Материалы к экзамену CCF — полностью

Все семь уроков одной страницей. Для чтения подряд и для сохранения в PDF: печать из браузера даёт готовый файл.

1. Из чего собран Cozystack

Три слоя, четыре варианта установки и почему платформа — это просто ещё один набор объектов Kubernetes.

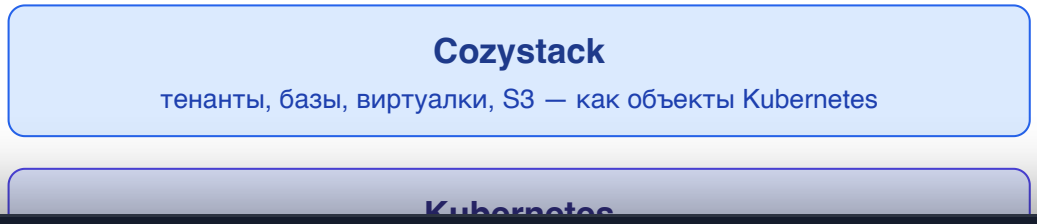
Начнём с вопроса, который на экзамене задают чаще любого другого: **что такое Cozystack по своей природе?** Не «для чего он», а именно из чего сделан.

Ответ короткий: это Kubernetes, которому добавили новые типы объектов. Не форк, не надстройка над чужим API, не отдельный сервер управления. Когда вы просите платформу завести базу данных, вы создаёте объект — такой же, как под или сервис, только называется он `Postgres`. Дальше за ним следит программа-оператор и делает всё остальное.

Это стоит уложить в голове до всего прочего, потому что из этого следует почти всё остальное. Работает `kubectl`? Работает. Работает Terraform? Работает. Права раздаются обычным RBAC? Да, обычным.

Три слоя

Платформа собрана из трёх этажей, и путать их порядок — самая частая ошибка на экзамене.



Talos Linux — операционная система узлов (node OS). Её особенность в том, что у неё нет командной строки: ни SSH, ни консоли, ни пакетного менеджера. Настраивается она декларативно, через свой API: вы отправляете описание желаемого состояния, узел приводит себя к нему. Корневая файловая система при этом только для чтения.

Для администратора, привыкшего зайти на сервер и что-нибудь поправить, это поначалу неуютно. Смысл в другом: если зайти и поправить нельзя, то и состояние узла не расползается со временем. Два узла, собранные по одному описанию, останутся одинаковыми через год.

Ставят при этом не стоковый Talos, а **собственный образ платформы**: в него добавлены модули ядра DRBD — на нём держится репликация LINSTOR — и ZFS. В обычном образе их нет.

Kubernetes — управляющий кластер (management cluster), который работает поверх Talos. Обычный, не переделанный: тот же API, те же контроллеры.

Здесь легко смешать два разных понятия. Управляющий кластер один — это и есть платформа. А кластеры, которые заказывают тенанты (tenant clusters), — уже её продукт: их управляющий слой работает подами внутри управляющего кластера.

Cozystack — набор компонентов, устанавливаемых в этот кластер. Именно они добавляют новые типы объектов и следят за ними.

Установка в три этапа

Экзамен спрашивает не команды, а порядок и смысл этапов.

1. **Talos на узлах.** Машины загружаются с образа платформы и получают машинное описание. Способов три: `boot-to-talos` — установщик пишет Talos на диск из уже запущенного Linux, загрузка с **ISO** и сетевая загрузка **PXE**. PXE

3. **Cozystack.** В кластер ставится сама платформа, дальше она разворачивает свои компоненты сама. Ставят её чартом `cozy-installer` в пространство имён `cozy-system`, а затем применяют один YAML — **Platform Package**. Это единственная точка настройки: домен платформы, адрес apiserver, диапазоны адресов и выбранный вариант установки.

Третий этап устроен любопытно: платформа не устанавливает компоненты по списку, а описывает желаемый состав и передаёт его **FluxCD**. Дальше Flux сам приводит кластер к описанию и продолжает следить, чтобы состав не расходился. Обновление платформы — это смена версии в описании, а не пересборка руками.

Каждый компонент платформы Flux ставит отдельным **релизом** — объект называется `HelmRelease`, в командах сокращается до `hr`. Поэтому вопрос «встала ли платформа» — это вопрос о релизах, а не о подах:

```
kubectl get hr -A
```

Все в состоянии `READY  True` — установка завершена.

Четыре варианта установки

Платформу ставят не целиком, а одним из четырёх наборов. Названия надо знать **дословно** — и вот почему это не придирка: в версии 1.5 их переименовали, а старые имена (`paas-full`, `distro-full` и подобные) остались жить в чужих статьях. Перепутать легко.

Вариант	Что это	Когда ваш случай
`isp-full`	вся платформа на Talos	своё железо, нужно всё сразу
`isp-full-generic`	то же, но на чужом Kubernetes	кластер уже есть: k3s, kubeadm, RKE2

Запомнить проще через одну фразу: `isp-full` — всё на Talos, `-generic` — то же на других дистрибутивах, `-hosted` — только платформа, инфраструктура снизу чужая, `default` — с нуля и вручную.

Выбирают один раз, при установке. Сменить набор на работающей платформе — это не переключатель, а переустановка.

Вариант задаёт состав по умолчанию, а тонкая настройка идёт **пакетами** (package). Каждый компонент платформы — пакет с именем вида `cozystack.<компонент>`, и посмотреть их состояние можно одной командой:

```
kubectl get package
```

Подправить набор дают два ключа в Platform Package: `bundles.enabledPackages` добавляет пакеты сверх варианта, `bundles.disabledPackages` — убирает. Оговорка, которую спрашивают: отключать пакеты можно только **до установки**. С работающей платформы компонент так не снимется — убирать придётся руками через Helm.

Из чего она состоит

Компонентов много, но на экзамене спрашивают, **кто за что отвечает**, а не полный список. Держите в голове такую карту:

Задача	Компонент
Виртуальные машины	KubeVirt
Управляющий слой тенантских кластеров	Kamaji
Дисковое хранилище	LINSTOR / DRBD
Объектное хранилище (S3)	SeaweedFS
Сеть подов	Cilium
Сеть тенантов, VPC	Kube-OVN
Внешние адреса	MetalLB
Метрики	VictoriaMetrics
Журналы	VictoriaLogs
Графики	Grafana
Доставка конфигурации	FluxCD
Вход по учётным записям	Keycloak

Строку про **Kamaji** стоит запомнить отдельно: он поднимает управляющий слой тенантских кластеров подами прямо в управляющем кластере — отдельных машин под мастера тенанту не надо.

Обратите внимание, чего в списке **нет**: Prometheus и Loki. Их часто подставляют в варианты ответа как ловушку — метрики и журналы здесь собирают

LINSTOR. Меньше трёх узлов не даёт отказоустойчивости ни хранилищу, ни управляющему слою.

