

# Design Document: EcoGrid VPP Core

## Platform

**Проект:** EcoGrid Virtual Power Plant Simulator **Версия:** 1.0 (Release Candidate)  
**Статус:** Утверждено к разработке **Классификация:** Внутреннее использование / R&D

# 1. Глоссарий и терминология

*Для обеспечения единого понимания контекста специалистами разного профиля вводятся следующие определения:*

- **VPP (Virtual Power Plant, Виртуальная электростанция):** Облачная система, объединяющая сотни мелких источников энергии (панели, батареи) для управления ими как одной большой электростанцией.
- **DER (Distributed Energy Resources):** Распределенные энергетические ресурсы — физические устройства (солнечные панели, ветряки, накопители), расположенные у потребителя.
- **Digital Twin (Цифровой двойник):** Программная модель физического устройства, которая с высокой точностью имитирует его поведение, реакции на команды и износ без использования реального "железа".
- **Arbitrage (Энергетический арбитраж):** Стратегия заработка, основанная на покупке электроэнергии в часы низких цен (ночью) и продаже или использовании накопленной энергии в часы пик (днем).
- **Peak Shaving (Срезание пиков):** Снижение потребления электричества из центральной сети в моменты максимальной нагрузки для предотвращения штрафов или переплат.
- **MQTT:** Легковесный протокол обмена сообщениями, стандарт де-факто в индустрии интернета вещей (IoT) для связи устройств с сервером.

## **2. Исполнительное резюме (Executive Summary)**

**EcoGrid VPP** — это программно-аппаратный комплекс для симуляции и оркестрации распределенных энергетических ресурсов. Платформа предназначена для моделирования сценариев управления микросетями (Microgrids) частных домохозяйств и малых сообществ.

**Бизнес-цель:** Создание масштабируемой модели для демонстрации экономической эффективности внедрения «умных» сетей. Система позволяет доказать возможность снижения операционных затрат на электроэнергию (OpEx) и генерации новой выручки за счет участия в рынках электроэнергии (балансировка частоты, арбитраж).

**Ключевое преимущество:** Использование технологии цифровых двойников позволяет моделировать поведение сети из сотен устройств (high-load сценарии) без капитальных затрат на закупку физического оборудования.

### 3. Архитектура системы

Система построена на принципах **Cloud-Native микросервисной архитектуры**. Разделение ответственности между языками программирования **Go** и **Python** обусловлено спецификой решаемых задач.

#### 3.1. Технологический стек и обоснование выбора

##### *A. Уровень периферийных вычислений и транспорта (Edge & Ingestion)*

- **Технология: Go (Golang).**
- **Роль:** «Нервная система» платформы.
- **Обоснование:**
  - **Производительность:** Go обеспечивает минимальную задержку (latency) при обработке тысяч сообщений от устройств в секунду.
  - **Параллелизм:** Механизм *goroutines* позволяет эффективно управлять тысячами одновременных подключений от эмулируемых устройств (солнечных панелей, инверторов) с минимальным потреблением оперативной памяти.
  - **Надежность:** Строгая типизация и бинарная компиляция исключают класс ошибок, свойственных интерпретируемым языкам, что критично для сервисов, работающих 24/7.

##### *B. Уровень аналитики и оптимизации (Optimization Engine)*

- **Технология: Python.**
- **Роль:** «Мозг» платформы.
- **Обоснование:**
  - **Математический аппарат:** Python является стандартом индустрии для Data Science. Использование библиотек **Promo** (линейное программирование) и **Pandas** позволяет реализовывать сложные алгоритмы экономической оптимизации, недоступные или трудоемкие для реализации на компилируемых языках.
  - **Гибкость:** Возможность быстро менять эвристические модели прогнозирования погоды и цен без пересборки ядра системы.

### *C. Уровень данных и коммуникации*

- **Message Broker:** MQTT (Mosquitto/RabbitMQ). Обеспечивает асинхронную связь «многие-ко-многим» между устройствами и ядром.
- **Storage:** TimescaleDB (PostgreSQL extension). Специализированная база данных для временных рядов. Позволяет хранить историю телеметрии (метрики каждые N секунд) и быстро выполнять аналитические запросы за большие периоды.

