason: "unsupported_operation"}, nil
        }
        return pluginipc.PermissionDecision{Decision: "allowed"}, nil
}
```

Настройки

Поле	Значение и проверка
agent_id	Обязательный ID peer на принимающей стороне
state_directory	Обязательный абсолютный собственный outbox-каталог
endpoint	Адрес настроенного приёмника, не самого plugin IPC
protocol	rest по умолчанию либо grpc
security	Настройки CA/certificate/private key либо явного Bearer файла
interval_seconds	5 по умолчанию, положительный интервал события
flush_interval_seconds	1 по умолчанию, положительный интервал отправителя
queue	Реальный active.QueueConfig: квоты/batch/rate/retention
transfers	TransferConfig для больших событий, выключен по умолчанию
payload_bytes	0 — JSON ok; больше 0 — synthetic stream, только при enabled transfers

Безопасный шаблон, который нужно дополнить собственными путями и receiver:

```
{
  "allowed_principals":["com.astraverge.demo-client"],
  "allowed_products":["com.astraverge.demo-product"],
  "agent_id":"com.astraverge.demo-agent",
  "state_directory":"ABSOLUTE_PRIVATE_STATE_DIRECTORY",
  "endpoint":"http://127.0.0.1:LOCAL_RECEIVER_PORT",
  "protocol":"rest",
  "interval_seconds":5,
  "flush_interval_seconds":1,
  "security":{"bearer_file":"ABSOLUTE_PRIVATE_BEARER_FILE"},
  "payload_bytes":0
}
```

Это шаблон с placeholders, не готовая инфраструктурная конфигурация. В production предпочтителен HTTPS/mTLS. HTTP допустим только при явном выборе Bearer-профиля. Не копируйте рабочие tokens или внутренние адреса в пример.

Реальное взаимодействие

open проверяет настройки и создаёт Exchange без отправки. Ни open, ни Activate не открывают outbox и не запускают фоновые задачи. Только разрешённый start после завершения установки открывает Publisher и запускает scheduler/flusher. По умолчанию каждые 5 секунд создаётся событие `{"status":"ok"}`. Повторный start идемпотентен.

status возвращает status, state, interval_seconds, queue_open и queue. До start queue=null. Состояния: prepared, stopped, running, stopping, closed. stop ждёт goroutines и закрытия outbox; Close также закрывает Exchange. State не удаляется. Ошибки запуска/остановки: start_failed/stop_failed. Неизвестный метод: unsupported_method.

Сборка и локальный запуск

```
cd doc_market/examples
go mod tidy
go build -o demo-active.exe ./demo-active
go test -count=1 -timeout=120s ./demo-active
```

Тест создаёт настоящий SDK Broker/RESTHandler на loopback, генерирует временный Bearer и SHA-256 binding, запускает отдельный executable через private test parent. Интервал сокращён до 1 секунды только настройками теста. Проверяются describe/credential, IPC status, unsupported_method и фактическое durably принятое событие `{"status":"ok"}` с Plugin=com.astraverge.demo-active.

После закрытия parent lease тест ожидает завершения executable и заново открывает outbox тем же SDK: это проверка освобождения lock. Неверные обязательные настройки должны завершить фабрику ошибкой, не создать «успешный» пустой плагин.

Вызовы и ожидаемые данные

После __activate тест вызывает `Client.Call(ctx, "start", nil)`, status и stop. До активации start возвращает not_activated. Чужой principal получает forbidden. Извлекаемый body содержит `{"status":"ok","interval_seconds":1,"queue":...}`. QueueStats меняются по времени/ACK, поэтому полный объект не фиксируется выдуманной константой.

После легитимной signed/static установки публичный запрос — POST `/api/v1/plugins/com.astraverge.demo-active/call` с обычной аутентификацией и `{"method":"status","body":null}`. Ответ обёрнут в `{"body":...}`. Unknown возвращает `{"error":{"code":"unsupported_method","message":"demo supports status, start and stop"}}`.

Событие не является этим IPC-ответом: оно отдельно Enqueue/Flush доставляется receiver. Наличие receipt/ACK не превращает произвольные внешние действия в exactly once. При ошибке отправки код сохраняет outbox и пишет безопасное сообщение без credentials.

Подписанный выпуск и общая установка

Паспорт должен содержать точные ID/version/platform/IPC/hash и подписанную policy. Подпись, лицензия покупателя и права потребителя независимы. До выдачи рабочих сертификатов production-регистрация учебной копии не считается проверенной.

Staged install: candidate инертен, поэтому старая генерация может владеть тем же outbox до переключения. После подтверждённого apply клиент явно вызывает start. Автоматического возобновления отправки после update/restart нет. Native-тест проверяет отсутствие outbox до start и освобождение lock после stop/lease loss.

Учебный private IPC-запуск не обходит production-проверку и не называется Developer Mode Agent. Полные результаты и native/cross-различия — отчёт.

Обязательный контракт SDK

Все плагины реализуют CheckPermissions и DescribeCompatibility из pluginipc.ManagementHandler. pluginhost.Plugin включает этот интерфейс: прежняя реализация без методов не компилируется с новым SDK. Полный контракт содержит точные поля compatibility, методы OS-peer API, команды CLI и правила denied/unknown. Нет исключений для бесплатных/customer-плагинов.

CheckPermissions должен использовать ту же политику, что предметные вызовы, без их исполнения. В обоих demo авторизованный start/Ping можно проверить без активации и без изменения outbox. DescribeCompatibility возвращает подписываемые требования к Agent/API/IPC/capabilities и схемам состояния/настроек. До exec проверяется паспорт v2, при handshake — фактические методы и совпадение report.

Service Health — проверки для балансировщика

`com.astraverge.service-health` 1.0.0 — плагин AstraVerge. Он проверяет HTTP-точку, процесс или оба условия и возвращает текст для HAProxy `agent-check`. Если выбраны обе проверки, для готовности нужны оба успешных результата. Слушатель открывает Agent, а плагин определяет критерий готовности.

Настройка

1. Включите подписанную сборку в `plugins` Agent. Для запуска нужна действующая подписанная лицензия AstraVerge на `av-agent-service-health`.
2. В `socket_inputs` задайте TCP-адрес и ID плагина. Для сетевого входа используйте токен и разрешённые адреса отправителей. При необходимости включите TLS на TCP-входе.
3. В JSON-настройках плагина задайте те же `network` и `address`, HTTP-точку и/или процесс. Сопоставьте ID и продукты клиентов с `allowed_principals` и `allowed_products`.
4. В HAProxy настройте `agent-check` на этот адрес. Для входа в режиме токена передайте `agent-send <секрет>`. Секрет состоит из 64 ASCII-символов шестнадцатеричного представления, а Agent хранит SHA-256 этой строки.

TCP и UDP в `socket_inputs`

Добавьте в YAML Agent эти входы в режиме `passive` или `both`. TCP-вход использует токен HAProxy. UDP-вход предназначен для локальной проверки без токена и привязан к loopback.

```
socket_inputs:
  - network: tcp
    address: 127.0.0.1:9230
    plugin: com.astraverge.service-health
    clients:
      - id: haproxy
        product: service-monitoring
        token_sha256: "<SHA-256 ASCII-строки секрета>"
        allowed_networks: ["127.0.0.1/32"]
  - network: udp
    address: 127.0.0.1:9231
    plugin: com.astraverge.service-health
    token_sha256: ""
    max_datagram_bytes: 1024
```

Замените заполнитель TCP-хешем своего секрета. Для HAProxy укажите исходную ASCII-строку в `agent-send`. Пара `network` и `address` для каждого входа должна совпадать с JSON-настройками плагина. Если вы переопределяли `limits.max_request_bytes`, для UDP-примера задайте не менее 1024 байт.

```
{
  "allowed_principals": ["haproxy"],
  "allowed_products": ["service-monitoring"],
  "checks": [
    {
      "network": "tcp",
      "address": "127.0.0.1:9230",
      "timeout_ms": 800,
      "http": {"url": "http://127.0.0.1:8080/healthz", "status": 200},
      "process": {"name": "nginx"},
      "weight_degraded": 25
    },
    {
      "network": "udp",
      "address": "127.0.0.1:9231",
      "http": {"url": "http://127.0.0.1:8080/healthz", "status": 200}
    }
```

```
  ]
}
```

После запуска агента отправьте 1024 байта на UDP-вход. При готовности пример выводит `100% ready`.