Сеть нормирована не слабее: все узлы в **одном сегменте L2**, задержка между ними — **меньше 10 мс** по времени обращения (RTT).

Если узлы сами виртуальные — стенд в vSphere или Proxmox, — включите вложенную виртуализацию и проброс флагов процессора. На железе это не нужно. Домашний стенд по этому минимуму — законный способ пройти весь getting-started целиком.

Платформа собрана. Дальше её надо между кем-то поделить — этим и займётся следующий урок.

Что спросят на экзамене

- Порядок слоёв снизу вверх: Talos → Kubernetes → Cozystack.
- Что Talos управляется через API и не имеет SSH, а корень доступен только на чтение.
- Что платформа расширяет Kubernetes своими типами объектов, а не заменяет его.
- Названия четырёх вариантов установки (bundles) — дословно.
- Кто за что отвечает: KubeVirt, LINSTOR, SeaweedFS, Cilium, Kube-OVN, MetalLB.
- Что метрики хранит VictoriaMetrics, а не Prometheus.
- Что Kamaji держит управляющий слой тенантских кластеров подами в управляющем кластере.
- Чем управляющий кластер (management cluster) отличается от тенантских кластеров.
- Что пакеты называются `cozystack.<компонент>`, список даёт `kubectl get package`.
- Ключи `bundles.enabledPackages` и `bundles.disabledPackages`, отключение — только до установки.
- Что Talm — рекомендованный инструмент начальной настройки Talos и сборки кластера.
- Зачем платформе свой образ Talos: модули ядра DRBD и ZFS.
- Способы установки Talos: boot-to-talos, ISO, PXE (Matchbox отдаёт образ, dnsmasq — DHCP и TFTP).
- Что платформу ставит `cozy-installer` в `cozy-system`, а настраивает Platform Package.
- Минимум железа: 3 узла, 8 ядер, 24 ГБ, диски 50 и 256 ГБ, один сегмент L2, задержка до 10 мс.
- Что готовность установки проверяют по состоянию релизов FluxCD.

2. Тенанты и доступ

Как платформа делит себя между тенантами, откуда берутся квоты и почему тенант — это не просто namespace.

Вторая по весу тема экзамена — и, пожалуй, самая полезная на практике. Если первый урок объяснял, из чего платформа собрана, то этот — как она делится между людьми.

Тенант — это объект, а не папка

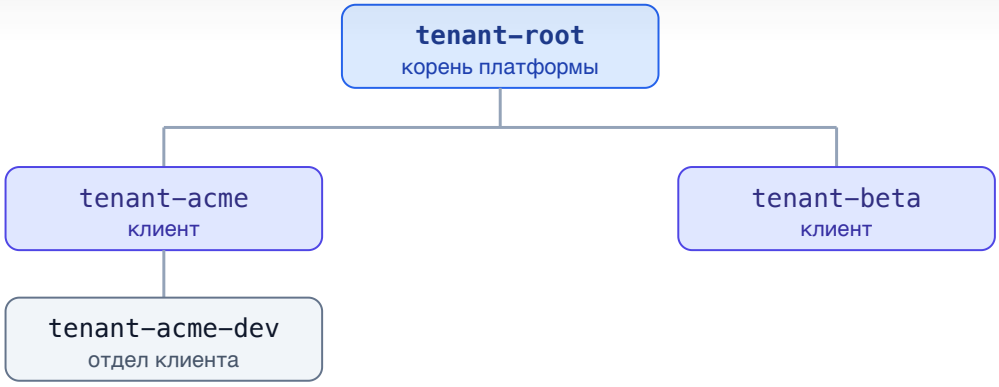
В привычном Kubernetes изоляция начинается и заканчивается пространством имён. Здесь иначе: **тенант (tenant) — это объект платформы**, и создание его влечёт за собой целый набор последствий.

Заводите вы его так же, как всё остальное:

```
apiVersion: apps.cozystack.io/v1alpha1
kind: Tenant
metadata:
  name: acme
  namespace: tenant-root
```

Платформа в ответ создаёт пространство имён (namespace), раздаёт права, вешает сетевые политики, заводит квоты и, если попросили, поднимает внутри собственный мониторинг и хранилище.

Заодно тенант получает свой домен, и собирается он по правилу `<имя>.<домен родителя>`. Платформа живёт на `cloud.example.com` — значит, у тенанта `alpha` будет `alpha.cloud.example.com`, а у его ребёнка `beta` —



Тенант живёт внутри родителя. Имя пространства имён складывается из слова tenant- и имени.

Правило именования спрашивают, и оно чуть хитрее, чем кажется: имя пространства имён собирается из слова `tenant-` и **всей цепочки предков через дефис, кроме корня**.

Путь тенанта	Пространство имён
`root/acme`	`tenant-acme`
`root/alpha/beta`	`tenant-alpha-beta`
`root/a/b/c`	`tenant-a-b-c`

Корень в имя не попадает — `tenant-root-alpha-beta` будет неверным ответом.

Отсюда же ограничение на имена: **дефис в имени тенанта запрещён**, потому что он занят под разделитель предков. Тенант `acme-dev` не существует — существует `dev` внутри `acme`, и живёт он в `tenant-acme-dev`.

В самом объекте эти два имени лежат в разных полях, и их различие тоже

Что наследуется, а что включается

У тенанта есть ровно четыре переключателя, и на экзамене спрашивают их буквальные имена: `etcd` — свой кластер etcd, `monitoring` — свой мониторинг, `ingress` — свой контроллер входящего трафика с TLS, `seaweedfs` — своё объектное хранилище S3. Каждый можно включить или оставить выключенным.

Ключевая мысль, которую любят проверять: **если сервис выключен, тенант поднимается вверх по дереву до ближайшего предка, у которого он включён**. Не обязательно до родителя и не обязательно до корня — до ближайшего. Не остаётся без него, а наследует. Тенант без своего мониторинга шлёт метрики в мониторинг родителя; тенант без своего хранилища кладёт файлы в родительское.

Включать своё имеет смысл, когда нужна настоящая изоляция — например, клиент не должен видеть чужие метрики даже теоретически. Плата за это — ресурсы: каждый включённый сервис это реально работающие процессы.

Квоты и переподписка

Тенанту назначают предел по процессорам и памяти:

```
spec:
  resourceQuotas:
    cpu: 16
    memory: 32Gi
```

Размер ресурсов задают **пресетом** (preset) в формате `<серия>.<размер>`. Серия закрепляет соотношение процессора к памяти: `t1` — 1:0.5, `c1` — 1:1, `s1` — 1:2, `u1` — 1:4, `m1` — 1:8. Размеры идут от `nano` до `4xlarge`, так что `u1.medium` — это универсальная серия среднего размера.

