Центр
Управления
Гетерогенными
Инфраструктурами

Система централизованного анализа и управления гетерогенными инфраструктурами — MiniCA

Инструментальные средства разработки

RU.CLRW.00947-01 95 01

ООО «Клируэй Текнолоджис»

Москва 2025

Оглавление

1. Перечень терминов и сокращений 3
2. Введение....5
   - 2.1. Идентификационные данные....5
   - 2.2. Описание....6
3. Идентификация подразделений организаций, участвующих в разработке 8
4. Языки программирования 8
   - 4.1. Список используемых языков программирования....8
5. Средства производства 8
6. Описание средств разработки 10
   - 6.1. Среда разработки (IDE)....10
   - 6.1.1. Средства разработки 10
   - 6.1.2. Компиляторы исходного кода....10
   - 6.2. Среда хранения исходных текстов 10
   - 6.2.1. Система хранения исходных текстов проекта....11
   - 6.3. Среда сборки 11
   - 6.3.1. Средства сборки продуктов....12
   - 6.3.2. Схемы и описание сборки продукта....12
   - 6.3.3. Скрипт сборки компонентов 16
   - 6.3.4. Скрипт автоматизированной сборки....16
   - 6.3.5. Характеристики стенда для сборки....33
   - 6.4. Среда хранения артефактов 34
   - 6.4.1. Система хранения артефактов....34
   - 6.5. Среда поиска уязвимостей 34
   - 6.5.1. Сканеры уязвимостей....34
   - 6.6. Среда тестирования проекта....35
   - 6.6.1. Средства и методы тестирования 35
   - 6.7. Среда развертывания 36
   - 6.7.1. Система управления контейнерами....36
   - 6.8. Среда хранения документов 37
   - 6.9. Средства разработки документации....37
   - 6.10. Среда планирования работ и управленческого учета 38
   - 6.10.1. Система планирования работ и управленческого учета....38

1. Перечень терминов и сокращений

Таблица 1 — Перечень терминов и сокращений.

Сокращение	Расшифровка	Описание
Ansible		Открытый инструмент для автоматизации управления конфигурациями, разработанный на языке Python. Он предназначен для настройки, развертывания и сопровождения серверов и приложений.
API	Application Programming Interface	Прикладной программный интерфейс компонента ИС
BPMN	Business Process Model and Notation	Система условных обозначений (нотация) и их описания в XML для моделирования бизнес-процессов.
HSM	Hardware Security Module	Специализированное устройство для генерации, хранения и управления криптографическими ключами.
IDE	Integrated Development Environment	Интегрированная среда разработки, включающая набор инструментов для написания, редактирования, компиляции, отладки и тестирования программного кода.
JWT	JSON Web Token	Открытый стандарт (RFC 7519), определяющий компактный и самодостаточный способ передачи информации между сторонами в виде JSON-объекта, подписанного цифровой подписью или зашифрованного.
KRA	Key Recovery Agent	Пользователь, имеющий специальный сертификат и права для резервного копирования и восстановления криптографических ключей и цифровых сертификатов.
PKI (ИОК)	Public Key Infrastructure	Инфраструктура открытых ключей (ИОК) — набор средств, распределенных служб и компонентов, используемых для поддержки крипто-задач (шифрования, аутентификации, подписи) на основе закрытого и открытого ключей.
RSA	Rivest - Shamir - Adleman	Реализация криптосистемы на основе закрытого и открытого ключей
SCEP	Simple Certificate Enrollment Protocol	Протокол инфраструктуры PKI, который используется многими производителями сетевого оборудования и программного обеспечения. SCEP на сегодня

Сокращение	Расшифровка	Описание
		является наиболее популярным, широкодоступным и проверенным протоколом автоматизированной регистрации сертификатов X.509.
SSL	Secure Socket Layers	Криптографический протокол, обеспечивающий организацию безопасной сетевой связи между клиентом и сервером и упрощенной проверкой подлинности сторон связи с применением сертификатов x509.
TLS / mTLS	Transport Level Security / mutual Transport Level Security	Развитие протокола SSL, криптографический протокол на основе сертификатов x509, обеспечивающий организацию безопасной сетевой связи и взаимную аутентификацию клиента и сервера.
UML	Unified Modeling Language	Язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.
X.509v3		Версия международного стандарта, определяющая структуру цифровых сертификатов, содержащих информацию о субъекте (пользователе, устройстве или организации), открытый ключ субъекта и дополнительные расширения, используемые для настройки поведения сертификата и улучшения функциональности. Является основой современных методов аутентификации и защиты данных в интернет-коммуникациях, включая TLS/SSL и электронную почту.
WSTEP		Протокол Microsoft WS-Trust X.509v3 Token Enrollment Extensions
ИС		Информационная система
ОС		Операционная система
Пайплайн	Pipeline	Последовательность взаимосвязанных этапов, через которые проходят задачи, проекты или данные от начала до завершения.
Плейбук	Playbook	Формализованный документ или набор инструкций, используемых для автоматизации действий на удалённых машинах. Применяется в инструментах вроде Ansible, содержит перечень задач, необходимых для выполнения определённых операций, и представлен в формате YAML.

Сокращение	Расшифровка	Описание
УЦ	Удостоверяющий Центр	Программный или программно-аппаратный комплекс, обеспечивающий управление жизненным циклом сертификатов x509
ЦС (CA)	Центр Сертификации (Certification Authority)	Сервер, предназначенный для выпуска и отзыва сертификатов, а также публикации CRL (COC). Часть Удостоверяющего Центра.

2. Введение

Настоящий документ относится к эксплуатационной документации ПО «Система централизованного анализа и управления гетерогенными инфраструктурами — MiniCA» (далее - MiniCA). Разработчиком ПО MiniCA является ООО «Клируэй Текнолоджис».

MiniCA — это центр сертификации, который выполняет выпуск сертификатов X.509v3 на основе CSR-запросов в формате PKCS#10, используя готовые шаблоны. Система позволяет создавать и настраивать эти шаблоны, а также отзыв сертификатов с указанием причины. MiniCA формирует списки отозванных сертификатов (CRL и DeltaCRL), предоставляет статус сертификатов через протокол OCSP и обеспечивает доступ к ним по серийному номеру или SHA1-отпечатку. Все запросы, сертификаты, журналы событий и CRL/DeltaCRL сохраняются в базе данных. Управление осуществляется через web-интерфейс и API, а также поддерживается работа с протоколом SCEP для выпуска сертификатов.

ПО MiniCA является развитием одного из модулей ПО «Система централизованного анализа и управления гетерогенными инфраструктурами (ЦУГИ)», зарегистрированной в Реестре российского ПО (Реестровая запись №13033 от 21.03.2022), обладает расширенным составом функций и позволяет применяться в отдельно устанавливаемом исполнении.

2.1. Идентификационные данные

Идентификационные данные приведены в Таблице 2.

Таблица 2 — Идентификационные данные.

Идентификационные данные ОО	
Название документа	Инструментальные средства разработки
Децимальный номер документа	RU.CLRW.00947-01 95 01

Идентификационные данные ОО	
Версия документа	1.0
Автор документа, предприятие-изготовитель	ООО «Клируэй Текнолоджис»

2.2. Описание

Основной функцией MiniCA является управление жизненным циклом сертификатов:

1) Выпуск сертификатов:
- Выпуск сертификатов X.509v3 на основе CSR запросов в формате PKCS#10;
- Поддержка шаблонов сертификатов для наполнения и контроля атрибутов сертификатов;
- Поддержка выпуска сертификатов по различным протоколам – SCEP, WSTEP и т.п.

2) Предоставление сертификата по запросу:
- Возможность получения выпущенных сертификатов по запросу;
- Возможность получения различных выборок и отчетов о выпущенных сертификатах.

3) Отзыв сертификата:
- Отзыв сертификатов с указанием причины;
- Формирование Списка Отозванных Сертификатов (COC, CRL);
- Формирование разностных списков отозванных сертификатов DeltaCRL;
- Предоставление информации о статусе сертификата по протоколу OCSP.