```
import socket

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as probe:
    probe.settimeout(2)
    probe.connect(("127.0.0.1", 9231))
    probe.send(bytes(1024))
    print(probe.recv(1024).decode("utf-8").rstrip("\n"))
```

Ответы и эксплуатация

- При готовности ответ начинается с `100% ready`. При сбое ответ содержит `drain fail` и причину, например `#http_503` или `#process_not_found`.
- `maxconn_ok` и `maxconn_failed` позволяют указать лимиты соединений. При корректной первой строке `mode_file` плагин использует её как ручной режим.
- Агент ограничивает UDP-ответ размером входной датаграммы. Для штатного `100% ready` нужны 11 байт с учётом перевода строки. Запрос в 1024 байта покрывает также диагностику и ручной режим. В режиме токена его 64 байта входят в этот размер.
- Сборки проверялись на Windows и кросс-компиляцией для других целевых ОС. Фактическую работу проверки процесса, сокета и HAProxy нужно подтвердить на целевой установке.

Настройки плагина · Настройки сокетов Agent · Статус клиентской поставки

Экспортируемые объявления SDK

Справочник сгенерирован из дерева исходников Agent. Он включает публичные пакеты модуля. Состав локально собранного SDK 2.1.0 описан в описании выпуска. Закрытые поля и реализации опущены. Платформенные объявления сгруппированы по исходным файлам. Сценарии использования приведены в руководстве и карте API.

pkg/api

capabilities.go

Источник: pkg/api/capabilities.go.

```
type Grant struct {
        Module          string          `yaml:"module" json:"module"`
        Operations      []string        `yaml:"operations" json:"operations"`
        Sources         []string        `yaml:"sources" json:"sources,omitempty"`
        Scopes          []string        `yaml:"scopes" json:"scopes,omitempty"`
        Resources       []string        `yaml:"resources" json:"resources,omitempty"`
}
```

configuration.go

Источник: pkg/api/configuration.go.

```
type ConfigKey struct {
        Product         string  `json:"product"`
        PrincipalID     string  `json:"principal_id"`
        Plugin          string  `json:"plugin"`
        Resource        string  `json:"resource"`
        RequestID       string  `json:"request_id"`
}

type ConfigApplyRequest struct {
        Key                     ConfigKey       `json:"key"`
        ExpectedRevision        string          `json:"expected_revision"`
        SchemaVersion           string          `json:"schema_version"`
```

```
        Document        string  `json:"document"`
}

type ConfigApplyInput struct {
        RequestID               string  `json:"request_id"`
        ExpectedRevision        string  `json:"expected_revision"`
        SchemaVersion           string  `json:"schema_version"`
        Document                string  `json:"document"`
}

type ConfigValidateInput struct {
        SchemaVersion   string  `json:"schema_version"`
        Document        string  `json:"document"`
}

func (r ConfigApplyRequest) Digest() string

type ConfigState string

const (
        ConfigAccepted          ConfigState     = "accepted"
        ConfigApplying          ConfigState     = "applying"
        ConfigSaved             ConfigState     = "saved"
        ConfigApplied           ConfigState     = "applied"
        ConfigRejected          ConfigState     = "rejected"
        ConfigOutcomeUnknown    ConfigState     = "outcome_unknown"
)

type ConfigResult struct {
        Key                     ConfigKey       `json:"key"`
        Digest                  string          `json:"digest"`
        State                   ConfigState     `json:"state"`
        PreviousRevision        string          `json:"previous_revision"`
        Revision                string          `json:"revision,omitempty"`
        ActiveRevision          string          `json:"active_revision,omitempty"`
        RestartRequired         bool            `json:"restart_required"`
        ErrorCode               string          `json:"error_code,omitempty"`
}

type ConfigResource struct {
        ID              string  `json:"id"`
        SchemaVersion   string  `json:"schema_version"`
}

type ConfigSnapshot struct {
        Resource        string  `json:"resource"`
        SchemaVersion   string  `json:"schema_version"`
        Document        string  `json:"document"`
        Revision        string  `json:"revision"`
        ActiveRevision  string  `json:"active_revision"`
}

type ConfigValidateRequest struct {
        Resource        string  `json:"resource"`
        SchemaVersion   string  `json:"schema_version"`
        Document        string  `json:"document"`
}

type ConfigIssue struct {
        Field   string  `json:"field"`
        Code    string  `json:"code"`
        Message string  `json:"message"`
}

type ConfigValidation struct {
        Valid           bool            `json:"valid"`
        Errors          []ConfigIssue   `json:"errors"`
        Warnings        []ConfigIssue   `json:"warnings"`
        RestartRequired bool            `json:"restart_required"`
}

type ConfigReconcileRequest struct {
        Key     ConfigKey       `json:"key"`
        Digest  string          `json:"digest"`
}
```

discovery_config.go

Источник: pkg/api/discovery_config.go.

```
type DiscoveryConfig struct {
        Enabled         bool                    `yaml:"enabled" json:"enabled"`
        Executable      string                  `yaml:"executable,omitempty" json:"-"`
        Scopes          []DiscoveryScope        `yaml:"scopes" json:"scopes"`
}

func (c DiscoveryConfig) Validate() error
```

errors.go

Источник: pkg/api/errors.go.

```
type Problem struct {
        Code            string  `json:"code"`
        Message         string  `json:"message"`
        Retryable       bool    `json:"retryable"`
}

type ErrorResponse struct {
        Error Problem `json:"error"`
}
```

inventory.go

Источник: pkg/api/inventory.go.

```
type InventorySection struct {
        Name            string          `json:"name"`
        Source          string          `json:"source"`
        ObservedAt      time.Time       `json:"observed_at"`
        Data            json.RawMessage `json:"data"`
        Complete        bool            `json:"complete"`
        Error           string          `json:"error,omitempty"`
}

type Environment struct {
        ID                      string          `json:"id"`
        BackendID               string          `json:"backend_id"`
        HostID                  string          `json:"host_id"`
        Name                    string          `json:"name"`
        Kind                    string          `json:"kind"`
        IdentityRevision        string          `json:"identity_revision"`
        IdentityQuality         string          `json:"identity_quality"`
        State                   string          `json:"state"`
        RawState                string          `json:"raw_state,omitempty"`
        Autostart               *bool           `json:"autostart"`
        VCPUs                   *uint64         `json:"vcpus"`
        MemoryBytes             *uint64         `json:"memory_bytes"`
        ObservedAt              time.Time       `json:"observed_at"`
        Source                  string          `json:"source"`
        Complete                bool            `json:"complete"`
        Errors                  []string        `json:"errors"`
}

type StartResult struct {
        EnvironmentID           string          `json:"environment_id"`
        CommandSucceeded        bool            `json:"command_succeeded"`
        RunningConfirmed        bool            `json:"running_confirmed"`
        ObservedState           string          `json:"observed_state"`
        ObservedAt              time.Time       `json:"observed_at"`
}

type EnvironmentSpec struct {
        Name            string          `json:"name"`
        Kind            string          `json:"kind"`
        TemplateID      string          `json:"template_id"`
```

```
        VCPUs           *uint64         `json:"vcpus,omitempty"`
        MemoryBytes     *uint64         `json:"memory_bytes,omitempty"`
        StoragePool     string          `json:"storage_pool,omitempty"`
        Networks        []string        `json:"networks,omitempty"`
}
```

jobs.go

Источник: pkg/api/jobs.go.