Если рядом с пресетом явно написан блок `resources`, побеждает он:

А теперь то, на чём спотыкаются практически все, — **переподписка** (overcommit, `cpuAllocationRatio`, по умолчанию **10**).

Работает она так: когда вы запускаете нагрузку с пределом в четыре процессора, платформа просит у планировщика не четыре, а **предел, делённый на коэффициент** — то есть четыре десятых. Гарантируется малое, а взять при необходимости можно много.

Формула короткая, и её стоит запомнить дословно:

```
запрос = предел ÷ cpuAllocationRatio
```

Смысл в том, что приложения почти никогда не потребляют свой предел. Раздавать гарантии по верхней планке значит держать половину железа простаивающей. Но если про коэффициент не знать, поведение планировщика кажется бессмысленным: попросили четыре ядра, а зарезервировано меньше половины одного.

Важно, где живёт этот коэффициент: он настраивается на уровне **всей платформы**, а не в отдельном тенанте. Один на всех — свойством тенанта его в вариантах ответа не называйте.

Изоляция

По умолчанию тенанты друг друга не видят: сетевые политики запрещают ходить между пространствами имён. И это не переключатель — **отключить изоляцию нельзя**. Флаг `isolated` существовал до версии 1.0 и убран; если он попадётся в вариантах ответа, это ловушка.

Отрезаны поды не только от соседей. По умолчанию они не достучатся ни до `kube-apiserver`, ни до собственного `etcd` тенанта. Нужен доступ — его открывают точечно, метками на поде:

Как люди попадают внутрь

Три пути, и это тоже спрашивают.

Дашборд — веб-интерфейс платформы. Вход через **Keycloak**, то есть по корпоративной учётной записи, а не по отдельному паролю.

kubeconfig — файл доступа, который выдаётся тенанту. Внутри него не сертификат, а вызов внешней программы: `kubectl` при первом обращении открывает браузер, вы входите через Keycloak, и выданный токен действует до истечения срока. Отсюда, кстати, требование поставить `kubelogin`: без него `kubectl` не знает, как логиниться.

Terraform — тот же API, только описанием.

Откуда берётся сам файл кубконфига

Готовым файлом кубконфиг тенанту не выдают — его собирает администратор платформы скриптом из документации. Знать этот скрипт наизусть не нужно, а вот **из чего он собирает файл** — спрашивают.

У каждого тенанта в его пространстве имён лежит одноимённый секрет: у тенанта `alpha` это секрет `alpha` в пространстве `tenant-alpha`. Скрипт достаёт оттуда три вещи:

Что берётся	Из какого поля	Зачем
токен доступа	`token`	им тенант предъявляет себя API-серверу
сертификат центра сертификации	`ca.crt`	по нему `kubectl` убеждается, что сервер настоящий

Отсюда важное следствие: **кубконфиг тенанта — это секрет ровно в том же смысле, что и пароль.** Кто получил файл, тот получил и права тенанта, до отзыва токена.

Права внутри тенанта раздаются обычными ролями Kubernetes. Никакой отдельной системы разрешений у платформы нет, и это ответ на популярный вопрос-ловушку.

Тенант заведён, квоты выданы, люди внутрь пущены. Теперь им надо что-то заказать.

Что спросят на экзамене

- Что тенант — объект платформы, а не просто пространство имён.
- Что имя пространства имён = `tenant-` плюс цепочка предков через дефис, без корня.
- Что дефис в имени тенанта запрещён.
- Что тенанты вкладываются друг в друга, а вложенный создаётся внутри родителя.
- Что выключенный сервис наследуется от ближайшего предка, у которого он включён, — не обязательно от родителя.
- Имена четырёх переключателей дословно: `etcd`, `monitoring`, `ingress`, `seaweedfs`.
- Что домен тенанта собирается как `<имя>.<домен родителя>`.
- Что `metadata.namespace` — пространство имён родителя, а `status.namespace` — своё.
- Что кубконфиг тенанта скрипт из документации собирает из **токена, CA-сертификата и адреса сервера**: первые два — из одноимённого секрета тенанта, адрес — из кубконфига администратора.
- Формат пресетов `<серия>.<размер>`: t1 (1:0.5), c1 (1:1), s1 (1:2), u1 (1:4), m1 (1:8); размеры от nano до 4xlarge.
- Что явно заданный блок `resources` перекрывает пресет.
- Что пресеты тенантов — не instance types KubeVirt (U, O, CX, M, RT): `u1.medium` есть и там, и там.
- Что запрос нагрузки = её предел, делённый на `cpuAllocationRatio` (по умолчанию 10).
- Что `cpuAllocationRatio` настраивается на уровне платформы, а не отдельного тенанта.
- Что изоляцию отключить нельзя, а флаг `isolated` убран в версии 1.0.
- Имена меток дословно: `policy.cozystack.io/allow-to-apiserver` и

3. Каталог управляемых сервисов

Что можно заказать у платформы, что происходит после нажатия кнопки и почему это всегда один и тот же механизм.

Каталог — то, ради чего платформу обычно и берут. Вместо «поставьте нам PostgreSQL» человек открывает список и заказывает базу, как заказывают виртуалку.

Что в списке

Каталог поделён на четыре группы, и на экзамене их называют по-английски: базы данных `Databases`, обмен сообщениями `Messaging`, платформенные сервисы `Platform services`, сетевые сервисы `Networking services`.

Группа	Позиции
Базы данных	PostgreSQL, MariaDB, MongoDB, ClickHouse, FoundationDB, Redis, Qdrant
Обмен сообщениями	Kafka, NATS, RabbitMQ
Платформенные сервисы	Harbor, OpenBao, Bucket, Tenant, Monitoring, Etcd, Ingress
Сетевые сервисы	VPN, балансировщик TCP, кеш HTTP, virtual-router, VPC

Список растёт от версии к версии, и заучивать его целиком незачем. Спрашивают другое — **что все они устроены одинаково**.

Один механизм на всё

Вот что стоит понять один раз и больше не возвращаться.

Один и тот же путь для базы, очереди, виртуалки и кластера.

Вы создаёте объект. Платформа заводит на него `HelmRelease`, Flux разворачивает чарт — а уже из чарта приезжает **оператор**, который дальше и ведёт вашу базу: следит за репликацией, снимает копии, переключает главную копию при отказе.

Своих движков баз платформа не пишет — берёт известные операторы. Их имена спрашивают:

Сервис	Оператор
PostgreSQL	CloudNativePG
Kafka	Strimzi

Отсюда два практических вывода, которые попадают в вопросы. Первый: **дашборд не делает ничего особенного** — нажатие кнопки собирает ровно такой же объект и отправляет его в API. Второй: если сервис не поднялся, смотреть надо на `HelmRelease`, а не гадать по подам.