4) Вспомогательные функции:
- Сохранение информации о всех запросах, сертификатах, CRL/DeltaCRL в базе данных;
- Текстовый журнал событий;
- Web-интерфейс с поддержкой ролевой модели;
- API-интерфейс с обеспечением авторизации и разделения ролей;
- Поддержка функций самодиагностики;
- Поддержка хранения ключа ЦС на аппаратных носителях HSM.

5) Выпуск сертификатов:
- Выпуск сертификатов X.509v3 на основе CSR запросов в формате PKCS#10;
- Поддержка шаблонов сертификатов для наполнения и контроля атрибутов сертификатов.

6) Отзыв сертификата:
- Отзыв сертификатов с указанием причины;

- Возвращение в действие сертификата (Unrevoke);
- Формирование Списка Отозванных Сертификатов (COC, CRL);
- Формирование разностных списков отозванных сертификатов DeltaCRL;
- Предоставление информации о статусе сертификата по протоколу OCSP.

7) Предоставление информации о сертификатах по запросу:

- Возможность получения выпущенных сертификатов по запросу;
- Возможность массовой выгрузки сертификатов;
- Возможность получения различных выборок и отчетов о выпущенных сертификатах.

8) Дополнительные функции:

- Поддержка протоколов запроса сертификатов:
  - Поддержка выпуска сертификатов по протоколу SCEP;
  - Поддержка выпуска сертификатов по протоколу MS-WSTEP;
  - Поддержка выпуска сертификатов по протоколам ACMEv2, MS-RPC, CMP, EST.
- Интерфейс пользователя
  - Интерфейс командной строки;
  - Web-интерфейс;
  - API-интерфейс.
- Журналирование:
  - Журнал событий MiniCA в базе данных;
  - Текстовый журнал событий.
- Мониторинг:
  - Поддержка функций самодиагностики;
  - Оповещения о событиях жизненного цикла сертификатов.
- Безопасность:
  - Разграничение полномочий пользователей к функциям UI (ролевая модель);
  - Разграничение полномочий пользователей к функциям MiniCA;
  - Защищенное хранения закрытых ключей в БД ЦС и управление ими с помощью Агента Восстановления Ключей (KRA);
  - Поддержка хранения ключа ЦС на аппаратных носителях HSM.
- Взаимодействие с другими системами:
  - Публикация CRL на внешние серверы;
  - Публикация сертификатов и CRL в объекты LDAP Microsoft Active Directory;
  - Публикация сертификатов и CRL в объекты LDAP Samba;
  - Интеграция с системой KeyBox.
- Функции обслуживания:
  - Резервное копирование;
  - Архивирование истекших сертификатов.

3. Идентификация подразделений организаций, участвующих в разработке

Подразделения разработчика, участвующие в разработке:

- Отдел разработки, состоящий из руководителей проектов разработки, архитекторов, системных аналитиков, разработчиков и тестировщиков;
- Отдел технической поддержки.

4. Языки программирования

4.1. Список используемых языков программирования

MiniCA написана с использованием следующих языков программирования:

- язык программирования Go, 1.21;
- язык программирования C#, .Net 8.0.x.

5. Средства производства

Для разработки и производства MiniCA были использованы средства, представленные в Таблице 3.

Таблица 3 — Сводная информация по инструментальным средствам разработки и производства.

Используемая система	Используемый программный продукт	Используемая версия
Средства разработки	VS Code	Зависит от разработчика
Система хранения исходных текстов проекта	GitLab CE	17.11.2
	GitLab Shell	14.41.0
	GitLab Workhorse	17.11.2
	GitLab API	4
	GitLab KAS	17.11.2
	Sonatype Nexus Repository Community	Community 3.77.2-02

Используемая система	Используемый программный продукт	Используемая версия
Компиляторы исходного кода/среда выполнения	.Net	.Net Runtime 8.0, ASP.NET core runtime 8
	Go	1.23.8
Средства сборки продуктов	dpkg	1.19.7
	CMake	3.25
	Visual Studio	Зависит от разработчика
Сканеры уязвимостей	Dependency Check	12.1.1
	SonarQube Community Build	25.4.0.105899
	Svace	4.0.250415
Средства тестирования	go test	Не является самостоятельной библиотекой/файлом, версия равна версии Go
	golangci-lint	1.64.8
Система хранения документов проекта	Confluence	9.2.0.
Средства разработки документации	Atlassian Confluence	9.2.0
	Microsoft Word	2019
	Draw.IO	25.0.2
	PlantUML	1.2025.4
	Figma	125.5.6
	Embed Figma resources within Confluence pages	3.3.0
Система планирования работ и управленческого учета	Jira Software Data Center	9.17

6. Описание средств разработки

6.1. Среда разработки (IDE)

Среда разработки IDE (Integrated Development Environment) — комплекс программных инструментов, предназначенный для создания, редактирования, компиляции, отладки и тестирования программного кода. IDE объединяет редактор кода, компилятор, отладчик и другие вспомогательные инструменты в единую рабочую среду.

6.1.1. Средства разработки

Применяются следующие средства разработки:

- VS Code (Visual Studio Code) —универсальный редактор кода с открытым исходным кодом;
- Visual Studio —интегрированная среда разработки (IDE). Включает инструменты для проектирования, написания, отладки и тестирования приложений. Позволяет создавать программы на различных языках программирования (C#, VB.NET, F#, C++ и другие), а также охватывает все аспекты разработки — от прототипирования до выпуска финального релиза.

6.1.2. Компиляторы исходного кода

Компиляция исходного кода осуществляется следующими инструментами:

- .Net — семейство компиляторов, разработанных специально для языков .Net (C#, Visual Basic, F#). Обладают современными возможностями рефакторинга, автодополнения и интерактивного анализа кода.
- Go (GCCgo, Gofrontend) — официальные компиляторы языка Go, представленные в составе стандартной поставки Go SDK.

6.2. Среда хранения исходных текстов

Среда хранения исходных текстов — инфраструктура и набор инструментов для надежного и организованного хранения файлов исходного кода программного обеспечения, включая систему контроля версий, репозитории и сервера для размещения данных. Она обеспечивает возможность совместного доступа и внесения изменений несколькими разработчиками одновременно, сохраняя историю всех модификаций.

6.2.1. Система хранения исходных текстов проекта

Для обеспечения надежности, безопасности и удобства управления кодом, используется собственный сервер GitLab — платформа для разработки программного обеспечения.

Основные компоненты GitLab:

- GitLab CE (Community Edition) — бесплатная версия платформы, включающая базовые функции для управления проектами, репозиториями, issue-tracker, CI/CD и wiki;
- GitLab Shell — компонент, ответственный за обработку SSH-запросов и взаимодействие с Git-репозиториями. Выполняет проверку прав доступа и управляет операциями Git, выступая посредником между клиентом и GitLab;
- GitLab Workhorse — внутренний прокси-сервер, принимающий внешние запросы и разгружающий основной веб-сервер GitLab. Обрабатывает тяжелые задачи, такие как скачивание файлов, генерация миниатюр и запросы Git, улучшая производительность системы;
- GitLab API — программный интерфейс для удалённого управления всеми аспектами GitLab через стандартные HTTP-запросы. Позволяет автоматизировать операции с проектами, изменениями, users, CI/CD пайплайнами и другими элементами.

6.3. Среда сборки

Среда сборки представляет собой закрытый контур, без возможности получения доступа к нему пользователями. Для получения доступа в среду сборки используются Специальные Технические Учетные Записи.

Сборка серверного компонента продукта происходит с использованием компилятора Go, версии 1.22. Сборка компонентов ControlPlane (web-консоль управления) происходит при помощи SDK .Net 8.x, а также NPM, в фреймворке Angular.

Релиз выпускается в виде Debian пакетов (.deb), SFX архивов (.sfx) и бинарных файлов, упакованных в архивы tar (.tar.gz). В релиз, помимо скомпилированных артефактов, прикладываются установочные скрипты на базе Python/Bash.

Сборка Linux версий производится на операционной системе Astra Linux Special Edition РУСБ.10015-01 (x86-64).

Исходные коды программного обеспечения хранятся на собственном сервере GitLab, перед каждой сборкой производится получение полной копии исходных кодов.

Перед попаданием в среду сборки код проходит следующие этапы сканирования:

- SAST, на базе SonarQube;

- Сканирование зависимостей на базе, OWASP Dependency Check при помощи открытой базы NVD.