```
type JobRequest struct {
        Module          string          `json:"module"`
        Operation       string          `json:"operation"`
        Scope           string          `json:"scope"`
        Resource        string          `json:"resource,omitempty"`
        Parameters      json.RawMessage `json:"parameters"`
        TimeoutSeconds  int             `json:"timeout_seconds"`
        IdempotencyKey  string          `json:"idempotency_key"`
}

type JobSubmission struct {
        Job     Job     `json:"job"`
        Reused  bool    `json:"reused"`
}

type Job struct {
        ID              string          `json:"id"`
        Module          string          `json:"module"`
        Operation       string          `json:"operation"`
        Scope           string          `json:"scope"`
        Resource        string          `json:"resource,omitempty"`
        State           string          `json:"state"`
        CreatedAt       time.Time       `json:"created_at"`
        StartedAt       *time.Time      `json:"started_at,omitempty"`
        FinishedAt      *time.Time      `json:"finished_at,omitempty"`
        ExpiresAt       *time.Time      `json:"expires_at,omitempty"`
        CancelRequested bool            `json:"cancel_requested"`
        Error           string          `json:"error,omitempty"`
        ErrorCode       string          `json:"error_code,omitempty"`
        HasResult       bool            `json:"has_result"`
}

type JobOutput struct {
        SchemaVersion   int                     `json:"schema_version"`
        Metadata        json.RawMessage         `json:"metadata"`
        Items           []json.RawMessage       `json:"items"`
}

type ResultPage struct {
        JobID           string                  `json:"job_id"`
        SchemaVersion   int                     `json:"schema_version"`
        Metadata        json.RawMessage         `json:"metadata"`
        Items           []json.RawMessage       `json:"items"`
        NextCursor      string                  `json:"next_cursor,omitempty"`
        SHA256          string                  `json:"sha256"`
        TotalItems      int                     `json:"total_items"`
}

type DiscoveryParameters struct {
        CIDR            string          `json:"cidr,omitempty"`
        Exclusions      []string        `json:"exclusions,omitempty"`
        Rate            int             `json:"requests_per_second,omitempty"`
}

type DiscoveryScope struct {
        ID              string          `yaml:"id" json:"id"`
        CIDR            string          `yaml:"cidr" json:"cidr"`
        Interface       string          `yaml:"interface" json:"interface"`
        Exclusions      []string        `yaml:"exclusions" json:"exclusions"`
        MaxRate         int             `yaml:"max_rate" json:"max_rate"`
}

type DiscoveryEvidence struct {
        Address         string          `json:"address"`
```

```
        MAC             string          `json:"mac,omitempty"`
        Interface       string          `json:"interface"`
        Mechanism       string          `json:"mechanism"`
        State           string          `json:"state"`
        Names           []string        `json:"names,omitempty"`
        ObservedAt      time.Time       `json:"observed_at"`
}
```

logs.go

Источник: pkg/api/logs.go.

```
type Event struct {
        ContentHash     string  `json:"content_hash"`
        RawDataBase64   string  `json:"raw_data_base64,omitempty"`
        LineBytes       int64   `json:"line_bytes"`
        Discarded       bool    `json:"discarded,omitempty"`
        FinalPartial    bool    `json:"final_partial,omitempty"`
        LineOffset      int64   `json:"line_offset"`
        LineNo          int     `json:"line_no"`

        RequestID       string  `json:"request_id"`
        ResponseID      string  `json:"response_id"`
        TraceID         string  `json:"trace_id"`
        SpanID          string  `json:"span_id"`
        SessionID       string  `json:"session_id"`

        ParseOK         bool    `json:"parse_ok"`
        ParseError      string  `json:"parse_error,omitempty"`
        RawLine         string  `json:"raw_line,omitempty"`

        Fields  map[string]string       `json:"fields"`
}

type ReadRequest struct {
        Cursor          string  `json:"cursor,omitempty"`
        Mode            string  `json:"mode"`
        MaxLines        int     `json:"max_lines,omitempty"`
        MaxBytes        int64   `json:"max_bytes,omitempty"`
}

type ReadResponse struct {
        SchemaVersion   int     `json:"schema_version"`
        SourceID        string  `json:"source_id"`
        SourceKind      string  `json:"source_kind"`
        Reader          string  `json:"reader"`
        Timezone        string  `json:"timezone"`
        GenerationID    string  `json:"generation_id"`
        FromCursor      string  `json:"from_cursor"`
        NextCursor      string  `json:"next_cursor"`
        Events          []Event `json:"events"`
        Encoding        string  `json:"encoding"`
        DataBase64      string  `json:"data_base64,omitempty"`
        LinesReturned   int     `json:"lines_returned"`
        PayloadBytes    int64   `json:"payload_bytes"`
        HasMore         bool    `json:"has_more"`
        WaitingForLine  bool    `json:"waiting_for_line"`
        Rotated         bool    `json:"rotated"`
        Truncated       bool    `json:"truncated"`
}

type Source struct {
        ID              string  `json:"id"`
        Kind            string  `json:"kind"`
        Reader          string  `json:"reader"`
        Parser          string  `json:"parser"`
        MaxLines        int     `json:"max_lines"`
        MaxBytes        int64   `json:"max_bytes"`
}
```

plugins.go

Источник: pkg/api/plugins.go.

```
type PluginInfo struct {
        ID                      string          `json:"id"`
        Version                 string          `json:"version"`
        Protocol                uint16          `json:"protocol"`
        SchemaVersion           int             `json:"schema_version"`
        State                   string          `json:"state"`
        Generation              int             `json:"generation"`
        Operations              []string        `json:"operations"`
        ResourceOperations      []string        `json:"resource_operations"`
}
```

provider_resource.go

Источник: pkg/api/provider_resource.go.

```
type Resource struct {
        ID              string          `json:"id"`
        Type            string          `json:"type"`
        Provider        string          `json:"provider"`
        Scope           string          `json:"scope"`
        NativeID        string          `json:"native_id"`
        Revision        string          `json:"revision,omitempty"`
        Configuration   json.RawMessage `json:"configuration"`
        Observed        json.RawMessage `json:"observed"`
        Operations      []string        `json:"operations"`
        Capabilities    map[string]bool `json:"capabilities"`
        ObservedAt      time.Time       `json:"observed_at"`
        JobID           string          `json:"job_id,omitempty"`
        ManagementOwner string          `json:"management_owner"`
}
```

transport.go

Источник: pkg/api/transport.go.

```
const TransportVersion = 1

func ValidPluginID(id string) bool

type PluginCall struct {
        Method  string          `json:"method"`
        Body    json.RawMessage `json:"body"`
}

type StreamCall struct {
        Method          string  `json:"method"`
        TransferID      string  `json:"transfer_id"`
        RequestID       string  `json:"request_id"`
}

func ValidPluginMethod(method string) bool
```

version.go

Источник: pkg/api/version.go.

```
const AgentVersion = "2.1.0"

const AgentReleaseDate = ""
```

pkg/pluginipc

client.go

Источник: pkg/pluginipc/client.go.

```
var ErrRequestTooLarge = errors.New("IPC request exceeds transport limit")

type Client struct {
}

func (c *Client) String() string

func (c *Client) GoString() string

func NewClient(endpoint, token, id string, connections int) *Client

func (c *Client) Call(parent context.Context, method string, body any) (Response, error)

func (c *Client) Close()
```

context.go

Источник: pkg/pluginipc/context.go.

```
type Execution struct {
        RequestID       string  `json:"request_id,omitempty"`
        JobID           string  `json:"job_id,omitempty"`
        PrincipalID     string  `json:"principal_id,omitempty"`
        Product         string  `json:"product,omitempty"`
}

func WithExecution(ctx context.Context, info Execution) context.Context

func ExecutionFromContext(ctx context.Context) Execution
```

logs.go

Источник: pkg/pluginipc/logs.go.

```
type LogsSettings struct {
        Config                  string  `json:"config"`
        Concurrency             int     `json:"concurrency"`
        ValidateOnly            bool    `json:"validate_only"`
        ConfigurationEnabled    bool    `json:"configuration_enabled,omitempty"`
}

type LogHTTPRequest struct {
        Source          string          `json:"source"`
        Operation       string          `json:"operation"`
        Method          string          `json:"method"`
        Body            json.RawMessage `json:"body,omitempty"`
}

type LogHTTPResponse struct {
        Status  int             `json:"status"`
        Body    json.RawMessage `json:"body"`
}
```

management.go

Источник: pkg/pluginipc/management.go.

```
const ManagementVersion = 1

const PermissionsMethod = "permissions.check"

const CompatibilityMethod = "compatibility.describe"

type PermissionQuery struct {
        Operation       string  `json:"operation"`
        Scope           string  `json:"scope,omitempty"`
```

```
        Resource        string  `json:"resource,omitempty"`
}

type PermissionDecision struct {
        Decision        string  `json:"decision"`
        Reason          string  `json:"reason,omitempty"`
}

func (d PermissionDecision) Valid() bool

type Compatibility struct {
        ManagementVersion               int             `json:"management_version"`
        MinAgentVersion                 string          `json:"min_agent_version"`
        MaxAgentVersionExclusive        string          `json:"max_agent_version_exclusive"`
        APIVersion                      int             `json:"api_version"`
        IPCVersion                      uint16          `json:"ipc_version"`
        AdminContract                   int             `json:"admin_contract"`
        Capabilities                    []string        `json:"capabilities"`
        SettingsSchemaVersion           int             `json:"settings_schema_version"`
        StateSchemaVersion              string          `json:"state_schema_version"`
}

type HostCompatibility struct {
        AgentVersion    string          `json:"agent_version"`
        APIVersion      int             `json:"api_version"`
        IPCVersion      uint16          `json:"ipc_version"`
        AdminContract   int             `json:"admin_contract"`
        Capabilities    []string        `json:"capabilities"`
}

type ManagementHandler interface {
        Handler
        CheckPermissions(context.Context, PermissionQuery) (PermissionDecision, error)
        DescribeCompatibility(context.Context) (Compatibility, error)
}

func (c Compatibility) Validate() error

func (c Compatibility) Supports(h HostCompatibility) bool

func (h HostCompatibility) Validate() error

func EqualCompatibility(a, b Compatibility) bool

func ValidateManagement(ctx context.Context, h Handler) error
```

protocol.go

Источник: pkg/pluginipc/protocol.go.