## **4. Функциональные требования и спецификация модулей**

### **4.1. Подсистема эмуляции устройств (Device Emulation Layer)**

*Реализация: Go, Docker Containers*

Система должна создавать изолированные цифровые двойники для следующих типов оборудования:

1. **PV Generation (Солнечные панели):**
  - Генерация на основе стохастической модели инсоляции (учет времени суток, облачности).
2. **Wind Turbine (Ветрогенератор):**
  - Моделирование вертикально-осевых турбин с учетом порывов ветра и инерции вращения.
3. **ESS (Energy Storage System / Батареи):**
  - Симуляция химических процессов: кривая заряда/разряда, деградация емкости (SoH), зависимость КПД от температуры.
4. **EV (Электромобиль с V2G):**
  - Поддержка технологии Vehicle-to-Grid (возврат энергии в сеть). Учет расписания доступности (автомобиль подключен только вечером и ночью).
5. **Smart Load (Тепловой насос):**
  - Использование тепловой инерции здания как аккумулятора энергии.

### **4.2. Подсистема сбора данных (Data Ingestion Service)**

*Реализация: Go*

- **Функции:**
  - Подписка на MQTT-топики всех устройств.
  - Валидация входящих пакетов (JSON Schema/Protobuf).
  - Пакетная запись (Batch Insert) в TimescaleDB для снижения нагрузки на диск.
  - Фильтрация аномальных значений (защита от "шумящих" датчиков).

### **4.3. Подсистема оптимизации (Optimization Engine)**

*Реализация: Python*

- **Цикл управления (Control Loop):** Запуск каждые 15 минут.
- **Входные векторы:**
  - Прогноз погоды (генератор данных на основе исторических паттернов).
  - Динамические тарифы рынка (RTP — Real Time Pricing).
  - Текущий уровень заряда накопителей (SoC).
- **Задачи оптимизатора:**
  - **Scenario A (Arbitrage):** Рассчитать план зарядки батареи в «дешевые» часы для продажи энергии в «дорогие».
  - **Scenario B (Peak Shaving):** Обнаружить прогноз пикового потребления и запланировать разрядку батареи для компенсации дефицита.
- **Выход:** Матрица команд для каждого устройства на следующий временной слот.

#### **4.4. Подсистема управления (Command & Control)**

*Реализация: Go*

- Трансляция решений оптимизатора в проприетарные команды устройств.
- Публикация управляющих сигналов в MQTT с гарантией доставки (QoS 1).
- Мониторинг исполнения команд (Feedback Loop).

## 5. Требования для департамента АСУТП и IoT

Для корректной интеграции с моделью физического мира необходимо соблюдать следующие требования к эмуляции:

1. **Дискретизация сигналов:** Устройства должны публиковать телеметрию с интервалом не более 1 секунды для критических параметров (напряжение, частота) и 1 минуты для метрических (температура, заряд).
2. **Обработка задержек:** Система управления должна быть устойчива к сетевым задержкам до 500 мс (Jitter tolerance).
3. **Fail-safe логика:** В случае потери связи с центром управления цифровой двойник должен переходить в безопасный режим (Safe Mode) — прекращение генерации или переход в автономное потребление.

## **6. Нефункциональные требования (SLA)**

### **1. Масштабируемость (Scalability):**

- Архитектура должна поддерживать горизонтальное масштабирование через **Docker Compose** (для локальных стендов) и **Kubernetes** (для продуктовой среды).
- Целевая нагрузка: эмуляция до 100+ устройств на одном узле стандартной конфигурации (8 vCPU, 16GB RAM).

### **2. Производительность:**

- Пропускная способность шины данных: не менее 1000 сообщений в секунду.
- Latency (время реакции системы на событие): < 1 секунды.

### **3. Observability (Наблюдаемость):**

- Все микросервисы должны экспортировать метрики в формате **Prometheus**.
- Визуализация состояния системы и финансовых показателей через дашборды **Grafana**.

## **7. План реализации и приемочное тестирование**

Проект считается завершенным при успешном прохождении следующих сценариев:

1. **Интеграционный тест:** Разворачивание всей инфраструктуры (БД, Брокер, Сервисы) одной командой (docker-compose up -d).
2. **Стресс-тест:** Стабильная работа при симуляции "шторма" (резкое падение генерации солнечных панелей) и одновременном скачке рыночных цен.
3. **Экономический валидатор:** Генерация отчета, подтверждающего положительный экономический баланс (прибыль) виртуальной станции по итогам 24-часовой симуляции.