6.3.1. Средства сборки продуктов

Используются следующие средства сборки:

- dpkg — базовый пакетный менеджер в системах семейства Debian. Устанавливает, удаляет и контролирует пакеты формата DEB, обеспечивая совместимость и управление зависимостями;
- CMake — кроссплатформенная система сборки, создающая сценарии для компиляции и сборки проектов. Упрощает переносимость проектов на разные платформы и окружения;
- Autotools — набор инструментов для автоматизации сборки и установки программ в Unix-средах. Автоконфигурация и автообновление makefiles обеспечивают бесшовную работу на различных версиях Unix-подобных систем.

6.3.2. Схемы и описание сборки продукта

6.3.2.1. Автоматизированные действия по сборке

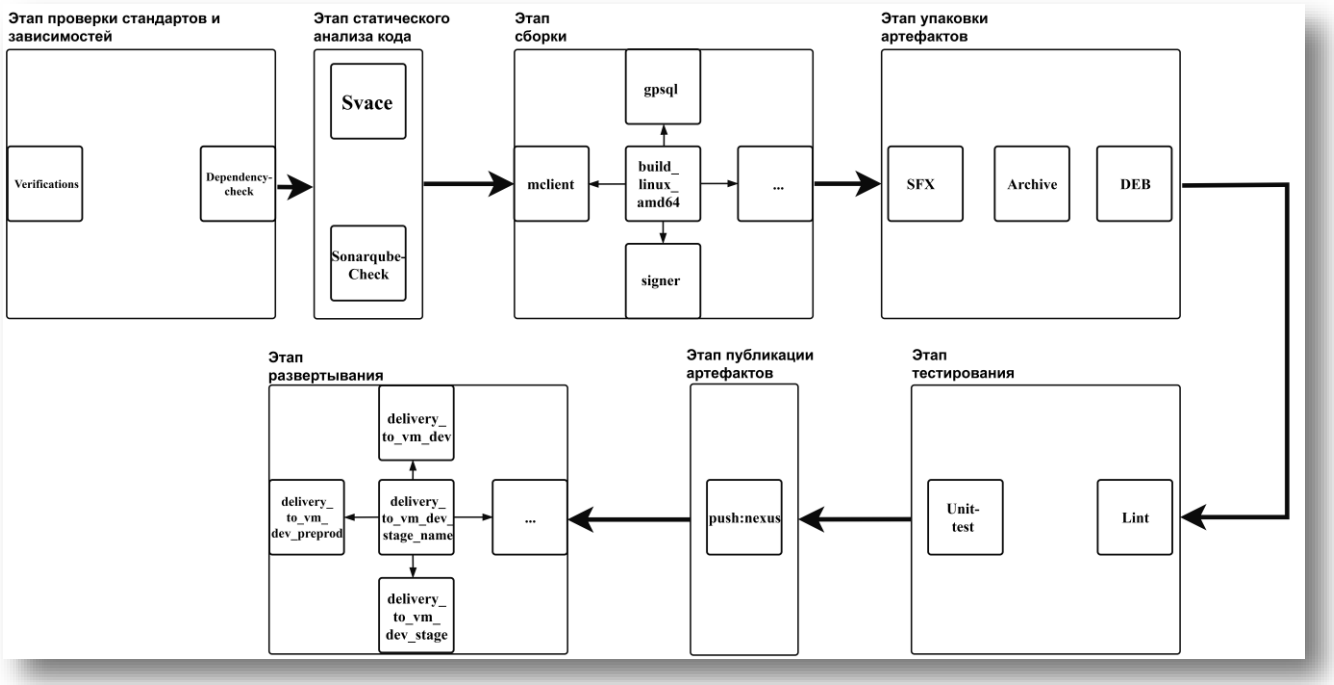

Рисунок 1 —Схема автоматизированной сборки

Описание стадий и блоков:

1) Этап проверки стандартов и зависимостей.

- Блок «Verifications». Внутренний CLI-инструмент `verifier`, запускаемый в выделенном Docker-контейнере, выполняет аудит проекта:
  1) Автоматически определяет пространство имён (`namespace`) и конфигурацию проекта;
  2) Проверяет структуру проекта, наличие README и других обязательных элементов.
- Блок «Dependency-check». Используется утилита OWASP Dependency-Check с подключением к базе уязвимостей NVD:
  1) Выполняется анализ зависимостей на наличие известных уязвимостей в проектах на Go, JavaScript и .NET;
  2) Используется API-ключ NVD для получения актуальных данных;
  3) Результат сохраняется в виде HTML-отчета, прикладываемого к пайплайну в качестве артефакта;
  4) Отчет хранится 1 день и доступен через GitLab-интерфейс.

2) Этап статического анализа кода

- Блок «Sonarqube-check». Проверка кода на уязвимости и потенциальные дефекты с помощью платформы SonarQube:
  1) Используется сканер `sonar-scanner` и внутренний сервер SonarQube;
  2) Сканирование запускается с параметрами, передаваемыми через переменные окружения;
  3) Используется кэш для ускорения повторных запусков;
  4) Результаты анализа доступны только через веб-интерфейс SonarQube, с авторизацией через LDAP.

3) Этап сборки

- Блок «build_linux_amd64». Компиляция исполняемых файлов для операционной системы Linux и архитектуры amd64:
  1) Используется Docker-образ `cwirunner` с предустановленным Go SDK;
  2) Компиляция выполняется с использованием `go build` и `ldflags` для внедрения метаданных: версии, SHA коммита и времени сборки;
  3) Сборка производится параллельно для всех микросервисов (`gpsql`, `mclient`, `signer` и другие);
  4) Артефакты сохраняются в GitLab и хранятся 1 день.

4) Этап упаковки артефактов

- Блок «Archive»:
  1) Бинарный файл упаковывается в архив `.tar.gz` с помощью стандартной утилиты `tar`;
  2) Результат сохраняется как артефакт GitLab.
- Блок «DEB»:

1) Используется кастомный скрипт dpkg-deb.sh для упаковки приложения в формат .deb;
2) Скрипт выполняется в директории make-deb/, с предварительной установкой прав доступа;
3) Артефакт хранится в течение 1 дня.

- Блок «SFX»:
  1) Бинарный файл упаковывается в самораспаковывающийся .run-архив с помощью утилиты makeself;
  2) Перед упаковкой устанавливаются зависимости (xz-utils, makeself);
  3) Каждый архив создается в рамках отдельного параллельного задания.

5) Этап тестирования

- Блок «unit-test»:
  1) Запуск модульных тестов проекта с помощью команды go test ./...;
  2) Используется тот же образ, что и для сборки;
  3) Переменная .netrc расшифровывается для доступа к приватным зависимостям;
  4) Ошибки не блокируют выполнение пайплайна (опция allow_failure: true).
- Блок «lint»:
  1) Проверка качества и стиля кода с помощью утилиты golangci-lint;
  2) Конфигурация линтера формируется из переменной LINT_CONFIG;
  3) Выполняется go mod tidy перед запуском линта;
  4) Результаты не влияют на статус пайплайна (allow_failure: true);
  5) Файл конфигурации сохраняется как артефакт.

6) Этап публикации артефактов

- Блок «push:nexus». Загрузка собранных артефактов в хранилище Nexus:
  1) Используется инструмент curl с передачей учетных данных через переменные окружения;
  2) Путь публикации строится по шаблону: https://repos.clearwayintegration.com/repository/release/<путь_проекта>/<версия>/<артефакт>;
  3) Каждый тип артефакта (.tar.gz, .deb, .run, .tgz) загружается своим шагом, с возможностью ручного запуска.

7) Этап развертывания

- Блоки «delivery_to_vm_dev»; «delivery_to_vm_stage»; «delivery_to_vm_preprod». Развёртывание артефактов на виртуальные машины с помощью Ansible:

1) Используется плейбук delivery-only.yml;
2) Инвентарные файлы подгружаются из отдельного репозитория по пути environments-configurations/<система>/<проект>/inventory.ini;

3) Авторизация выполняется через переменные ANSIBLE_USER и ANSIBLE_PASSWORD;
4) Развёртывание может быть триггером ручного запуска в зависимости от типа тега (RC, Release);
5) Происходит параллельное развертывание для каждого сервиса.