Какие типы приложений вообще есть на конкретном кластере, показывает одна команда:

```
kubectl api-resources | grep apps.cozystack
```

Результат работы платформа пишет обратно в сам объект — в `status.conditions`. У живого приложения там стоит `Ready: True`; если не стоит — дальше по цепочке, к `HelmRelease`.

Три пути к одному и тому же

Дашборд — форма с полями. Поля берутся из описания приложения, поэтому список параметров всегда соответствует версии платформы.

kubectl — тот же объект текстом. Этот путь важен не потому, что удобнее, а потому, что описание можно отревьюить, положить в Git и откатить. Кнопку откатить нельзя.

Terraform — для тех, у кого инфраструктура уже описана им. Официальный провайдер зовут `cozystack/cozystack`; имя стоит запомнить, его спрашивают.

И общее для всех трёх путей предупреждение: `kubectl delete` на объекте приложения сносит приложение целиком — вместе с подами и данными. Обращайтесь с этой командой так же, как с удалением базы.

Что задаётся при заказе

Пользователи и базы — платформа заводит их сама. Именно поэтому в файле схемы, который вы потом накатываете, нет команд создания базы и пользователя: они уже выполнены.

Пароли платформа генерирует сама и кладёт в секрет. В дашборде он виден на вкладке `Secrets` у самого приложения.

Отказоустойчивость не бесплатна

Заказывая сервис, вы выбираете число копий. Одна копия — учебный стенд: узел ушёл, сервис ушёл с ним. Две и больше — платформа сама следит, кто главный, и переключает при отказе.

Здесь же живёт частая ловушка: **отказоустойчивость сервиса и сохранность данных — разные вещи**. Копии спасают от падения узла, но не от удалённой таблицы. Для второго нужны резервные копии, и о них отдельный урок.

Что принесла версия 1.5

Три факта из этого релиза спрашивают прямо. Шифрование клиентских подключений (TLS) появилось у четырёх сервисов: Kafka, NATS, Qdrant и PostgreSQL. Резервные копии управляемых приложений заработали из коробки — раньше администратору приходилось сначала настраивать хранилище под них. И появились `ApplicationDefinition`: механизм регистрирует в каталоге новый тип приложения поверх Helm-чарта, и у него сразу есть свой объект API, форма в дашборде и та же цепочка с `HelmRelease`. Так организации добавляют в общий каталог собственные сервисы — каталог не закрытый список.

Каталог мы прошли, но одну его позицию — виртуальные машины — стоит разобрать отдельно: у неё своя модель, и привычки из vSphere здесь подводят.

Что спросят на экзамене

- Четыре группы каталога: `Databases`, `Messaging`, `Platform services`, `Networking services` — и кто в какой.
- Что `Tenant` — сам позиция каталога, и вложенные тенанты создают через неё.
- Операторов по именам: CloudNativePG — PostgreSQL, Strimzi — Kafka, Percona — MongoDB, Altinity — ClickHouse.
- Что заказ любого сервиса — это объект в API, а дашборд лишь собирает его за вас.
- Что список типов показывает `kubectl api-resources | grep apps.cozystack`.
- Что готовность читают в `status.conditions` — там должно быть `Ready: True`.
- Что `kubectl delete` на объекте удаляет приложение вместе с данными.
- Имя провайдера Terraform — `cozystack/cozystack`.
- Что нового в 1.5: TLS для Kafka, NATS, Qdrant и PostgreSQL; резервные копии из коробки; `ApplicationDefinition` для расширения каталога.
- Цепочку: объект → HelmRelease → чарт → оператор → работающие поды.
- Что диагностику начинают с состояния HelmRelease.
- Что пользователей и базы заводит платформа, а пароли кладёт в секрет.
- Разницу между `replicated` и `local`.
- Что число копий защищает от отказа узла, но не заменяет резервные копии.

> Подробнее: управляемые приложения · кластеры Kubernetes

4. Виртуальные машины

Как виртуалка становится обычной нагрузкой Kubernetes, зачем диск отдельно от машины и где проходят честные границы.

Тема, на которой администратору VMware проще всего — и на которой он же чаще всего попадается, потому что привычные аналогии тут работают лишь наполовину.

Машина как обычная нагрузка

Виртуализацию обеспечивает **KubeVirt**. Идея у него такая: виртуальная машина — это процесс, который запускается в поде, а управляется как всё остальное в кластере.

Из этого следует то, что для человека из мира vSphere звучит непривычно. Планировщик Kubernetes выбирает узел для машины так же, как для приложения. Ресурсы считаются теми же запросами и пределами. Права выдаются тем же RBAC.

Гипервизор при этом честный — KVM, тот же, что в любом Linux.

Два объекта вместо одного

Здесь главное отличие от привычной модели, и его спрашивают.

Диск создаётся отдельно и живёт своей жизнью. Содержимое он берёт одним из четырёх способов, и их стоит различать: загрузка по ссылке (`http`), клон подготовленного образа платформы (`image`), выгрузка файла со своей машины (`upload`) и пустой источник `{}` — чистый диск без содержимого, когда нужно просто место под данные.

Третий объект модели — **`VMImage`**, подготовленный образ, который платформа хранит у себя. Команда публикует условную Ubuntu один раз, а новые диски клонируют её через `source.image.name` вместо того, чтобы каждый раз тянуть файл из интернета.

Ниже всё это — обычное хранилище Kubernetes: диск существует как `PersistentVolumeClaim`, а наливает в него образ компонент **CDI** (`Containerized Data Importer`) через объект `DataVolume`. Руками их здесь не трогают, но на нижних слоях встретите именно их.

Размер машины задаётся не числами, а именованным **типом инстанса** (`instance type`) вроде `u1.medium` — это механизм самого KubeVirt, серии U, O, CX, M и RT. Не путайте его с пресетами тенанта и управляемых приложений: строка `u1.medium` валидна и там, и там, но системы разные. Смотрите, что размеряет вопрос: тенант и приложение берут пресет, `VMInstance` — тип инстанса.

Как попасть внутрь

Здесь заканчиваются привычки и начинается специфика.

cloud-init — стандартный механизм облачных образов: при первом запуске система читает описание и настраивает себя. Пароль, ключи, пакеты, команды. Это ближайший аналог Customization Specification, только описанный текстом рядом с самой машиной.

virtctl — отдельная утилита для консоли, экрана и проброса портов. Путей внутрь

ранней загрузки.

Отдельная оговорка про ограниченный доступ. Тенанту консоль выдают как подресурс **запущенного экземпляра**, а не объекта машины целиком, и под такими правами цель называют с префиксом типа: `virtctl console --namespace=tenant-acme vmi/vm-instance-app`. Голое имя в этом случае попадает не туда и возвращает отказ.