```
const Version uint16 = 1

const MaxRequest = 1 << 20

const MaxResponse = 256 << 20

type Identity struct {
        ManagementVersion       int             `json:"management_version,omitempty"`
        StagedActivation        bool            `json:"staged_activation,omitempty"`
        IngressVersion          int             `json:"ingress_version,omitempty"`
        StreamingVersion        int             `json:"streaming_version,omitempty"`
        MaxStreamRequestBytes   int64           `json:"max_stream_request_bytes,omitempty"`
        MaxStreamResponseBytes  int64           `json:"max_stream_response_bytes,omitempty"`
        StreamChunkBytes        int             `json:"stream_chunk_bytes,omitempty"`
        ID                      string          `json:"id"`
        Version                 string          `json:"version"`
        Protocol                uint16          `json:"protocol"`
        Operations              []string        `json:"operations"`
        ResourceOperations      []string        `json:"resource_operations,omitempty"`
        SchemaVersion           int             `json:"schema_version"`
}

type Problem struct {
        Code    string  `json:"code"`
```

```
        Message string  `json:"message"`
}

func (p *Problem) Error() string

func (p *Problem) ErrorCode() string

type Response struct {
        Body    json.RawMessage `json:"body,omitempty"`
        Error   *Problem        `json:"error,omitempty"`
}

type Config struct {
        Staged          bool            `json:"staged,omitempty"`
        StreamLimits    StreamLimits    `json:"stream_limits,omitempty"`
        ID              string          `json:"id"`
        Token           string          `json:"token"`
        Settings        json.RawMessage `json:"settings"`
        Concurrency     int             `json:"concurrency"`
}

func (c Config) String() string

func (c Config) GoString() string
```

server.go

Источник: pkg/pluginipc/server.go.

```
type Handler interface {
        Identity() Identity
        Invoke(context.Context, string, json.RawMessage) Response
}

func Serve(parent context.Context, endpoint, token string, concurrency int, h Handler) error

func ServePlugin(parent context.Context, endpoint, token string, concurrency int, h Handler) error

func ServePluginWithStreamLimits(parent context.Context, endpoint, token string, concurrency int, h Handler, limits
StreamLimits) error

func LoadConfig(path string) (Config, error)
```

stream.go

Источник: pkg/pluginipc/stream.go.

```
const MaxLogicalSize int64 = 1_000_000_000_000_000

var ErrStreamingUnsupported = errors.New("plugin does not support streaming IPC")

var ErrStreamRequestLimit = errors.New("stream request exceeds limit; partial execution may have occurred")

type StreamLimits struct {
        MaxRequestBytes         int64   `json:"max_request_bytes" yaml:"max_request_bytes"`
        MaxResponseBytes        int64   `json:"max_response_bytes" yaml:"max_response_bytes"`
        ChunkBytes              int     `json:"chunk_bytes" yaml:"chunk_bytes"`
}

func (l *StreamLimits) Defaults() error

type StreamHandler interface {
        InvokeStream(context.Context, string, io.Reader, io.Writer) error
}

type Ingress struct {
        Network         string  `json:"network"`
        Address         string  `json:"address"`
        Anonymous       bool    `json:"anonymous,omitempty"`
}

type IngressHandler interface {
```

```
        InvokeIngress(context.Context, Ingress, io.Reader, io.Writer) error
}

func NewClientWithStreamLimits(endpoint, token, id string, connections int, limits StreamLimits) (*Client, error)

func (c *Client) CallStream(parent context.Context, method string, input io.Reader, output io.Writer) error

func (c *Client) CallStreamSized(parent context.Context, method string, input io.Reader, output io.Writer, size int64) error

func (c *Client) CallIngress(parent context.Context, ingress Ingress, input io.ReadCloser, output io.Writer, size int64)
error
```

transport_unix.go

Источник: pkg/pluginipc/transport_unix.go.

```
func Endpoint(dir, instance string) string

func SecureDirectory(dir string) error
```

transport_windows.go

Источник: pkg/pluginipc/transport_windows.go.

```
func Endpoint(dir, instance string) string

func SecureDirectory(dir string) error
```

pkg/pluginhost

activation.go

Источник: pkg/pluginhost/activation.go.

```
type StagedPlugin interface {
        Plugin
        Activate(context.Context) error
}
```

configjournal.go

Источник: pkg/pluginhost/configjournal.go.

```
type ConfigRecord struct {
        Request api.ConfigApplyRequest  `json:"request"`
        Result  api.ConfigResult        `json:"result"`
}

type ConfigJournalData struct {
        Version         int                             `json:"version"`
        Records         map[string]ConfigRecord         `json:"records"`
        Resources       map[string]api.ConfigSnapshot   `json:"resources"`
}

type ConfigJournal struct {
}

func ConfigRecordID(key api.ConfigKey) string

func OpenConfigJournal(dir string, maxRecords, maxBytes int) (*ConfigJournal, error)

func ValidConfigResult(r api.ConfigResult) bool

func ConfigTransition(from, to api.ConfigState) bool

func (j *ConfigJournal) View() (ConfigJournalData, error)
```

```
func (j *ConfigJournal) Update(change func(*ConfigJournalData) error) error

func (j *ConfigJournal) Close() error
```

main.go

Источник: pkg/pluginhost/main.go.

```
type Plugin interface {
        pluginipc.ManagementHandler
        Close() error
}

func Main(id string, open func(json.RawMessage) (Plugin, error))
```

pkg/client

client.go

Источник: pkg/client/client.go.

```
type Client struct {
}

func NewHTTP(base, token string) (*Client, error)

func New(base, serverCA, certificate, privateKey string) (*Client, error)

type HTTPError struct {
        Status          int
        Message         string
        Code            string
        Retryable       bool
}

func (e *HTTPError) Error() string

func (c *Client) Close()

func (c *Client) Call(ctx context.Context, plugin, method string, body json.RawMessage) (pluginipc.Response, error)

func (c *Client) Plugins(ctx context.Context) ([]string, error)

func (c *Client) ReloadPlugin(ctx context.Context, plugin string) (map[string]any, error)
```

pkg/active

agent_transfer_client.go

Источник: pkg/active/agent_transfer_client.go.

```
type AgentTransferClient struct {
}

func NewAgentTransferClient(protocol, endpoint string, security Security, plugins ...string) (*AgentTransferClient, error)

func (c *AgentTransferClient) Close() error

func (c *AgentTransferClient) Transfer(ctx context.Context, _ string, request TransferRequest) (TransferResponse, error)

func (c *AgentTransferClient) Stream(ctx context.Context, plugin string, call api.StreamCall) (Result, error)

func (c *AgentTransferClient) ConfirmStreamResult(ctx context.Context, plugin, requestID string) error

func (c *AgentTransferClient) Call(ctx context.Context, plugin string, call api.PluginCall) (pluginipc.Response, error)
```

batch.go

Источник: pkg/active/batch.go.

```
func (b *Broker) PublishBatch(ctx context.Context, id string, events []Event) (BatchACK, error)

type RetryConfig struct {
        InitialDelayMillis      int     `yaml:"initial_delay_ms" json:"initial_delay_ms"`
        MaxDelayMillis          int     `yaml:"max_delay_ms" json:"max_delay_ms"`
        JitterPercent           int     `yaml:"jitter_percent" json:"jitter_percent"`
}

func (c *RetryConfig) Defaults() error

func (c RetryConfig) Delay(attempt int) time.Duration
```

broker.go

Источник: pkg/active/broker.go.