6.3.2.2. Ручная сборка

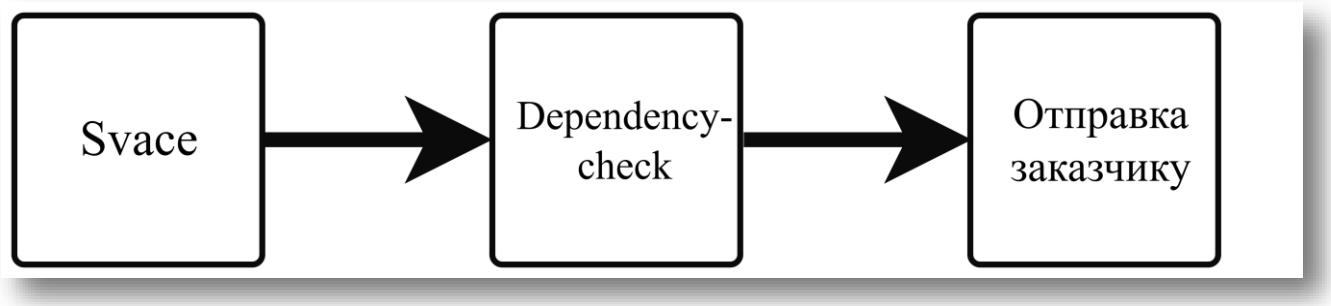

Рисунок 2 — Схема ручной сборки

Описание блоков:

1) Блок «Svace». Проводится статический анализ (SAST) кода с использованием инструмента Svace. На данный момент не встроен в пайплайн.
2) Блок «Dependency-check». Дистрибутив еще раз проверяется на безопасность, когда в нем собраны все сервисы, идущие к заказчику. Проверка производится полуавтоматически.
3) Блок «Отправка заказчику». Производится отправка дистрибутива заказчику.

6.3.3. Скрипт сборки компонентов

```
- echo ${NETRC_ENV} | base64 -d > ~/.netrc
- echo "Building ${SERVICE_NAME} version ${VERSION}"
- |
  if [ -z "${LDFLAGS+x}" ]; then
    export MODULE_PATH=$(grep '^module ' go.mod | awk '{print $2}')
    export LDFLAGS="-X ${MODULE_PATH}/disp.GitCommit=$CI_COMMIT_SHORT_SHA -X ${MODULE_PATH}/disp.Version=$VERSION"
  fi
  set -x
  go build -o ${BIN_DIR}/${OUT_BINARY_NAME} -trimpath -ldflags="${LDFLAGS}" ./cmd/${OUT_BINARY_NAME}
```