Что умеет и чего не умеет

Экзамен любит вопросы про возможности — и здесь важно не перепутать, чего платформа не делает, с тем, что она делает иначе.

Живая миграция есть. Работающая машина переезжает на другой узел без остановки — этим пользуются, когда узел нужно вывести на обслуживание.

Снимки состояния есть. Их обеспечивает слой хранилища.

Проброс видеокарт есть, причём настраивается автоматически. Список устройств, которые разрешено пробрасывать, задаёт платформа, а не тенант.

Здесь стоит запомнить и то, **что именно изменилось в версии 1.5**, — на этом строят отдельный вопрос. Сам проброс существовал и раньше, но подготовку узлов приходилось делать руками: размечать узлы с видеокартами и вписывать разрешённые устройства прямо в настройки KubeVirt командой `kubectl patch`. В 1.5 автоматизировали ровно это. Группа узлов, в которой объявлены видеокарты, теперь сама получает нужную метку, а платформа сама заполняет список разрешённых устройств. Формулировка «проброс появился только в 1.5» неверна — появилась **автоматическая подготовка узлов**, а не сама возможность.

Чего действительно нет — **автоматической балансировки нагрузки между узлами**. В vSphere этим занимается DRS: он сам решает, что машину пора

Если узел умрёт

Вопрос, который задаёт любая команда перед переездом с vSphere: перезапустятся ли машины на другом узле сами. Честный ответ для версии 1.5 — **нет**, и он же правильный на экзамене.

Причина не в недоделке, а в устройстве. Когда узел перестаёт отвечать, кластер не знает, что с ним: он выключился, или потерял сеть и продолжает работать. Для обычного приложения разницы нет — лишняя копия никому не мешает. Для виртуальной машины с диском разница принципиальная: запустить вторую копию той же машины, пока первая, возможно, ещё пишет на диск, — это порча данных. Поэтому безопасный автоперезапуск требует **фенсинга** (fencing) — гарантированного отключения подозрительного узла, прежде чем машину поднимут заново. В поставке такого механизма нет, и в восстановлении после отказа узла остаются ручные шаги.

Живая миграция здесь не помогает и помочь не может: она переносит **работающую** машину с **живого** узла. Это плановая эвакуация перед обслуживанием, а не реакция на аварию.

Отдельно — чтобы не смешивать две разные вещи. У **тенантных кластеров Kubernetes** рабочие узлы это тоже виртуальные машины, но там проверка здоровья узлов и замена сломанных работают и включены по умолчанию. Только это не «перезапуск той же машины», а **замена**: неисправную машину удаляют и создают новую, чистую, взамен. Для узла тенантного кластера это нормально — его состояние в кластере не хранится. Для вашей собственной виртуалки с данными на диске — неприемлемо, и именно поэтому к ней тот же механизм не применяют.

Машина работает. Пакеты до неё довозит сеть — а там четыре компонента, которые легко перепутать между собой.

Что спросят на экзамене

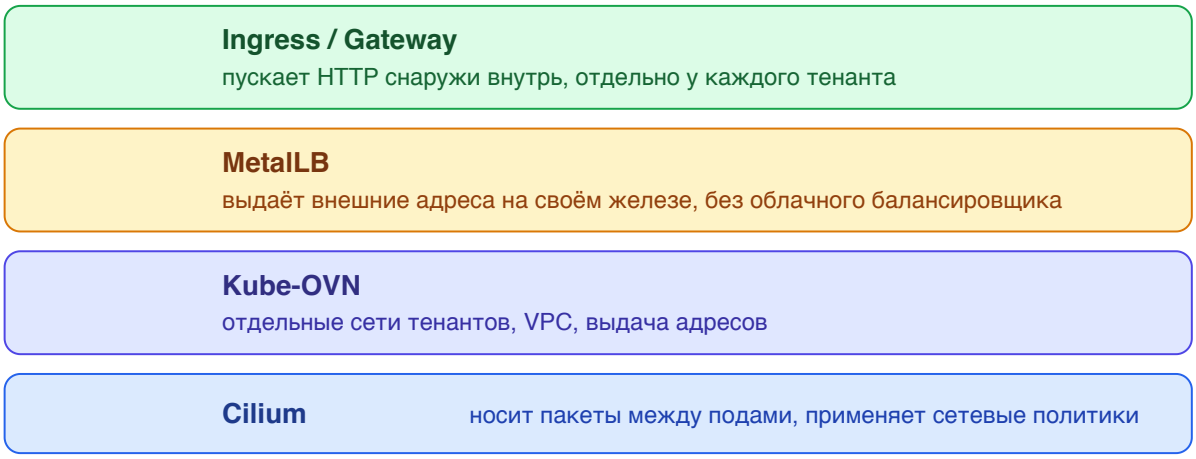
- Что виртуализацию обеспечивает KubeVirt, а машина — обычная нагрузка кластера.
- Что диск и машина — два отдельных объекта, и диск переживает машину.
- Четыре источника содержимого диска: `http`, `image`, `upload` и пустой `{}` — чистый диск.
- Что `VMImage` — третий объект модели: подготовленный образ, из которого клонируют диски.
- Что диск существует как PVC, а образ в него наливает CDI через `DataVolume`.
- Что размер машины задаёт instance type KubeVirt (серии U/O/CX/M/RT), а не пресет тенанта.
- Что первичная настройка идёт через cloud-init.
- Три пути внутрь: `virtctl console`, `virtctl vnc`, `virtctl ssh` — и что при сломанной сети в машине сработает консоль.
- Что префикс `vmi/` нужен под ограниченными правами тенанта — из-за доступа к подресурсу запущенного экземпляра.
- Что живая миграция, снимки состояния и проброс видеокарт есть.
- Что автоматической балансировки между узлами, как у DRS, нет.
- Что в 1.5 автоматизировали **подготовку узлов** под проброс видеокарт — разметку узлов и список разрешённых устройств, — а не сам проброс.
- Что автоперезапуска виртуальной машины на другом узле после отказа **нет**: для этого нужен фенсинг, которого в поставке нет, и в восстановлении остаются ручные шаги.

> Подробнее: виртуализация · хранилище

5. Сети

Четыре компонента, каждый со своей задачей: кто носит пакеты внутри, кто делит сети между тенантами, кто выдаёт внешние адреса и кто пускает HTTP.

Сетевая часть выглядит сложной, пока не разложить её по задачам. Компонентов четыре, и у каждого своя работа — экзамен спрашивает именно распределение ролей, а не настройки.



Снизу вверх: от пакетов между подами до входа снаружи.

Cilium — основа

Отвечает за то, чтобы поды видели друг друга, и за сетевые политики. Работает на **eBPF** — технологии, которая позволяет исполнять программы прямо в ядре Linux, не переключаясь в пространство пользователя на каждый пакет.