```
type Broker struct {
}

func NewBroker(stateDirectory string, agentIDs []string, maxInFlight int) (*Broker, error)

func NewBrokerWithConfig(stateDirectory string, agentIDs []string, maxInFlight int, queue QueueConfig) (*Broker, error)

func NewBrokerWithAuthenticator(stateDirectory string, auth *Authenticator, maxInFlight int, queue QueueConfig) (*Broker,
error)

func (b *Broker) Submit(agentID string, cmd Command) (string, error)

func (b *Broker) Wait(ctx context.Context, agentID, id string) (Result, error)

func (b *Broker) Execute(ctx context.Context, agentID string, cmd Command) (Result, error)

func (b *Broker) Poll(ctx context.Context, agentID string) (*Command, error)

func (b *Broker) Complete(_ context.Context, agentID string, result Result) error

func (b *Broker) Maintain(now time.Time) error

func (b *Broker) RunMaintenance(ctx context.Context)

func (b *Broker) QueueStats() QueueStats

func (b *Broker) EventQueueStats() QueueStats

func (b *Broker) Close() error
```

credentials.go

Источник: pkg/active/credentials.go.

```
type CredentialStatus struct {
        ReloadedAt              time.Time       `json:"reloaded_at"`
        CertificateExpiresAt    time.Time       `json:"certificate_expires_at,omitempty"`
        CertificateExpiring     bool            `json:"certificate_expiring"`
        LastError               string          `json:"last_error,omitempty"`
        UsingPrevious           bool            `json:"using_previous"`
}

type CredentialReporter interface{ CredentialStatus() CredentialStatus }
```

enrollment.go

Источник: pkg/active/enrollment.go.

```
type EnrollmentConfig struct {
        Enabled                 bool            `yaml:"enabled"`
        PermitsFile             string          `yaml:"permits_file"`
        ClientCA                string          `yaml:"client_ca"`
        ClientCAKey             string          `yaml:"client_ca_key"`
        CertificateSeconds      int             `yaml:"certificate_seconds"`
        MaxNodes                int             `yaml:"max_nodes"`
        RevokedAgentIDs         []string        `yaml:"revoked_agent_ids"`
}

type EnrollmentPermit struct {
        TokenSHA256     string          `json:"token_sha256"`
        AgentID         string          `json:"agent_id"`
        ExpiresAt       time.Time       `json:"expires_at"`
        AllowedNetworks []string        `json:"allowed_networks"`
}

type EnrollmentRequest struct {
        Identity        InstallationIdentity    `json:"identity"`
        Hostname        string                  `json:"hostname"`
        OS              string                  `json:"os"`
        Architecture    string                  `json:"architecture"`
        Version         string                  `json:"version"`
        TokenSHA256     string                  `json:"token_sha256,omitempty"`
        CSR             string                  `json:"csr,omitempty"`
}

type EnrollmentResponse struct {
        AgentID         string  `json:"agent_id"`
        Auth            string  `json:"auth"`
        Certificate     string  `json:"certificate,omitempty"`
}

type Enroller struct {
}

func NewEnroller(dir string, auth *Authenticator, config EnrollmentConfig) (*Enroller, error)

func (e *Enroller) PendingJournalError() error

func (e *Enroller) ReloadPeers(peers []Peer, revokedIDs []string, grace time.Duration, maxPrevious int) error

func (e *Enroller) Handler() http.Handler

type EnrollmentClientConfig struct {
        Endpoint                string  `yaml:"endpoint"`
        CAFile                  string  `yaml:"ca_file"`
        BootstrapTokenFile      string  `yaml:"bootstrap_token_file"`
        CredentialsDirectory    string  `yaml:"credentials_directory"`
        Auth                    string  `yaml:"auth"`
        TimeoutSeconds          int     `yaml:"timeout_seconds"`
}

func Enroll(ctx context.Context, config EnrollmentClientConfig, identity InstallationIdentity, version string)
(EnrollmentResponse, error)

func (e *Enroller) Peers() []Peer
```

events.go

Источник: pkg/active/events.go.

```
func (b *Broker) Publish(_ context.Context, agentID string, event Event) error

func (b *Broker) Events(agentID string) ([]Event, error)

type Publisher struct {
}

func OpenPublisher(dir string) (*Publisher, error)

func OpenPublisherWithConfig(dir string, config QueueConfig) (*Publisher, error)
```

```
func (p *Publisher) Close() error

func (p *Publisher) Enqueue(plugin string, body json.RawMessage) (string, error)

func (p *Publisher) Flush(ctx context.Context, exchange Exchange, agentID string) error

func (p *Publisher) QueueStats() QueueStats
```

failover.go

Источник: pkg/active/failover.go.

```
type Target struct {
        Endpoint        string          `yaml:"endpoint" json:"endpoint"`
        Protocol        string          `yaml:"protocol" json:"protocol"`
        Priority        int             `yaml:"priority" json:"priority"`
        Security        Security        `yaml:"security" json:"security"`
}

type FailoverConfig struct {
        DeliveryDomain          string  `yaml:"delivery_domain" json:"delivery_domain"`
        FailbackSeconds         int     `yaml:"failback_seconds" json:"failback_seconds"`
        HandshakeSeconds        int     `yaml:"handshake_seconds" json:"handshake_seconds"`
}

func NewFailoverExchange(targets []Target, config FailoverConfig, stateDir string) (Exchange, error)
```

grpc.go

Источник: pkg/active/grpc.go.

```
func RegisterGRPC(registrar grpc.ServiceRegistrar, broker *Broker, auth *Authenticator)
```

heartbeat.go

Источник: pkg/active/heartbeat.go.

```
type PeerState struct {
        AgentID         string          `json:"agent_id"`
        State           string          `json:"state"`
        LastContact     time.Time       `json:"last_contact"`
        LastDelivery    time.Time       `json:"last_delivery"`
        Heartbeat       Heartbeat       `json:"heartbeat"`
        QueueGrowing    bool            `json:"queue_growing"`
}

func (b *Broker) SetHeartbeatTimeoutSeconds(seconds int) error

func (b *Broker) Heartbeat(_ context.Context, heartbeat Heartbeat) error

func (b *Broker) PeerStatus(agentID string, now time.Time) (PeerState, error)
```

hello.go

Источник: pkg/active/hello.go.

```
func (b *Broker) NegotiateTransfers(ctx context.Context, owner string, supported bool) error

type Hello struct {
        Transfers       *TransferConfig `json:"transfers,omitempty"`
        Version         int             `json:"version"`
        StoreID         string          `json:"store_id"`
        DeliveryDomain  string          `json:"delivery_domain"`
}

func (b *Broker) SetDeliveryDomain(domain string) error
```

```
func (b *Broker) Hello(_ context.Context, agentID string) (Hello, error)
```

identity.go

Источник: pkg/active/identity.go.

```
type InstallationIdentity struct {
        InstallationID  string          `json:"installation_id"`
        AgentID         string          `json:"agent_id"`
        MachineDigest   string          `json:"machine_digest"`
        CreatedAt       time.Time       `json:"created_at"`
}

func EnsureIdentity(stateDirectory, agentID, machineID string) (InstallationIdentity, error)
```

inbox.go

Источник: pkg/active/inbox.go.

```
type Inbox struct {
}

func OpenInbox(dir string) (*Inbox, error)

func OpenInboxWithConfig(dir string, config QueueConfig) (*Inbox, error)

func (i *Inbox) Close() error

func (i *Inbox) LookupResult(cmd Command) (Result, bool, error)

func (i *Inbox) ReceiptExists(cmd Command) (bool, error)

func (i *Inbox) Process(ctx context.Context, cmd Command, invoke func(context.Context, Command) Result) (Result, error)

func (i *Inbox) ACK(id string) error

func (i *Inbox) ACKWithCleanup(id string, cleanup func(Result) error) error

func (i *Inbox) Maintain(now time.Time) error

func (i *Inbox) RunMaintenance(ctx context.Context)

func (i *Inbox) QueueStats() QueueStats
```

inbox_completion.go

Источник: pkg/active/inbox_completion.go.

```
type PendingCompletion struct {
        Result          Result
        InputID         string
        Stream          bool
        RemoteACK       bool
}

func (i *Inbox) StageCompletion(cmd Command) (*PendingCompletion, error)

func (i *Inbox) RecordRemoteACK(id string) error

func (i *Inbox) UnacknowledgedIDs() []string

func (i *Inbox) PendingCompletion(id string) (*PendingCompletion, error)
```

inspect.go

Источник: pkg/active/inspect.go.