6.3.4. Скрипт автоматизированной сборки

```
---
workflow:
 rules:
 - if: $CI_PIPELINE_SOURCE == "merge_request_event"
   when: never
 - if: "$CI_COMMIT_BRANCH =~ /^feature\\/.+$/ || $CI_COMMIT_BRANCH =~ /^bugfix\\/.+$/
     || $CI_COMMIT_BRANCH =~ /^hotfix\\/.+$/"
   variables:
     VERSION: "$DEFAULT_VERSION"
     DOTNET_CONFIGURATION: Debug
     MSBUILD_EXTRA_ARGS: "/p:DebugType=full /p:DebugSymbols=true /p:IncludeSymbols=true
       /p:EmbedAllSources=true"
 - if: $CI_COMMIT_BRANCH == $AUTODEPLOY_DEV_BRANCH || $CI_COMMIT_BRANCH == "develop"
     || $CI_COMMIT_BRANCH == "stage" || $CI_COMMIT_BRANCH == "master" || $CI_COMMIT_BRANCH
     == "main"  || $CI_COMMIT_BRANCH == "devops" || $CI_COMMIT_BRANCH == "demo"
   variables:
     VERSION: "$DEFAULT_VERSION"
     DOTNET_CONFIGURATION: Debug
```

```
    MSBUILD_EXTRA_ARGS: "/p:DebugType=full /p:DebugSymbols=true /p:IncludeSymbols=true
      /p:EmbedAllSources=true"
  - if: "$CI_COMMIT_TAG && $CI_COMMIT_TAG =~ $RC_TAG_REGEX"
    variables:
      VERSION: "$CI_COMMIT_TAG"
      DOTNET_CONFIGURATION: Release
      MSBUILD_EXTRA_ARGS: ''
  - if: "$CI_COMMIT_TAG && $CI_COMMIT_TAG =~ $TAG_REGEX"
    variables:
      VERSION: "$CI_COMMIT_TAG"
      DOTNET_CONFIGURATION: Release
      MSBUILD_EXTRA_ARGS: ''
  - when: never
verification:
  stage: verifications
  image: "${VERIFIER_IMAGE_NAME}:${VERIFIER_IMAGE_TAG}"
  allow_failure:
    exit_codes:
    - 2
  script:
  - if [ -z "${NAMESPACE_NAME}" ]; then export NAMESPACE_NAME=$(echo $CI_PROJECT_NAMESPACE);
    fi
  - echo $NAMESPACE_NAME
  - echo /app/verifier --path $CI_PROJECT_PATH --title "$CI_PROJECT_TITLE" --name
    $CI_PROJECT_NAME --log-level debug --namespace ${NAMESPACE_NAME} --type ${VERIFIER_CHECKS_TYPES}
    .
  - /app/verifier --path $CI_PROJECT_PATH --title "$CI_PROJECT_TITLE" --name $CI_PROJECT_NAME
    --log-level debug --namespace ${NAMESPACE_NAME} --type ${VERIFIER_CHECKS_TYPES}
    .
dependency-check:
  stage: verifications
  image:
    name: "${DEPENDENCY_CHECK_IMAGE_NAME}:${DEPENDENCY_CHECK_IMAGE_TAG}"
    entrypoint:
    - ''
```

```
54:   script:
55:   - git config --global
      url."https://${GITLAB_USER}:${GITLAB_TOKEN}@clearway.gitlab.yandexcloud.net/".instead
      Of
56:     "https://clearway.gitlab.yandexcloud.net/"
57:   - echo "/usr/share/dependency-check/bin/dependency-check.sh --project
      "${CI_PROJECT_NAME}"
58:     --scan "." --format "HTML" --nvdApiKey "${NVD_API_KEY}" --enableExperimental --data
59:     dependency-check-data --out "dependency-check-report""
60:   - /usr/share/dependency-check/bin/dependency-check.sh --project
      "${CI_PROJECT_NAME}"
61:     --scan "." --format "HTML" --nvdApiKey "${NVD_API_KEY}" --enableExperimental --data
62:     dependency-check-data --out "dependency-check-report"
63:   artifacts:
64:     paths:
65:     - dependency-check-report/
66:     expire_in: 1 days
67:   allow_failure: true
68:   cache:
69:     key: dependency-check
70:     paths:
71:     - dependency-check-data
72:   rules:
73:   - if: "$CI_COMMIT_TAG"
74:     when: manual
75: create-release:
76:   stage: release
77:   image: "${RELEASE_IMAGE_NAME}:${RELEASE_IMAGE_TAG}"
78:   allow_failure:
79:     exit_codes:
80:     - 2
81:   script:
82:   - |
83:     # Проверяем существование релиза через API с JOB-TOKEN
84:     if curl -s --header "JOB-TOKEN: $CI_JOB_TOKEN" \
85:       "${CI_API_V4_URL}/projects/$CI_PROJECT_ID/releases/$CI_COMMIT_TAG" | grep -q
      '"tag_name"'; then
86:       echo "Release already exists - skipping"
87:       exit 2  # Желтый статус
```

```
    fi
  - mkdir -p .chglog
  - "cat > .chglog/config.yml <<'EOF'\nstyle: gitlab\ntemplate: CHANGELOG.tpl.md\ninfo:\n
    \ title: \"Changelog\"\noptions:\n  sort: \"date\"\n  commits:\n    filters:\n
    \    Type: [\"feat\", \"fix\", \"perf\"]  # Только эти типы\n  commit_groups:\n
    \   title_maps:\n      feat: \"\U0001F7E2 Features\"\n      fix: \"\U0001F534
    Bug Fixes\"\n      perf: \"\U0001F680 Performance\"\n  header:\n    pattern:
  \"^\\\\s*(\\\\w+):\\\\s*(.*)$\"
    \ # Формат: \"type: subject\"\n    pattern_maps:\n      - Type  # 1-я группа -
    тип\n      - Subject  # 2-я группа - заголовок\nEOF\n"
  - |
    cat > .chglog/CHANGELOG.tpl.md <<'EOF'
    {{- if .Versions }}
    {{- $current := index .Versions 0 -}}
    ## {{ $current.Tag.Name }}
    ### Сравнение с: {{ if $current.Tag.Previous }}{{ $current.Tag.Previous.Name }}{{ else
  }}началом репозитория{{ end }}
    {{$current.Tag.Date.Format "2 January 2006"}}
    {{ if $current.CommitGroups -}}
    {{ range $current.CommitGroups -}}
    ### {{ .Title }}
    {{ range .Commits -}}
    - {{ .Subject }} ({{ .Hash.Short }})
    {{ end }}
    {{ end -}}
    {{ end -}}
    {{ end -}}
    EOF
    git fetch --tags -f --unshallow
    if [[ "$CI_COMMIT_TAG" =~ $TAG_REGEX ]]; then
      git-chglog --tag-filter-pattern=$TAG_REGEX $CI_COMMIT_TAG > CHANGELOG.md
    else
      # Для RC-тега, сравнит с предыдущим релизным тегом если он есть
      git-chglog $CI_COMMIT_TAG > CHANGELOG.md
    fi
  release:
    name: Release $CI_COMMIT_TAG
    description: CHANGELOG.md
    tag_name: "$CI_COMMIT_TAG"
```

```
    ref: "$CI_COMMIT_SHA"
    assets:
      links:
      - name: Binary $SERVICE_NAME
        url: https://repos.clearwayintegration.com/repository/release/${CI_PROJECT_PATH}/${VERSION}/${OUT_BINARY_NAME}-${VERSION}-linux-amd64.tar.gz
  rules:
  - if: "$CI_COMMIT_TAG"
".build_linux_amd64":
  stage: build
  image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
  variables:
    CGO_ENABLED: '0'
    GOOS: linux
    GOARCH: amd64
  script:
  - echo ${NETRC_ENV} | base64 -d > ~/.netrc
  - echo "Building ${SERVICE_NAME} version ${VERSION}"
  - |
    if [ -z "${LDFLAGS+x}" ]; then
      export MODULE_PATH=$(grep '^module ' go.mod | awk '{print $2}')
      export LDFLAGS="-X ${MODULE_PATH}/disp.GitCommit=$CI_COMMIT_SHORT_SHA -X ${MODULE_PATH}/disp.Version=$VERSION"
    fi
    set -x
    go build -o ${BIN_DIR}/${OUT_BINARY_NAME} -trimpath -ldflags="${LDFLAGS}" ./cmd/${OUT_BINARY_NAME}
  artifacts:
    paths:
    - "${BIN_DIR}/${OUT_BINARY_NAME}"
    expire_in: 1 day
  rules:
  - if: $MONOREPO == null || $MONOREPO == ""
build_linux_amd64:
  stage: build
  image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
  variables:
    CGO_ENABLED: '0'
```

```
    GOOS: linux
    GOARCH: amd64
    LDFLAGS: "-X=main.Version=${VERSION} -X=main.GitHash=${CI_COMMIT_SHORT_SHA} -X=main.BuildTime=${CI_PIPELINE_CREATED_AT}"
  script:
  - echo ${NETRC_ENV} | base64 -d > ~/.netrc
  - echo "Building ${SERVICE_NAME} version ${VERSION}"
  - |
    if [ -z "${LDFLAGS+x}" ]; then
      export MODULE_PATH=$(grep '^module ' go.mod | awk '{print $2}')
      export LDFLAGS="-X ${MODULE_PATH}/disp.GitCommit=$CI_COMMIT_SHORT_SHA -X ${MODULE_PATH}/disp.Version=$VERSION"
    fi
    set -x
    go build -o ${BIN_DIR}/${OUT_BINARY_NAME} -trimpath -ldflags="${LDFLAGS}" ./cmd/${OUT_BINARY_NAME}
  artifacts:
    paths:
    - "${BIN_DIR}/${OUT_BINARY_NAME}"
    expire_in: 1 day
  rules:
  - if: $BUILD_TARGETS != "" && $BUILD_TARGETS =~ /build_linux_amd64/
  parallel:
    matrix:
    - SERVICE_NAME: gpsql
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: maskpass
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mclient
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: minica
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mjwt