Есть и третья роль, которую легко пропустить: Cilium **заменяет собой kube-proxy** (`kube-proxy replacement`). Обычно сервисы Kubernetes обслуживает kube-proxy через iptables, а здесь поиск нужного адресата идёт по хеш-таблице и стоит одинаково при любом числе сервисов. На платформе с сотнями тенантов это заметно.

Разделение труда между ним и Kube-OVN стоит запомнить, потому что оба называются «сетью»: **Cilium — основной CNI**, он даёт сеть подам и применяет политики для всех. **Kube-OVN работает поверх** и отвечает за наложенную сеть, выдачу адресов и VPC тенантов.

Kube-OVN — сети тенантов

Kube-OVN строит поверх физической сети узлов **наложенную сеть** (`overlay`) и добавляет к ней два свойства, которые экзамен спрашивает чаще прочего.

Централизованная выдача адресов (`centralized IPAM`): адреса раздаются из одного места, а не отдельно на каждом узле, и весь кластер по умолчанию живёт в одном общем диапазоне подов.

Стабильные адреса подов (`stable pod IPs`): под сохраняет свой адрес, когда переезжает на другой узел. Контейнеру это удобство, а виртуальной машине — необходимость: гостевые системы, лицензии и правила межсетевых экранов обычно привязаны к адресу, и смена адреса при каждой миграции сломала бы их.

Ещё Kube-OVN даёт **VPC** — приватные сети со своим адресным пространством. И VPC, и виртуальный маршрутизатор (`virtual-router`) — позиции каталога: тенант заказывает их так же, как базу данных.

Ближайшая аналогия из привычного мира — VPC у облачных провайдеров или сети NSX.

MetalLB держит пул адресов и раздаёт их сервисам, а затем объявляет их в сеть — либо по ARP, либо через BGP, если сеть построена на маршрутизации. Начиная с версии 1.5 за сторону BGP отвечает **FRR-K8s** (FRR–K8s) — именно он объявляет маршруты физическим роутерам.

Именно он нужен, чтобы виртуалка или приложение получили адрес, доступный снаружи кластера.

Ingress — вход для HTTP

Для веб-приложений выдавать каждому свой адрес расточительно. **Ingress** принимает запросы на общий адрес и раскладывает их по именам и путям.

Особенность платформы: **точка входа (ingress) своя у каждого тенанта**. Это не общий контроллер на весь кластер — тенант с включённым `ingress` получает собственный `ingress-nginx`, со своими правилами, своим внешним адресом от MetalLB и своими сертификатами. Тенант без него пользуется родительским, как и с остальными сервисами.

TenantGateway — путь Gateway API

Kubernetes постепенно уходит со старого Ingress на **Gateway API** — более выразительный стандарт маршрутизации трафика. В платформе он появился в версии 1.5 в виде объекта `TenantGateway`.

Пакеты через него носит **Cilium**: nginx в этом пути нет вовсе. И это не замена `ingress-nginx`, а альтернатива — в 1.5 живут оба варианта.

Сертификаты `TenantGateway` умеет выпускать в двух режимах проверки. **HTTP-01** — центр сертификации забирает токен по обычному HTTP, значит домен должен быть виден из интернета. **DNS-01** — проверка через DNS-запись; только этот

CoreDNS разрешает имена внутри кластера — поэтому в настройках приложений пишут `postgres-db-rw`, а не адрес: имя переживает переезд базы на другой узел, а адрес нет.

ExternalDNS смотрит за опубликованными сервисами и точками входа и сам заводит для них записи у вашего DNS-провайдера. Приложение опубликовали — имя начало разрешаться, заявку в сетевой отдел писать не нужно.

cert-manager выпускает сертификаты и продлевает их сам, без напоминаний.

Всё это работает ровно до первого сбоя. О том, как о сбое узнать и что стоит подготовить заранее, — следующий урок.

Что спросят на экзамене

- Что Cilium носит пакеты между подами и работает на eBPF.
- Что Cilium заменяет собой kube-proxy.
- Что Kube-OVN даёт отдельные сети тенантов и VPC.
- Что у Kube-OVN централизованная выдача адресов и один общий диапазон подов на кластер.
- Что под сохраняет адрес при переезде на другой узел — и почему это критично для BM.
- Что VPC и виртуальный маршрутизатор (`virtual-router`) — позиции каталога.
- Что MetalLB выдаёт внешние адреса на своём железе, объявляя их по ARP или BGP.
- Что с версии 1.5 за BGP в MetalLB отвечает FRR-K8s.
- Что точка входа для HTTP — `ingress-nginx` — заводится у каждого тенанта отдельно.
- Что `TenantGateway` появился в 1.5, это Gateway API, и пакеты для него носит Cilium.
- Два режима выпуска сертификатов: HTTP-01 и DNS-01; wildcard — только DNS-01.
- Что записи в DNS для опубликованных приложений заводит ExternalDNS.
- Что имена внутри кластера разрешает CoreDNS, а сертификаты выпускает cert-manager.
- Почему в настройках приложений пишут имена сервисов, а не адреса.

> Подробнее: сети платформы

6. Наблюдаемость и резервные копии

Кто собирает метрики и журналы, где их смотреть и чем резервная копия отличается от отказоустойчивости.

Две темы в одном уроке, потому что обе про одно: что делать, когда что-то пошло не так, — и что вы успели подготовить заранее.

Метрики и журналы

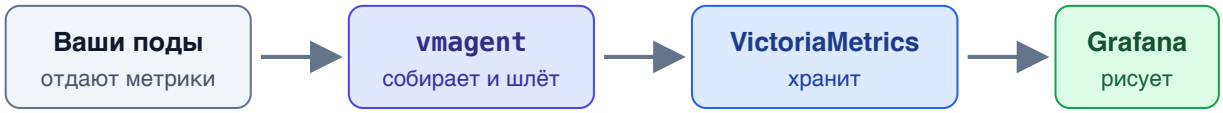
Стек здесь не тот, который ожидают по привычке, и это любимая ловушка экзамена.

Что собирают	Чем
Метрики	**VictoriaMetrics**
Журналы	**VictoriaLogs**
Графики	**Grafana**
Оповещения	**Alerta**

Ни Prometheus, ни Loki, ни Elasticsearch в платформе нет. VictoriaMetrics при этом понимает язык запросов PromQL — то есть привычные запросы работают, а хранилище другое: компактное и дешевле на больших объёмах.

Цепочка оповещений короткая, и порядок в ней спрашивают: **VMAlert** проверяет правила по метрикам в VictoriaMetrics → **Alerta** сводит срабатывания в одно место, убирает дубликаты и маршрутизирует их → почта, SMS, мессенджеры. Alerta нужна затем, чтобы одна авария не приходила восемью письмами из восьми источников.