```
type DeliveryInspection struct {
        TransferID      string  `json:"transfer_id,omitempty"`
        ID              string  `json:"id"`
        State           string  `json:"state"`
        Status          int     `json:"status,omitempty"`
        Bytes           int64   `json:"bytes"`
        BodyOmitted     bool    `json:"body_omitted"`
}

func InspectDelivery(dir, id string) (DeliveryInspection, error)

func InspectTransfer(dir, id string) (TransferInfo, error)

func InspectTransfers(dir string, config TransferConfig) (TransferStats, error)

func InspectQueue(dir string, config QueueConfig) (QueueStats, error)
```

metrics.go

Источник: pkg/active/metrics.go.

```
const PrometheusContentType = "text/plain; version=0.0.4; charset=utf-8"

func EncodeMetricsBody(operation string, status int, contentType string, body []byte) []byte
```

passive.go

Источник: pkg/active/passive.go.

```
type AgentRequest struct {
        Transfer        *TransferRequest        `json:"transfer,omitempty"`
        Stream          *api.StreamCall         `json:"stream,omitempty"`
        Operation       string                  `json:"operation"`
        Plugin          string                  `json:"plugin,omitempty"`
        Call            *api.PluginCall         `json:"call,omitempty"`
}

type AgentResponse struct {
        Status  int             `json:"status"`
        Body    json.RawMessage `json:"body"`
}

func RegisterAgentGRPC(registrar grpc.ServiceRegistrar, handler http.Handler)

type AgentGRPCClient struct{}

func NewAgentGRPCClient(endpoint string, security Security) (*AgentGRPCClient, error)

func (c *AgentGRPCClient) Close() error

func (c *AgentGRPCClient) Invoke(ctx context.Context, request AgentRequest) (AgentResponse, error)
```

protocol.go

Источник: pkg/active/protocol.go.

```
const (
        MaxCommand      = 1 << 20
        MaxResult       = (256 << 20) + (64 << 10)
        PollWait        = 20 * time.Second
)

type Command struct {
        Stream          *api.StreamCall `json:"stream,omitempty"`
        ID              string          `json:"id"`
        Operation       string          `json:"operation"`
        Plugin          string          `json:"plugin,omitempty"`
        Call            *api.PluginCall `json:"call,omitempty"`
```

```
        Deadline        time.Time       `json:"deadline"`
}

type Result struct {
        TransferID      string          `json:"transfer_id,omitempty"`
        ID              string          `json:"id"`
        Status          int             `json:"status"`
        Body            json.RawMessage `json:"body,omitempty"`
}

type PollRequest struct {
        AgentID string `json:"agent_id"`
}

type PollResponse struct {
        Command *Command `json:"command,omitempty"`
}

type CompleteRequest struct {
        AgentID string  `json:"agent_id"`
        Result  Result  `json:"result"`
}

type Event struct {
        TransferID      string          `json:"transfer_id,omitempty"`
        ID              string          `json:"id"`
        Plugin          string          `json:"plugin"`
        CreatedAt       time.Time       `json:"created_at"`
        Body            json.RawMessage `json:"body,omitempty"`
}

type PublishRequest struct {
        AgentID string  `json:"agent_id"`
        Event   Event   `json:"event"`
}

type Heartbeat struct {
        AgentID         string                  `json:"agent_id"`
        Version         string                  `json:"version"`
        Ready           bool                    `json:"ready"`
        Queue           QueueStats              `json:"queue"`
        PluginStates    map[string]string       `json:"plugin_states"`
}

type BatchRequest struct {
        AgentID string  `json:"agent_id"`
        Events  []Event `json:"events"`
}

type BatchRejection struct {
        ID              string  `json:"id"`
        Code            string  `json:"code"`
        Retryable       bool    `json:"retryable"`
}

type BatchACK struct {
        Accepted        []string                `json:"accepted"`
        Rejected        []BatchRejection        `json:"rejected"`
}

var ErrOutcomeUnknown = errors.New("active outcome unknown; operation was not retried")

var ErrBusy = errors.New("active transport concurrency limit")

var ErrUnauthorized = errors.New("active unauthorized")

var ErrStaleResult = errors.New("active result unknown or already completed")

var ErrTransfersUnsupported = errors.New("agent does not advertise large-transfer support; operation not invoked")

type Exchange interface {
        Hello(context.Context, string) (Hello, error)
        Poll(context.Context, string) (*Command, error)
        Complete(context.Context, string, Result) error
        Publish(context.Context, string, Event) error
        Heartbeat(context.Context, Heartbeat) error
        PublishBatch(context.Context, string, []Event) (BatchACK, error)
```

```
        Close() error
}
```

publisher_transfer.go

Источник: pkg/active/publisher_transfer.go.

```
func (p *Publisher) EnableTransfers(config TransferConfig) error

func (p *Publisher) RunMaintenance(ctx context.Context)

func (p *Publisher) EnqueueStream(ctx context.Context, agentID, plugin string, source io.Reader) (string, error)

func (p *Publisher) TransferStatus(owner, id string) (TransferInfo, error)

func (p *Publisher) EnqueueStoredTransfer(owner, plugin, id string) (string, error)

func (p *Publisher) EnqueueTransfer(plugin string, info TransferInfo) (string, error)
```

queue.go

Источник: pkg/active/queue.go.

```
type QueueConfig struct {
        BatchMaxRecords         int     `yaml:"batch_max_records" json:"batch_max_records"`
        BatchMaxBytes           int     `yaml:"batch_max_bytes" json:"batch_max_bytes"`
        SendConcurrency         int     `yaml:"send_concurrency" json:"send_concurrency"`
        BytesPerSecond          int64   `yaml:"bytes_per_second" json:"bytes_per_second"`
        SendJitterPercent       int     `yaml:"send_jitter_percent" json:"send_jitter_percent"`
        MaxBytes                int64   `yaml:"max_bytes" json:"max_bytes"`
        MaxRecords              int     `yaml:"max_records" json:"max_records"`
        RetentionSeconds        int     `yaml:"retention_seconds" json:"retention_seconds"`
        MaintenanceSeconds      int     `yaml:"maintenance_seconds" json:"maintenance_seconds"`
        SendIntervalMillis      int     `yaml:"send_interval_ms" json:"send_interval_ms"`
        MaxSendPerFlush         int     `yaml:"max_send_per_flush" json:"max_send_per_flush"`
}

func (c *QueueConfig) Defaults() error

type QuotaError struct {
        Resource        string
        Limit           int64
}

func (e *QuotaError) Error() string

func (e *QuotaError) Is(target error) bool

var ErrResultExpired = errors.New("confirmed result expired; command was not replayed")

type QueueStats struct {
        PendingRecords          int             `json:"pending_records"`
        OldestPendingAgeSeconds float64         `json:"oldest_pending_age_seconds"`
        Redeliveries            uint64          `json:"redeliveries"`
        Records                 int             `json:"records"`
        Bytes                   int64           `json:"bytes"`
        MaxBytes                int64           `json:"max_bytes"`
        MaxRecords              int             `json:"max_records"`
        Oldest                  time.Time       `json:"oldest,omitempty"`
        LastACK                 time.Time       `json:"last_ack,omitempty"`
        Errors                  uint64          `json:"errors"`
        Deliveries              uint64          `json:"deliveries"`
}
```

rest.go

Источник: pkg/active/rest.go.

```
const RESTPrefix = "/api/active/v1/"

func RESTHandler(b *Broker, auth *Authenticator) http.Handler

func NewExchange(protocol, endpoint string, security Security) (Exchange, error)
```

security.go

Источник: pkg/active/security.go.

```
type Peer struct {
        AgentID                 string          `yaml:"agent_id"`
        TokenSHA256             string          `yaml:"token_sha256"`
        CertificateSHA256       string          `yaml:"certificate_sha256"`
        AllowedNetworks         []string        `yaml:"allowed_networks"`
}

type Authenticator struct {
}

func NewAuthenticator(peers []Peer, bearer bool) (*Authenticator, error)

func (a *Authenticator) ReloadPeers(peers []Peer, grace time.Duration, maxPrevious int) error

func NewBootstrapAuthenticator(bearer bool) *Authenticator

func (a *Authenticator) AddPeer(peer Peer) error

type Security struct {
        CertificateWarningSeconds       int     `yaml:"certificate_warning_seconds" json:"certificate_warning_seconds"`
        CAFile                          string  `yaml:"ca_file" json:"ca_file"`
        Certificate                     string  `yaml:"certificate" json:"certificate"`
        PrivateKey                      string  `yaml:"private_key" json:"private_key"`
        BearerFile                      string  `yaml:"bearer_file" json:"bearer_file"`
}

func (s Security) ClientCredentials(secure bool) (*tls.Config, string, error)

func ServerTLS(certificate, privateKey, clientCA string, bearer bool) (*tls.Config, error)
```

transfer.go

Источник: pkg/active/transfer.go.