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mocsp
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: signer
      OUT_BINARY_NAME: "$SERVICE_NAME"
  extends:
```

```
196:   - ".build_linux_amd64"
197:   - ".service_list"
198:  unit-test:
199:   stage: test
200:   image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
201:   script:
202:   - echo ${NETRC_ENV} | base64 -d > ~/.netrc
203:   - go test ./...
204:   allow_failure: true
205:   rules:
206:   - if: "$CI_COMMIT_BRANCH"
207:  lint:
208:   stage: test
209:   image: "${GOLANGCI_LINT_IMAGE_NAME}:${GOLANGCI_LINT_IMAGE_TAG}"
210:   script:
211:   - echo ${NETRC_ENV} | base64 -d > ~/.netrc
212:   - echo "$LINT_CONFIG" > .golangci-lint.yaml
213:   - go mod tidy
214:   - "echo \"\U0001F50D Запускаем golangci-lint\""
215:   - golangci-lint run --timeout 5m --config .golangci-lint.yaml ./...
216:   artifacts:
217:    name: ".golangci-lint.yaml"
218:    when: always
219:    expire_in: 1 days
220:    paths:
221:    - ".golangci-lint.yaml"
222:   allow_failure: true
223:   rules:
224:   - if: "$CI_COMMIT_BRANCH"
225:  ".package:ARCHIVE":
226:   stage: package
227:   image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
228:   dependencies:
229:   - build_linux_amd64
230:   script:
231:   - tar -czvf "${BIN_DIR}/${OUT_BINARY_NAME}-${VERSION}-linux-amd64.tar.gz" -C
      ${BIN_DIR}/
232:    ${OUT_BINARY_NAME}
233:   - echo "✅ Артефакт создан - ${BIN_DIR}/${OUT_BINARY_NAME}-${VERSION}-linux-
```

```
    amd64.tar.gz"
  artifacts:
    paths:
    - "${BIN_DIR}/${OUT_BINARY_NAME}-*.tar.gz"
    expire_in: 1 days
  rules:
  - if: $MONOREPO == null || $MONOREPO == ""
package:ARCHIVE:
  stage: package
  image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
  dependencies:
  - build_linux_amd64
  script:
  - tar -czvf "${BIN_DIR}/${OUT_BINARY_NAME}-${VERSION}-linux-amd64.tar.gz" -C
${BIN_DIR}/
    ${OUT_BINARY_NAME}
  - echo "✅ Артефакт создан - ${BIN_DIR}/${OUT_BINARY_NAME}-${VERSION}-linux-
amd64.tar.gz"
  artifacts:
    paths:
    - "${BIN_DIR}/${OUT_BINARY_NAME}-*.tar.gz"
    expire_in: 1 days
  rules:
  - if: "$CI_COMMIT_TAG"
    when: manual
    allow_failure: true
  parallel:
    matrix:
    - SERVICE_NAME: gpsql
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: maskpass
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mclient
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: minica
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mjwt
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mocsp
```

```
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: signer
      OUT_BINARY_NAME: "$SERVICE_NAME"
  extends:
  - ".package:ARCHIVE"
  - ".service_list"
".package:DEB":
  stage: package
  image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
  script:
  - |
    cd ${DEB_BUILD_SCRIPT_PATH}
    bash dpkg-deb.sh
  artifacts:
    paths:
    - "${DEB_ARTIFACT_PATH}"
    expire_in: 1 day
  rules:
  - if: "$CI_COMMIT_TAG"
".package:SFX":
  stage: package
  image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
  script:
  - bash ${SFX_BUILD_SCRIPT_PATH}
  artifacts:
    paths:
    - "${SFX_ARTIFACT_PATH}"
    expire_in: 1 day
  rules:
  - if: "$CI_COMMIT_TAG"
".push:nexus":
  stage: push
  image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
  variables:
    ARTIFACT_NAME: "${OUT_BINARY_NAME}-${VERSION}-linux-amd64.tar.gz"
    ARTIFACT_PATH: "${BIN_DIR}/${ARTIFACT_NAME}"
  script:
  - "echo \"\U0001F4E4 Загружаем ${ARTIFACT} в Nexus...\""
  - |
```

```
    set -x
    curl --fail-with-body -k -v \
      -u "${NEXUS_USER}:${NEXUS_PASSWORD}" \
      --upload-file "${ARTIFACT_PATH}" \
      "https://repos.clearwayintegration.com/repository/release/${CI_PROJECT_PATH}/${VERSION}/${ARTIFACT_NAME}"
  - echo "✅ Успешно загружено в Nexus."
  rules:
  - if: "$CI_COMMIT_TAG"
  - if: $MONOREPO == null || $MONOREPO == ""
push:nexus:ARCHIVE:
  stage: push
  image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
  variables:
    ARTIFACT_NAME: "${OUT_BINARY_NAME}-${VERSION}-linux-amd64.tar.gz"
    ARTIFACT_PATH: "${BIN_DIR}/${ARTIFACT_NAME}"
  script:
  - "echo \"\U0001F4E4 Загружаем ${ARTIFACT} в Nexus...\""
  - |
    set -x
    curl --fail-with-body -k -v \
      -u "${NEXUS_USER}:${NEXUS_PASSWORD}" \
      --upload-file "${ARTIFACT_PATH}" \
      "https://repos.clearwayintegration.com/repository/release/${CI_PROJECT_PATH}/${VERSION}/${ARTIFACT_NAME}"
  - echo "✅ Успешно загружено в Nexus."
  rules:
  - if: "$CI_COMMIT_TAG"
    when: manual
    allow_failure: true
  parallel:
    matrix:
    - SERVICE_NAME: gpsql
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: maskpass
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mclient
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: minica
```

```
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mjwt
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mocsp
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: signer
      OUT_BINARY_NAME: "$SERVICE_NAME"
  extends:
  - ".push:nexus"
  - ".service_list"
  dependencies:
  - package:ARCHIVE
".delivery_to_vm_template":
  stage: delivery
  image: "${DEPLOY_TOOL_IMAGE_NAME}:${DEPLOY_TOOL_IMAGE_TAG}"
  variables:
    ANSIBLE_HOST_KEY_CHECKING: 'False'
  script:
  - echo $SYSTEM_NAME
  - cd /deploy-tool
  - git clone
https://auth:${GITLAB_INVENTORI_TOKEN}@gitlab.clearwayintegration.com/devops/envir
onments-configurations.git
  - cat environments-configurations/${SYSTEM_NAME}/${CI_PROJECT_NAME}/inventory.ini
  - echo "user - $ANSIBLE_USER"
  - ansible-playbook -i environments-
configurations/${SYSTEM_NAME}/${CI_PROJECT_NAME}/inventory.ini
    delivery-only.yml --limit $ENVIRONMENT --user $ANSIBLE_USER -e ansible_password="{{
    lookup('env', 'ANSIBLE_PASSWORD') }}" -e ansible_become_pass="{{ lookup('env',
    'ANSIBLE_PASSWORD') }}"
delivery_to_vm_dev:
  stage: delivery
  image: "${DEPLOY_TOOL_IMAGE_NAME}:${DEPLOY_TOOL_IMAGE_TAG}"
  variables:
    ANSIBLE_HOST_KEY_CHECKING: 'False'
    ENVIRONMENT: dev
    SYSTEM_NAME: miu
  script:
  - echo $SYSTEM_NAME
```

```
382:   - cd /deploy-tool
383:   - git clone
       https://auth:${GITLAB_INVENTORI_TOKEN}@gitlab.clearwayintegration.com/devops/envir
       onments-configurations.git
384:   - cat environments-configurations/${SYSTEM_NAME}/${CI_PROJECT_NAME}/inventory.ini
385:   - echo "user - $ANSIBLE_USER"
386:   - ansible-playbook -i environments-
       configurations/${SYSTEM_NAME}/${CI_PROJECT_NAME}/inventory.ini
387:     delivery-only.yml --limit $ENVIRONMENT --user $ANSIBLE_USER -e ansible_password="{{
388:     lookup('env', 'ANSIBLE_PASSWORD') }}" -e ansible_become_pass="{{ lookup('env',
389:     'ANSIBLE_PASSWORD') }}"
390:   parallel:
391:     matrix:
392:     - SERVICE_NAME: gpsql
393:       OUT_BINARY_NAME: "$SERVICE_NAME"
394:     - SERVICE_NAME: maskpass
395:       OUT_BINARY_NAME: "$SERVICE_NAME"
396:     - SERVICE_NAME: mclient
397:       OUT_BINARY_NAME: "$SERVICE_NAME"
398:     - SERVICE_NAME: minica
399:       OUT_BINARY_NAME: "$SERVICE_NAME"
400:     - SERVICE_NAME: mjwt
401:       OUT_BINARY_NAME: "$SERVICE_NAME"
402:     - SERVICE_NAME: mocsp
403:       OUT_BINARY_NAME: "$SERVICE_NAME"
404:     - SERVICE_NAME: signer
405:       OUT_BINARY_NAME: "$SERVICE_NAME"
406:   extends:
407:   - ".delivery_to_vm_template"
408:   - ".service_list"
409:   rules:
410:   - if: "$CI_COMMIT_TAG && $CI_COMMIT_TAG =~ $RC_TAG_REGEX"
411:     when: manual
412: delivery_to_vm_stage:
413:   stage: delivery
414:   image: "${DEPLOY_TOOL_IMAGE_NAME}:${DEPLOY_TOOL_IMAGE_TAG}"
415:   variables:
416:     ANSIBLE_HOST_KEY_CHECKING: 'False'
417:     ENVIRONMENT: stage
```

```
418:   script:
419:   - echo $SYSTEM_NAME
420:   - cd /deploy-tool
421:   - git clone
https://auth:${GITLAB_INVENTORI_TOKEN}@gitlab.clearwayintegration.com/devops/envir
onments-configurations.git
422:   - cat environments-configurations/${SYSTEM_NAME}/${CI_PROJECT_NAME}/inventory.ini
423:   - echo "user - $ANSIBLE_USER"
424:   - ansible-playbook -i environments-
configurations/${SYSTEM_NAME}/${CI_PROJECT_NAME}/inventory.ini
425:     delivery-only.yml --limit $ENVIRONMENT --user $ANSIBLE_USER -e ansible_password="{{
426:     lookup('env', 'ANSIBLE_PASSWORD') }}" -e ansible_become_pass="{{ lookup('env',
427:     'ANSIBLE_PASSWORD') }}"
428:   parallel:
429:     matrix:
430:     - SERVICE_NAME: gpsql
431:       OUT_BINARY_NAME: "$SERVICE_NAME"
432:     - SERVICE_NAME: maskpass
433:       OUT_BINARY_NAME: "$SERVICE_NAME"
434:     - SERVICE_NAME: mclient
435:       OUT_BINARY_NAME: "$SERVICE_NAME"
436:     - SERVICE_NAME: minica
437:       OUT_BINARY_NAME: "$SERVICE_NAME"
438:     - SERVICE_NAME: mjwt
439:       OUT_BINARY_NAME: "$SERVICE_NAME"
440:     - SERVICE_NAME: mocsp
441:       OUT_BINARY_NAME: "$SERVICE_NAME"
442:     - SERVICE_NAME: signer
443:       OUT_BINARY_NAME: "$SERVICE_NAME"
444:   extends:
445:   - ".delivery_to_vm_template"
446:   - ".service_list"
447:   rules:
448:   - if: "$CI_COMMIT_TAG && $CI_COMMIT_TAG =~ $RC_TAG_REGEX"
449:     when: manual
450: delivery_to_vm_preprod:
451:   stage: delivery
452:   image: "${DEPLOY_TOOL_IMAGE_NAME}:${DEPLOY_TOOL_IMAGE_TAG}"
453:   variables:
```

```
    ANSIBLE_HOST_KEY_CHECKING: 'False'
    ENVIRONMENT: preprod
  script:
  - echo $SYSTEM_NAME
  - cd /deploy-tool
  - git clone https://auth:${GITLAB_INVENTORI_TOKEN}@gitlab.clearwayintegration.com/devops/environments-configurations.git
  - cat environments-configurations/${SYSTEM_NAME}/${CI_PROJECT_NAME}/inventory.ini
  - echo "user - $ANSIBLE_USER"
  - ansible-playbook -i environments-configurations/${SYSTEM_NAME}/${CI_PROJECT_NAME}/inventory.ini
    delivery-only.yml --limit $ENVIRONMENT --user $ANSIBLE_USER -e ansible_password="{{
    lookup('env', 'ANSIBLE_PASSWORD') }}" -e ansible_become_pass="{{ lookup('env',
    'ANSIBLE_PASSWORD') }}"
  parallel:
    matrix:
    - SERVICE_NAME: gpsql
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: maskpass
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mclient
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: minica
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mjwt
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mocsp
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: signer
      OUT_BINARY_NAME: "$SERVICE_NAME"
  extends:
  - ".delivery_to_vm_template"
  - ".service_list"
  rules:
  - if: "$CI_COMMIT_TAG && $CI_COMMIT_TAG =~ $TAG_REGEX"
    when: manual
sonarqube-check:
  image:
```

```
    name: "${SONARQUBE_IMAGE}:${SONARQUBE_TAG}"
    entrypoint:
    - ''
  stage: sonarqube-check
  cache:
    key: sonar-cache
    paths:
    - ".sonar/cache"
  script:
  - |
    sonar-scanner \
      -Dsonar.projectKey="${SERVICE_NAME}" \
      -Dsonar.projectName="${SERVICE_NAME}" \
      -Dsonar.projectVersion="${VERSION}" \
      -Dsonar.sources="." \
      -Dsonar.host.url="${SONAR_HOST_URL}" \
      -Dsonar.login="${SONAR_TOKEN}"
  allow_failure: true
  rules:
  - if: "$CI_COMMIT_BRANCH"
stages:
- ".pre"
- verifications
- sonarqube-check
- build
- package
- test
- push
- release
- delivery
- ".post"
cache:
  key: "${CI_COMMIT_REF_SLUG}"
  paths:
  - go/pkg/mod
  - "~/.cache/go-build"
  policy: pull-push
".service_list":
  parallel:
```

```
    matrix:
    - SERVICE_NAME: gpsql
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: maskpass
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mclient
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: minica
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mjwt
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: mocsp
      OUT_BINARY_NAME: "$SERVICE_NAME"
    - SERVICE_NAME: signer
      OUT_BINARY_NAME: "$SERVICE_NAME"
package:DEB:
  stage: package
  image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
  script:
  - |
    cd ${DEB_BUILD_SCRIPT_PATH}
    bash dpkg-deb.sh
  artifacts:
    paths:
    - "${DEB_ARTIFACT_PATH}"
    expire_in: 1 day
  rules:
  - if: "$CI_COMMIT_TAG"
  variables:
    DEB_BUILD_SCRIPT_PATH: "${CI_PROJECT_DIR}/make-deb/"
    DEB_ARTIFACT_PATH: "${CI_PROJECT_DIR}/make-deb/*.deb"
  before_script:
  - chmod -R 755 ${DEB_BUILD_SCRIPT_PATH}dpkg-debian/DEBIAN
  extends: ".package:DEB"
package:SFX:
  stage: package
  image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
  script:
  - bash ${SFX_BUILD_SCRIPT_PATH}
```

```
  artifacts:
   paths:
   - "${SFX_ARTIFACT_PATH}"
   expire_in: 1 day
  rules:
  - if: "$CI_COMMIT_TAG"
  before_script:
  - apt-get update && apt-get install -y makeself xz-utils
  parallel:
   matrix:
   - SERVICE_NAME: minica -sfx
     SFX_BUILD_SCRIPT_PATH: "${CI_PROJECT_DIR}/make-sfx/make- minica -sfx.sh"
     SFX_ARTIFACT_PATH: "${CI_PROJECT_DIR}/make-sfx/ minica -sfx*.run"
   - SERVICE_NAME: mocsp-sfx
     SFX_BUILD_SCRIPT_PATH: "${CI_PROJECT_DIR}/make-sfx/make-mocsp-sfx.sh"
     SFX_ARTIFACT_PATH: "${CI_PROJECT_DIR}/make-sfx/mocsp-sfx*.run"
  extends: ".package:SFX"
push:nexus:DEB:
  stage: push
  image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
  variables:
   ARTIFACT_NAME: minica -${VERSION}_amd64.deb
   ARTIFACT_PATH: "${CI_PROJECT_DIR}/make-deb/${ARTIFACT_NAME}"
  script:
  - "echo \"\U0001F4E4 Загружаем ${ARTIFACT} в Nexus...\""
  - |
   set -x
   curl --fail-with-body -k -v \
     -u "${NEXUS_USER}:${NEXUS_PASSWORD}" \
     --upload-file "${ARTIFACT_PATH}" \
     "https://repos.clearwayintegration.com/repository/release/${CI_PROJECT_PATH}/${VERSION}/${ARTIFACT_NAME}"
  - echo "✅ Успешно загружено в Nexus."
  rules:
  - if: "$CI_COMMIT_TAG"
  - if: $MONOREPO == null || $MONOREPO == ""
  dependencies:
  - package:DEB
  extends: ".push:nexus"
```

```
606:  push:nexus:SFX:
607:   stage: push
608:   image: docker-registry.clearwayintegration.com/devinfra/cwirunner:1.0.6
609:   variables:
610:    ARTIFACT_NAME: "${OUT_BINARY_NAME}-${VERSION}-linux-amd64.tar.gz"
611:    ARTIFACT_PATH: "${BIN_DIR}/${ARTIFACT_NAME}"
612:   script:
613:   - "echo \"\U0001F4E4 Загружаем ${ARTIFACT} в Nexus...\""
614:   - |
615:    set -x
616:    curl --fail-with-body -k -v \
617:       -u "${NEXUS_USER}:${NEXUS_PASSWORD}" \
618:       --upload-file "${ARTIFACT_PATH}" \
619:       "https://repos.clearwayintegration.com/repository/release/${CI_PROJECT_PATH}/${VE
     RSION}/${ARTIFACT_NAME}"
620:   - echo "✅ Успешно загружено в Nexus."
621:   rules:
622:   - if: "$CI_COMMIT_TAG"
623:   - if: $MONOREPO == null || $MONOREPO == ""
624:   dependencies:
625:   - package:SFX
626:   parallel:
627:    matrix:
628:    - ARTIFACT_NAME: minica-sfx-${VERSION}_amd64.run
629:      ARTIFACT_PATH: "${CI_PROJECT_DIR}/make-sfx/${ARTIFACT_NAME}"
630:    - ARTIFACT_NAME: mocsp-sfx-${VERSION}_amd64.run
631:      ARTIFACT_PATH: "${CI_PROJECT_DIR}/make-sfx/${ARTIFACT_NAME}"
632:   extends: ".push:nexus"
```