Grafana приезжает с готовыми дашбордами, и экзамен спрашивает их по названиям групп: **Cluster Overview** — здоровье кластера целиком, **Node Metrics** — процессор, память, диск и сеть по узлам, **ETCD** — состояние хранилища etcd, **Storage** — здоровье LINSTOR и SeaweedFS, **Tenant Applications** — нагрузки и управляемые приложения по тенантам. Есть ещё дашборды по трафику точки входа и по управляемым базам данных.



Тот же путь у журналов, только вместо vmagent — fluent-bit, а вместо VictoriaMetrics — VictoriaLogs.

Про наследование помните из второго урока: тенант без своего мониторинга шлёт метрики родительскому. А вот включать сбор **задним числом бесполезно** — записей за прошлое взять неоткуда, и это стоит решать при создании кластера, а не когда график понадобился.

Резервные копии

Объект	Что делает
`BackupClass`	куда и как складывать. Заводит администратор платформы, действует на весь кластер
`Plan`	расписание: снимать по такому-то графику
`BackupJob`	разовый запуск, здесь и сейчас
`Backup`	результат — сама снятая копия

Ключевая пара — `Plan` и `BackupJob`. Нужна копия **каждую ночь** — это `Plan`. Нужна копия **прямо сейчас, перед обновлением** — это `BackupJob`. Восстановление — отдельный объект, `RestoreJob`.

Начиная с версии 1.5 в платформе есть готовый `cozy-default` — класс копий, который работает сразу, без настройки хранилища.

Velero. Уровнем выше: копирует объекты самого кластера и тома. Это про восстановление платформы, а не отдельной базы. С версии 1.5 он не опция, а штатный компонент — ставится по умолчанию.

Настраивают его два объекта: `BackupStorageLocation` — куда складывать копии, и `VolumeSnapshotLocation` — где хранить снимки томов. Кластеру Kubernetes внутри тенанта Velero включают одним полем: `spec.addons.velero.enabled`.

Где проходит граница

Самое ценное в этой теме — понимать, чего резервные копии **не** делают.

Копия управляемого приложения берёт **только данные**. В неё не попадают ни

Инкрементальных копий виртуальных машин нет: отслеживания изменённых блоков в платформе не делают, каждая копия полная. Если за спиной привычка к цепочкам инкрементов, окна копирования и объём хранилища придётся пересчитать.

Копии — не отказоустойчивость. Реплики спасают от падения узла, копии — от удалённых данных. Это разные беды, и одно другое не заменяет.

Копия, которую ни разу не разворачивали, — не копия, а надежда. Восстановление надо пробовать, пока оно не понадобилось.

И главное: копия хранится в объектном хранилище. Если оно живёт в том же кластере, что и данные, то от потери кластера целиком она не спасёт. Для настоящей защиты хранилище должно быть снаружи.

Мы разобрали, из чего платформа сделана и что она умеет. Остался последний вопрос — почему она устроена именно так.

Что спросят на экзамене

- Что метрики хранит VictoriaMetrics, а журналы — VictoriaLogs. Не Prometheus и не Loki.
- Что VictoriaMetrics совместима с PromQL.
- Что метрики собирает vmagent, а хранит их VMCluster.
- Порядок оповещений: VMAlert → Alerta → почта, SMS, мессенджеры.
- Группы стандартных дашбордов: Cluster Overview, Node Metrics, ETCD, Storage, Tenant Applications.
- Четыре объекта: `BackupClass`, `Plan` (расписание), `BackupJob` (разовый запуск), `Backup` (результат).
- Что копия по расписанию — это `Plan`, а не `BackupJob`.
- Что `cozy-default` работает из коробки с версии 1.5.
- Что Velero работает на уровне платформы, а не отдельного сервиса.
- Что Velero стал штатным компонентом с версии 1.5, а в тенантном кластере включается полем `spec.addons.velero.enabled`.
- Что `BackupStorageLocation` задаёт, куда складывать копии, а `VolumeSnapshotLocation` — где хранить снимки томов.
- Чего копия «только данные» не берёт: HelmRelease, объект CR, секреты оператора.
- Что приложение-приёмник должно существовать до восстановления.
- Что инкрементальных копий виртуальных машин нет — каждая копия полная.
- Что реплики и резервные копии решают разные задачи.

> Подробнее: мониторинг · сервисы кластера

7. Как это вообще работает

Желаемое состояние, операторы и GitOps — четыре идеи, на которых держится всё остальное.

Последний урок — про идеи, а не про компоненты. Они лежат в основании всего, о чём шла речь раньше, и вопросы по ним формулируются просто: «почему это работает именно так».

Вы описываете результат, а не действия

Привычный подход к администрированию — последовательность шагов: поставь, настрой, запусти, проверь. Здесь иначе: вы описываете, **как должно быть** — желаемое состояние, desired state, а система сама решает, что для этого сделать.

Разница проявляется, когда что-то ломается. Скрипт, отработавший вчера, сегодня не поможет: он уже выполнен. А описание желаемого состояния действует постоянно — если реальность от него отклонилась, её вернут обратно.

Из чего состоит манифест

Описание желаемого состояния живёт в файле YAML, который называют **манифестом**. Какой бы объект вы ни описывали — тенант, базу, виртуальную машину, — верхнеуровневых полей всегда четыре, и путать их на экзамене не стоит.

Поле	Что в нём
`apiVersion`	к какой версии API относится тип, например `apps.cozystack.io/v1alpha1`
`kind`	сам тип: `Tenant`, `Bucket`, `VMInstance`
`metadata`	паспортные данные: имя, пространство имён, метки, аннотации
`spec`	**желаемое состояние** — всё, чем объект должен стать

Пятое поле, `status`, вы не пишете никогда: его заполняет контроллер, и в нём написано, как дела обстоят на самом деле. Отсюда простое правило чтения любого объекта: **spec — это ваше требование, `status` — ответ платформы на него.** Цикл сверки выше — это и есть непрерывное сведение одного к другому.

```yaml
apiVersion: apps.cozystack.io/v1alpha1
kind: Bucket
metadata:
  name: images           # как объект зовут
  namespace: tenant-lab  # где он живёт
spec:                    # каким он должен стать
  replicas: 2
```

Определение своего типа — CRD (custom resource definition). Вы регистрируете новый тип, и дальше его объекты хранит и отдаёт обычный API-сервер вместе со встроенными. Так в платформе устроены, например, резервные копии (`backups.cozystack.io`) и шлюзы (`gateway.cozystack.io`).

Агрегированный API-сервер. Отдельная программа берёт на себя целую группу API, а основной API-сервер переадресует ей запросы. В Cozystack это `cozystack-api` в пространстве имён `cozy-system`, и именно он отвечает за самые заметные группы — `apps.cozystack.io` (тенанты, бакеты, базы, кластеры), `core.cozystack.io`, `sdn.cozystack.io`. CRD с именем `tenants.apps.cozystack.io` в кластере вы не найдёте: этот тип не зарегистрирован, он **отдаётся** агрегированным сервером.