```
const Petabyte int64 = 1_000_000_000_000_000

func NewTransferID() (string, error)

type TransferConfig struct {
        RetentionSeconds        int     `yaml:"retention_seconds" json:"retention_seconds"`
        MaintenanceSeconds      int     `yaml:"maintenance_seconds" json:"maintenance_seconds"`
        Enabled                 bool    `yaml:"enabled" json:"enabled"`
        MaxRequestBytes         int64   `yaml:"max_request_bytes" json:"max_request_bytes"`
        MaxResponseBytes        int64   `yaml:"max_response_bytes" json:"max_response_bytes"`
        MaxBatchBytes           int64   `yaml:"max_batch_bytes" json:"max_batch_bytes"`
        MaxStoreBytes           int64   `yaml:"max_store_bytes" json:"max_store_bytes"`
        MaxTransfers            int     `yaml:"max_transfers" json:"max_transfers"`
        ChunkBytes              int     `yaml:"chunk_bytes" json:"chunk_bytes"`
}

func (c *TransferConfig) Defaults() error

type TransferInfo struct {
        CreatedAt       time.Time       `json:"created_at"`
        Retired         bool            `json:"retired,omitempty"`
        ID              string          `json:"id"`
        Owner           string          `json:"owner"`
        Kind            string          `json:"kind"`
        Size            int64           `json:"size"`
        SHA256          string          `json:"sha256,omitempty"`
        Offset          int64           `json:"offset"`
```

```
        Ready           bool            `json:"ready"`
        ConfirmedAt     time.Time       `json:"confirmed_at,omitempty"`
}

type TransferStats struct {
        MaintenanceFailures     uint64  `json:"maintenance_failures"`
        LastMaintenanceError    string  `json:"last_maintenance_error,omitempty"`
        Bytes                   int64   `json:"bytes"`
        MaxBytes                int64   `json:"max_bytes"`
        Transfers               int     `json:"transfers"`
        MaxTransfers            int     `json:"max_transfers"`
}

type TransferStore struct {
}

func OpenTransferStore(dir string, config TransferConfig) (*TransferStore, error)

func (s *TransferStore) Begin(owner, id, kind string, size int64, sha string) (TransferInfo, error)

func (s *TransferStore) Status(owner, id string) (TransferInfo, error)

func (s *TransferStore) Append(owner, id string, offset int64, data []byte) (int64, error)

func (s *TransferStore) Finish(owner, id string, size int64, sha string) (TransferInfo, error)

func (s *TransferStore) ReadChunk(owner, id string, offset int64) ([]byte, TransferInfo, error)

func (s *TransferStore) Open(owner, id string) (io.ReadCloser, TransferInfo, error)

func (s *TransferStore) Retain(owner, id string) error

func (s *TransferStore) Confirm(owner, id string) error

func (s *TransferStore) Stats() TransferStats

func (s *TransferStore) Close() error

func (i TransferInfo) String() string
```

transfer_maintenance.go

Источник: pkg/active/transfer_maintenance.go.

```
func (s *TransferStore) HoldRequest(id string) func()

func (s *TransferStore) ConfirmRequest(owner, id string) error

func (s *TransferStore) Maintain(now time.Time) error

func (s *TransferStore) DiscardIncompleteResponse(owner, id string) error

func (s *TransferStore) DiscardUnreferencedResponse(owner, id, referencedID string) error

func (s *TransferStore) RunMaintenance(ctx context.Context)
```

transfer_transport.go

Источник: pkg/active/transfer_transport.go.

```
type TransferRequest struct {
        Plugin          string  `json:"plugin,omitempty"`
        RequestID       string  `json:"request_id,omitempty"`
        Action          string  `json:"action"`
        ID              string  `json:"id"`
        Kind            string  `json:"kind,omitempty"`
        Size            int64   `json:"size,omitempty"`
        SHA256          string  `json:"sha256,omitempty"`
        Offset          int64   `json:"offset,omitempty"`
        Data            []byte  `json:"data,omitempty"`
}
```

```
type TransferResponse struct {
        Info            TransferInfo    `json:"info"`
        Offset          int64           `json:"offset"`
        ChunkBytes      int             `json:"chunk_bytes"`
        Data            []byte          `json:"data,omitempty"`
}

type TransferExchange interface {
        Transfer(context.Context, string, TransferRequest) (TransferResponse, error)
}

func (b *Broker) EnableTransfers(config TransferConfig) error

func (b *Broker) Transfer(ctx context.Context, owner string, request TransferRequest) (TransferResponse, error)

func (s *TransferStore) Handle(ctx context.Context, owner string, request TransferRequest) (TransferResponse, error)

func UploadTransfer(ctx context.Context, exchange TransferExchange, owner string, local *TransferStore, id string) error

func DownloadTransfer(ctx context.Context, exchange TransferExchange, owner string, local *TransferStore, id, kind string)
(TransferInfo, error)
```

transfer_writer.go

Источник: pkg/active/transfer_writer.go.

```
type TransferWriter struct {
}

func (s *TransferStore) NewWriter(owner, id, kind string) (*TransferWriter, error)

func (w *TransferWriter) Write(p []byte) (int, error)

func (w *TransferWriter) Finish() (TransferInfo, error)

func StoreTransfer(ctx context.Context, store *TransferStore, owner, id, kind string, source io.Reader) (TransferInfo, error)
```

pkg/admin

client.go

Источник: pkg/admin/client.go.

```
func ReadInstallation(path string) (Installation, error)

func Send(ctx context.Context, installation Installation, call Call) (Reply, error)

func Discover(ctx context.Context, path string) (Installation, error)
```

management.go

Источник: pkg/admin/management.go.

```
func CheckPermissions(ctx context.Context, i Installation, owner, plugin string, q PermissionCheck) (PermissionResult, error)

func Inventory(ctx context.Context, i Installation, target *pluginipc.HostCompatibility) (PluginInventory, error)

func DecodeTarget(data []byte) (pluginipc.HostCompatibility, error)
```

types.go

Источник: pkg/admin/types.go.

```
const Contract = "com.astraverge.agent-admin/1"

const MaxMessage = 2 << 20
```

```
type Endpoint struct {
        Transport       string  `json:"transport"`
        Address         string  `json:"address"`
        Auth            string  `json:"auth"`
        ServerCAFile    string  `json:"server_ca_file,omitempty"`
}

type Installation struct {
        Contract        string          `json:"contract"`
        InstallationID  string          `json:"installation_id"`
        AgentVersion    string          `json:"agent_version"`
        APIVersion      int             `json:"api_version"`
        IPCVersion      int             `json:"ipc_version"`
        ServiceState    string          `json:"service_state"`
        AdminEndpoint   string          `json:"admin_endpoint"`
        ServerIdentity  string          `json:"server_identity"`
        AdminAuth       string          `json:"admin_auth"`
        Capabilities    []string        `json:"capabilities"`
        ClientEndpoints []Endpoint      `json:"client_endpoints"`
}

type Credential struct {
        Type    string  `json:"type"`
        SHA256  string  `json:"sha256"`
}

type Consumer struct {
        PrincipalID             string          `json:"principal_id"`
        ProductID               string          `json:"product_id"`
        Credential              Credential      `json:"credential"`
        AllowedNetworks         []string        `json:"allowed_networks"`
        Permissions             []string        `json:"permissions"`
        RequireAccessProof      bool            `json:"require_access_proof"`
}

type Files struct {
        Executable              string  `json:"executable"`
        DeveloperCertificate    string  `json:"developer_certificate"`
        Release                 string  `json:"release"`
        Settings                string  `json:"settings"`
        License                 string  `json:"license,omitempty"`
        BuildTimestamp          string  `json:"build_timestamp,omitempty"`
        BuildTrust              string  `json:"build_trust,omitempty"`
        LicenseDelegation       string  `json:"license_delegation,omitempty"`
        LicenseTrust            string  `json:"license_trust,omitempty"`
}

type LicenseBinding struct {
        IssuerID        string  `json:"issuer_id"`
        Product         string  `json:"product"`
        LicenseID       string  `json:"license_id"`
        CustomerID      string  `json:"customer_id"`
        MinimumRevision uint32  `json:"minimum_revision"`
}

type Requirements struct {
        AdminContract   int             `json:"admin_contract"`
        IPCVersion      int             `json:"ipc_version"`
        Capabilities    []string        `json:"capabilities,omitempty"`
}

type Request struct {
        Contract                        string          `json:"contract"`
        OperationID                     string          `json:"operation_id"`
        Action                          string          `json:"action"`
        InstallationID                  string          `json:"installation_id"`
        OwnerID                         string          `json:"owner_id"`
        PluginID                        string          `json:"plugin_id"`
        ExpectedPluginGeneration        uint64          `json:"expected_plugin_generation"`
        Requirements                    Requirements    `json:"requirements"`
        BundleDirectory                 string          `json:"bundle_directory,omitempty"`
        Files                           Files           `json:"files"`
        LicenseBinding                  *LicenseBinding `json:"license_binding,omitempty"`
        Consumers                       []Consumer      `json:"consumer_bindings,omitempty"`
        PrincipalID                     string          `json:"principal_id,omitempty"`
}
```