6.3.5. Характеристики стенда для сборки

Сборка осуществляется с использованием контейнеров на основе Docker, размещаемых в инфраструктуре организации. Характеристики стенда указаны Таблице 4.

Таблица 4 — Характеристика стенда для сборки

ОС	Архитектура	Процессор	Оперативная память	ПЗУ
Ubuntu 24.04	X86-64	48 Ядер Intel Xeon Gold	256 ГБ	16 ГБ

6.4. Среда хранения артефактов

Среда хранения артефактов проекта — централизованная инфраструктура для безопасного и удобного хранения артефактов (архивов, бинарных файлов, пакетов приложений), созданных в ходе разработки и сопровождения проекта.

6.4.1. Система хранения артефактов

В качестве хранения артефактов используется Sonatype Nexus Repository Community Edition — корпоративный репозиторий программного обеспечения, обеспечивающий централизованное управление артефактами, такими как архивы, бинарные файлы, пакеты приложений и прочие ресурсы, необходимые для разработки и развертывания программного обеспечения.

Основные возможности Sonatype Nexus Repository Community Edition:

- Центральное хранилище для всех артефактов проекта;
- Поддержка популярных форматов (Maven, Docker, npm, NuGet и др.);
- Простая интеграция с инструментами сборки и CI/CD (Jenkins, GitLab, CircleCI);
- Локальное кэширование публичных репозиториев для экономии трафика;
- Удаление старых и неиспользуемых артефактов для освобождения дискового пространства.

6.5. Среда поиска уязвимостей

Среда поиска уязвимостей — это специализированная инфраструктура и набор инструментов, предназначенный для выявления недостатков в информационной безопасности информационных систем, приложений.

6.5.1. Сканеры уязвимостей

Для повешения качества и безопасности программного обеспечения используются следующие инструменты:

- Dependency Check — утилита для анализа зависимостей проекта с целью выявления известных уязвимостей. Определяет слабые звенья в используемых библиотеках и формирует рекомендации по их замене или обновлению;
- SonarQube Community Build — платформа для статического анализа кода, применяемая для поиска багов, потенциальных ошибок и улучшения общего

качества кода. Выделяет недостатки в структуре проекта и даёт рекомендации по повышению его устойчивости и читаемости;

- Svace — средство анализа безопасности исходного кода, которое ищет уязвимости и отклонения от нормативных требований. Помогает предотвратить утечки данных и соблюдение стандартов информационной безопасности.

Подробная информация о сканерах уязвимостей приведена в документе "Программное обеспечение. Система централизованного анализа и управления гетерогенными инфраструктурами – MiniCA. Безопасность разработки RU.CLRW.00947-01 97 01".

6.6. Среда тестирования проекта

Среда тестирования проекта — это специально созданная инфраструктура, предназначенная для проверки работоспособности программного обеспечения и выявления дефектов. Она включает аппаратные и программные компоненты, настроенные для выполнения тестов в условиях, близких к производственной среде.

6.6.1. Средства и методы тестирования

Виды тестирования:

- Модульное тестирование (Unit tests) – проверка отдельных компонентов (go test);
- Интеграционное тестирование – проверка взаимодействия модулей (связка OpenSSL + MiniCA);
- Функциональное тестирование – проверка соответствия требованиям. Ручное тестирование сценариев выпуска\отзыва\возвращения сертификатов через все поддерживаемые протоколы (SCEP, MS-WSTEP).

Процесс контроля качества:

- Тест-кейсы хранятся в Zephyr for Jira;
- Автоматизированные проверки запускаются при каждом коммите (CI/CD через GitLab CI);
- Ручное тестирование выполняется перед релизом (чек-лист основных сценариев).

Управление дефектами:

- Все найденные ошибки фиксируются в баг-трекере Jira.
- Критические баги исправляются до выпуска версии.

В качестве средств тестирования используются встроенная утилита go test языка Go, сторонняя утилита golanci-lint, с открытым исходным кодом и Zephyr for Jira:

- go test — встроенный утилита для автоматического тестирования Go-программ. Позволяет запускать юнит-, интеграционные и benchmark-тесты, оценивая корректность и производительность кода;
- golangci-lint — утилита для статического анализа, включающий более сотни линтеров. Проверяет код на наличие потенциальных ошибок, плохую практику программирования и нарушения стиля, улучшая качество и консистентность кода;
- Zephyr for Jira — плагин для Jira, предназначенный для управления тестированием, планирования, анализа требований и составления отчетов о тестировании.

Подробная информация о средствах тестирования приведена в документе "Программное обеспечение. Система централизованного анализа и управления гетерогенными инфраструктурами - MiniCA. Тестовая документация RU.CLRW.00947-01 51 01".

Ответственные:

- Разработчики – выполнение модульного тестирования;
- QA-инженер – выполнение интеграционного и ручного тестирования.

6.7. Среда развертывания

Среда развертывания — инфраструктура, на которой происходит установка и эксплуатация программного обеспечения. Включает аппаратные и программные ресурсы, необходимые для нормальной работы приложения. Для MiniCA используется контейнеризация, позволяющая запускать приложения в изолированных контейнерах, обеспечивая переносимость и консистентность среды между разными этапами жизненного цикла продукта.

6.7.1. Система управления контейнерами

В качестве инструментов для управления контейнерами и развертыванием приложений используются:

- K8s (Kubernetes) — платформа для управления контейнерами, обеспечивающая автоматизацию развертывания, масштабирования и восстановления контейнеризированных приложений. Kubernetes решает задачи отказоустойчивости, распределения нагрузок и автоматического масштабирования.
- Containerd —движок для запуска и управления контейнерами.

6.8. Среда хранения документов

Среда хранения документов проекта — централизованная информационная инфраструктура, обеспечивающая сбор, упорядоченность и доступность проектной документации для всех участников процесса. Основная цель заключается в обеспечении быстрого доступа к необходимым материалам (макетам, техзаданиям, планам-графикам и др.) и прозрачности ведения проекта.

Хранение документации осуществляется в Confluence — корпоративной платформе для совместного ведения документации, организации рабочих групп и общения сотрудников. Доступ к системе возможен только после успешного прохождения процедур идентификации и аутентификации. В случае неуспешного результата идентификации и аутентификации, доступ невозможен.

Для хранящихся в системе Confluence указываются их обозначения, наименования, версия. Платформа позволяет однозначно идентифицировать дату и время внесения изменений в документ, а также имя пользователя, внесшего изменения.

Доступ к документации разрешен на чтение для всех сотрудников структурных подразделений, доступ на чтение и запись имеют только разработчики и технические писатели.

Финальные электронные версии эксплуатационных документов выгружаются в виде pdf-файлов и docx-файлов.

6.9. Средства разработки документации

В качестве средств разработки документации используются:

- Atlassian Confluence – тиражируемая вики-система для внутреннего использования организациями с целью создания единой базы знаний. Огромное множество шаблонов для быстрого начала работы. Confluence предоставляет пользователям большое количество шаблонов для создания новый страниц и позволяет вести совместную работу с документами в режиме реального времени.
- Microsoft Word – текстовый процессор, предназначенный для создания, просмотра, редактирования и форматирования текстов статей, деловых бумаг, а также иных документов, с локальным применением простейших форм таблично-матричных алгоритмов.
- Draw.IO – используется в виде десктопного приложения и ПО, помогая создавать блок-схемы, прототипы, инфографику и диаграммы любого вида. Поддерживает различные нотации, в том числе BPMN (Business Process Model and Notation), UML-диаграммы (Unified Modeling Language) и др.
- PlantUML – универсальный инструмент, позволяющий быстро и просто создавать широкий спектр диаграмм. С помощью PlantUML, используя простой и

интуитивно понятный язык, можно создавать хорошо структурированные UML-диаграммы, такие как «Диаграмма последовательности», «Диаграмма компонентов» и пр. PlantUML интегрирован с Atlassian Confluence для улучшения рабочего процесса.

- Figma – это популярный инструмент для дизайнеров. В программе создают интерфейсы приложений, веб-сайтов, прототипы и графические элементы для самых разных проектов.

6.10. Среда планирования работ и управленческого учета

Среда планирования работ и управленческого учета — система процессов и инструментов, направленная на планирование, контроль и координацию действий в рамках проекта.

6.10.1. Система планирования работ и управленческого учета

Планирование работ и управленческий учет осуществляется в Jira — программном решении для управления проектами и отслеживания задач. Для совершения каких-либо действий, необходимо аутентифицироваться.

Общие возможности Jira:

- Формирование и контроль выполнения задач командой.
- Отображение текущего статуса задач, сроков и исполнителей.
- Подробная аналитика, статистика выполнения, графики загрузки команды.
- Настройка ролей и уровней доступа для сотрудников.
- Канбан-доски и доски Scrum для визуального отслеживания этапов работы.
- Многопроектность с индивидуальными структурами задач и составами команд.
- Поддержка большого числа пользователей и параллельного ведения проектов.

При этом для каждого проекта указывается свой список ролей пользователей, которые могут получить к нему доступ, полномочия пользователей в проекте также разграничиваются. Для каждой задачи в проекте указывается тема, описание, указываются сроки выполнения, исполнитель, наблюдатель за выполнением, ведется процент готовности задачи, имеется возможность указания приоритета и статуса задачи, есть возможность прикреплять материалы и оставлять комментарии.