?????????????????? ?
??????????????
??????????????
??????????????

??????????

Эта статья разбирает отказоустойчивую и масштабируемую архитектуру обработки заданий печати, копирования и сканирования.

????? ????? ??? ?????

Базовой конфигурации из одного ПринтМенеджера достаточно, пока хватает производительности одного сервера и нет требований по отказоустойчивости. Горизонтальное масштабирование и балансировка нагрузки решают задачи:

- **Производительность** — несколько ПринтМенеджеров обрабатывают задания параллельно, увеличивая общую пропускную способность.
- **Отказоустойчивость** — потеря одного или нескольких серверов не останавливает обработку заданий на оставшихся.

???????????? ????????????

Главное архитектурное отличие этой схемы от базовой конфигурации в наборе компонентов и в взаимном расположении: несколько равноправных узлов обработки перед общими базой данных и реплицированным хранилищем.

??? ???????? ????????????
??????? ????????????

Компонент базовой конфигурации	Что происходит в кластере
Nginx как точка входа	Точкой входа становится **балансировщик HAProxy**. Это реверс-прокси, но умеющий распределять запросы между несколькими активными ПринтМенеджерами по определенным правилам. Сам nginx не пропадает, но перестает быть точкой входа.
Redis (брокер)	Улучшается до master-slave **кластера с поддержкой Redis Sentinel,** который обеспечивает отказоустойчивость кластера Redis.
PostgreSQL	Становится **общей базой данных** на весь кластер — логически один компонент, к которому обращаются все узлы. Внутри эта база данных должна быть физически или логически реплицирована.
Локальное хранилище образов документов CUPS	Меняется на **общее NFS-хранилище** — папка которая доступна на чтение и запись каждому приложению, т.е. задание может быть создано на одном узле, а прочитано на другом. По одному CUPS на каждый из узлов, каждый из CUPS содержит одинаковое количество принтеров. Поскольку CUPS становится больше, то нагрузка на каждый из них становится меньше, а значит возможна печать большего количества документов одновременно без перегрузки.
Celery (один экземпляр)	По одному Celery на каждом узле — обрабатывают различные задачи; именно это даёт прирост производительности.
Django	По одному Django на каждом узле. Поскольку Django становится больше, они независимо обрабатывают разные запросы извне
Планировщик	Становится **распределённым**: экземпляр существует на каждом узле, но согласовывает запуск периодических задач через Redis Sentinel, чтобы одна и та же задача не запускалась одновременно на всех узлах.
Конвертер-сервер	Не меняет своих функций, но иным способом обрабатывает TCP запросы полученные от сокета HAProxy, чтобы правильно определять адрес физического ковертера.

Несмотря на появление HAProxy, Redis Sentinel и общего NFS-хранилища, архитектура обработки самого задания принципиально не меняется. Все изменения направлены на одно: чтобы несколько экземпляров ПринтМенеджера могли совместно обрабатывать задания, используя общие данные и не мешая друг другу.

??? ?????????????? ??????

В базовой конфигурации клиент отправляет запрос напрямую на единственный сервер: nginx → Django → Celery → CUPS.

В кластере тот же путь проходит через дополнительную точку координации:

1. Клиент отправляет запрос не на конкретный сервер, а на **балансировщик**.
2. Балансировщик направляет его на один из активных узлов.
3. На выбранном узле запрос обрабатывается как обычно: nginx → Django.
4. Django передаёт задачу через Redis — кластер координирует, чтобы её взял в работу ровно один свободный Celery, независимо от того, на каком узле он находится.
5. Метаданные задания кладутся в общую PostgreSQL, образ документа — в общее NFS-хранилище.
6. Печать выполняет тот узел, куда Celery отправитзадачу.

Поскольку Celery и Django на каждом узле работает независимо и асинхронно друг относительно друга, то с ростом числа узлов растёт и пропускная способность по заданиям — это и есть причина увеличивать число узлов ради производительности, отдельно от вопроса отказоустойчивости.

??????? ??????????????????

Основа отказоустойчивости – это возможности системы перераспредлять запросы и задачи с узлов, которые отказали, на работающие узлы.

- для перераспределения запросов нужен HAProxy
- для перераспределения задач нужен Redis Sentinel

HAProxy внутри себя имеет реплику, и внешний трафик с запросами может быть направлен на одну или другую реплику с использованием различных механизмов. Балансировщик определяет доступен ли узел, и если нет, то отправляет запрос в первый встречный работающий узел.

Redis Sentinel кластеру для работы необходим **кворум** — большинство узлов Redis должно оставаться активным: активных узлов должно быть строго больше, чем неактивных. Если это условие нарушается, Redis Sentinel не могут договориться, кто из оставшихся узлов должен взять на себя роль мастера, и кластер теряет работоспособность целиком.

Узлов в кластере	Можно потерять одновременно
3	1
5	2
7	3

Классический пример — 3 узла (1 мастер + 2 слейва): такая конфигурация переживает потерю любого одного узла, но не двух сразу.

Revision #2
Created 2026-07-12 15:12:52 UTC by DD
Updated 2026-07-12 15:15:01 UTC by DD