```
type Failure struct {
	Code                          string  `json:"code"`
	Stage                         string  `json:"stage"`
	Message                       string  `json:"message"`
	PreviousRegistrationPreserved bool    `json:"previous_registration_preserved"`
	RecoveryRequired              bool    `json:"recovery_required"`
}

func (e *Failure) Error() string

type Access struct {
	PrincipalID     string  `json:"principal_id"`
	Verified        bool    `json:"verified"`
}

type Result struct {
	PluginID               string                  `json:"plugin_id"`
	OwnerID                string                  `json:"owner_id"`
	Generation             uint64                  `json:"registration_generation"`
	RegistrationPersisted  bool                    `json:"registration_persisted"`
	ProcessState           string                  `json:"process_state"`
	ClientAccess           []Access                `json:"client_access"`
	ApplicationReadiness   map[string]string       `json:"application_readiness"`
}

type Operation struct {
	Contract        string          `json:"contract"`
	OperationID     string          `json:"operation_id"`
	RequestSHA256   string          `json:"request_sha256"`
	State           string          `json:"state"`
	Result          *Result         `json:"result,omitempty"`
	Error           *Failure        `json:"error"`
}

type Call struct {
	Permissions     *PermissionCheck               `json:"permissions,omitempty"`
	Target          *pluginipc.HostCompatibility   `json:"target,omitempty"`
	PluginID        string                         `json:"plugin_id,omitempty"`
	OwnerID         string                         `json:"owner_id,omitempty"`
	Method          string                         `json:"method"`
	OperationID     string                         `json:"operation_id,omitempty"`
	Request         *Request                       `json:"request,omitempty"`
}

type Reply struct {
	Permissions     *PermissionResult       `json:"permissions,omitempty"`
	Inventory       *PluginInventory        `json:"inventory,omitempty"`
	Registration    *Registration           `json:"registration,omitempty"`
	Installation    *Installation           `json:"installation,omitempty"`
	Operation       *Operation              `json:"operation,omitempty"`
	Error           *Failure                `json:"error,omitempty"`
}

type PermissionCheck struct {
	PrincipalID     string                      `json:"principal_id"`
	ClientAddress   string                      `json:"client_address,omitempty"`
	Query           pluginipc.PermissionQuery   `json:"query"`
}

type PermissionResult struct {
	PluginID                string                          `json:"plugin_id"`
	PrincipalID             string                          `json:"principal_id"`
	Decision                string                          `json:"decision"`
	Routing                 string                          `json:"routing"`
	Management              string                          `json:"management"`
	Network                 string                          `json:"network"`
	Domain                  pluginipc.PermissionDecision    `json:"domain"`
	AuthenticationVerified  bool                            `json:"authentication_verified"`
	Reason                  string                          `json:"reason,omitempty"`
}

type InventoryPlugin struct {
	OS                      string                          `json:"os,omitempty"`
	Arch                    string                          `json:"arch,omitempty"`
	PluginID                string                          `json:"plugin_id"`
	Source                  string                          `json:"source"`
```

```
        OwnerID                 string                          `json:"owner_id,omitempty"`
        Enabled                 bool                            `json:"enabled"`
        Version                 string                          `json:"version,omitempty"`
        Generation              uint64                          `json:"registration_generation"`
        ProcessState            string                          `json:"process_state"`
        Requirements            *pluginipc.Compatibility        `json:"requirements,omitempty"`
        RequirementsVerified    bool                            `json:"requirements_verified"`
        RuntimeContractVerified bool                            `json:"runtime_contract_verified"`
        TargetCompatible        bool                            `json:"target_compatible"`
        Blockers                []string                        `json:"blockers"`
}

type PluginInventory struct {
        InstallationID          string                          `json:"installation_id"`
        SnapshotSHA256          string                          `json:"snapshot_sha256"`
        Complete                bool                            `json:"complete"`
        UpgradeReady            bool                            `json:"upgrade_ready"`
        Target                  pluginipc.HostCompatibility     `json:"target"`
        Plugins                 []InventoryPlugin               `json:"plugins"`
        PendingOperations       []string                        `json:"pending_operations"`
        Blockers                []string                        `json:"blockers"`
}

type Registration struct {
        PluginID        string          `json:"plugin_id"`
        OwnerID         string          `json:"owner_id"`
        Generation      uint64          `json:"registration_generation"`
        ProcessState    string          `json:"process_state"`
        Consumers       []string        `json:"consumers"`
}

func DecodeRequest(data []byte) (Request, error)

func Raw(value any) json.RawMessage
```

pkg/providerjobs

queue.go

Источник: pkg/providerjobs/queue.go.

```
type Queue struct {
}

func Open(dir string, m Handler) (*Queue, error)

func OpenWithOptions(dir string, m Handler, options Options) (*Queue, error)

func (q *Queue) Maintain() error

func (q *Queue) Close() error

func (q *Queue) Invoke(ctx context.Context, method string, body json.RawMessage) pluginipc.Response
```

types.go

Источник: pkg/providerjobs/types.go.

```
type Handler interface {
        Invoke(context.Context, string, json.RawMessage) pluginipc.Response
        Authorized(context.Context, string, string, string) bool
}

type Options struct {
        MaxRecords              int
        MaxStateBytes           int
        MaxConcurrent           int
        DefaultTimeout          time.Duration
        MaxTimeout              time.Duration
        Retention               time.Duration
        MaintenanceInterval     time.Duration
```

```
}

type Progress struct {
        Stage           string  `json:"stage"`
        Completed       int     `json:"completed"`
        Total           int     `json:"total"`
}
```

pkg/providerpolicy

policy.go

Источник: pkg/providerpolicy/policy.go.

```
type Policy struct {
        PrincipalID     string          `json:"principal_id"`
        Product         string          `json:"product"`
        Grants          []api.Grant     `json:"grants"`
}

func Authorized(ctx context.Context, policies []Policy, module, operation, scope, resource string) bool
```

pkg/pluginipc/plugintest

executable.go

Источник: pkg/pluginipc/plugintest/executable.go.

```
type ExecutableOptions struct {
        ID              string
        Settings        any
        Package         string
        Executable      string
        Staged          bool
}

type Executable struct {
        Client *pluginipc.Client
}

func StartExecutable(t *testing.T, options ExecutableOptions) *Executable

func (p *Executable) CheckIdentityAndAuth(t *testing.T)

func (p *Executable) Stop(t *testing.T, leaseLoss bool)
```

suite.go

Источник: pkg/pluginipc/plugintest/suite.go.

```
type Options struct {
        ID              string
        Method          string
        Body            any
        Execution       pluginipc.Execution
        Check           func(*testing.T, pluginipc.Response)
}

func Run(t *testing.T, factory func(*testing.T) pluginipc.Handler, options Options)
```

pkg/keychain

keychain_darwin.go

Источник: pkg/keychain/keychain_darwin.go.

```
func Get(service, account string) ([]byte, error)

func Set(service, account string, value []byte) error

func Delete(service, account string) error
```

keychain_other.go

Источник: pkg/keychain/keychain_other.go.

```
func Get(service, account string) ([]byte, error)

func Set(service, account string, value []byte) error

func Delete(service, account string) error
```

Всего экспортируемых имён и функций: 360.