Кто за какую группу отвечает, видно одной командой — в колонке `SERVICE` либо адрес программы, либо слово `Local`, означающее «обычный API-сервер, тип из CRD»:

```
kubectl get apiservices | grep cozystack
```

Но тип сам по себе ничего не делает: это запись, которую кластер согласен хранить. Работу выполняет **оператор** (operator) — программа, которая следит за объектами своего типа и приводит мир в соответствие.

Отсюда практический вывод: если объект создан, а ничего не произошло, вопрос не к объекту, а к оператору. Он либо не запущен, либо не смог.

GitOps

Раз состояние описывается текстом, текст можно хранить в системе контроля версий. Тогда Git становится единственным источником истины, а специальная программа следит, чтобы кластер ему соответствовал.

Платформа использует **FluxCD** и применяет этот подход **к самой себе**: её

У FluxCD два объекта, которые нужно знать по именам. `HelmRepository` — источник, откуда берутся чарты. `HelmRelease` (короткое имя `hr`) — заявление «этот чарт такой-то версии с такими значениями должен быть установлен и оставаться установленным».

Как временно отключить сверку

Непрерывная сверка мешает ровно в одном случае: когда нужно что-то починить руками. Правку, сделанную в обход описания, контроллер откатит через несколько секунд — он для этого и существует.

На такой случай у `HelmRelease` есть выключатель `spec.suspend`. Переведённый в `suspend` релиз контроллер **перестаёт сверять**: он не переустанавливает чарт, не откатывает изменения и вообще не трогает объект. При этом **нагрузка продолжает работать** — поды не удаляются, приложение отвечает, — а ваши ручные правки сохраняются до тех пор, пока сверку не включат обратно.

```
kubectl patch hr <имя> -n <пространство> --type merge -p '{"spec":{"susp
```

Обратная операция — `"suspend": false`. В момент возврата контроллер сверяет объект заново и всё, что вы поправили руками мимо описания, откатывается. Поэтому `suspend` — инструмент диагностики на время расследования, а не способ жить с ручными правками.

Helm

Один объект — это один объект. Приложение — это обычно десяток: развёртывание, сервис, настройки, секреты, правила доступа.

Helm упаковывает такой набор в **чарт** — шаблоны плюс значения. Меняя значения, вы получаете разные установки из одной упаковки.

В платформе Helm — не рекомендация, а несущая конструкция: каждая позиция

Всё, что делает платформа, проходит по ней. Заказали базу — прошло по ней. Создали тенант — прошло по ней. Установилась сама платформа — тоже.

Словарь Kubernetes, который спросят здесь же

Этот раздел экзамена описан как «седьмая тема плюс базовая терминология», поэтому шесть слов стоит проговорить. Английские названия — ровно те, что будут в вопросах.

Объект	Что это и зачем
`Namespace`	именованное пространство, группирует ресурсы; у каждого тенанта своё
`Pod`	наименьшая единица нагрузки — один контейнер или несколько вместе
`Deployment`	описывает желаемое: какой образ, сколько реплик, как обновлять
`ReplicaSet`	его исполнитель: держит нужное число подов; руками его почти не трогают
`Service`	стабильное имя и адрес перед меняющимся набором подов
`PVC`	заявка на постоянное хранилище, которое переживает перезапуск пода
`Secret`	хранит чувствительное: пароли, токены, ключи

У `Service` три типа: **ClusterIP** — адрес виден только внутри кластера, это тип по

- `kubectl api-resources` — какие типы ресурсов кластер вообще знает, включая добавленные платформой
- `kubectl apply -f manifest.yaml --dry-run=server` — отдать манифест на проверку серверу, ничего не сохраняя
- `helm upgrade --install` — идемпотентно: поставит, если релиза нет, обновит, если есть

Что спросят на экзамене

- Что описывается желаемое состояние, а не последовательность действий.
- Что контроллер непрерывно сверяет желаемое с фактическим и устраняет разницу.
- Что желаемое состояние объекта лежит в `spec`, фактическое — в `status`, а `apiVersion`, `kind` и `metadata` отвечают за версию API, тип и имя.
- Что CRD добавляет тип, а работу выполняет оператор.
- Что `kubectl get tenants` работает потому, что Cozystack расширяет API Kubernetes: тип `Tenant` отдаёт агрегированный сервер `cozystack-api`, а не CRD.
- Что `suspend` у HelmRelease останавливает сверку, но не удаляет нагрузку: поды продолжают работать, ручные правки сохраняются до возврата сверки.
- Что платформа применяет GitOps к самой себе через FluxCD.
- Что желаемое состояние платформы задаёт Platform Package, а чарты FluxCD берёт из OCI-регистри.
- Два объекта FluxCD: `HelmRepository` — источник чартов, `HelmRelease` — заявление об установленном чарте.
- Что чарт — единица упаковки, и каждая позиция каталога это чарт.
- Что сам Cozystack ставится Helm-чартом `cozy-installer`.

> Подробнее: обзор платформы · концепции FluxCD

8. Шпаргалка

Одна страница на весь экзамен: кто за что отвечает, чем его подменяют в вариантах ответа и как готовиться по дням.

Последняя страница перед экзаменом. Читать её вместо уроков бесполезно — она не объясняет, а напоминает. А вот пробежать глазами за полчаса до попытки — самое то.

Кто за что отвечает

Третья колонка важнее первых двух: экзамен строит неверные варианты ответа на подменах, и почти все они — из этого списка.

Задача	Компонент	Чем подменяют в вариантах
Операционная система узлов	Talos Linux	Ubuntu, CoreOS
Виртуальные машины	KubeVirt	Proxmox, oVirt
Control plane тенантских кластеров	Kamaji	Cluster API сам по себе
Дисковое хранилище	LINSTOR / DRBD	Ceph, Longhorn
Объектное хранилище	SeaweedFS	MinIO, Ceph RGW
Сеть подов, политики	Cilium	Calico, Flannel
Сети тенантов, VPC	Kube-OVN	Cilium, Calico
Внешние адреса	MetalLB	облачный балансировщик
Метрики	VictoriaMetrics	**Prometheus**
Журналы	VictoriaLogs	**Loki**, Elasticsearch
Графики	Grafana	Kibana
Оповещения	VMAlert → Alerta	Alertmanager
Доставка конфигурации	FluxCD	ArgoCD
Вход	Keycloak	Dex

Числа, которые спрашивают

Что	Значение
Минимум узлов	3
На узел	8 ядер, 24 ГБ памяти
Диски на узел	50 ГБ + 256 ГБ
Задержка между узлами	меньше 10 мс
Сеть	один сегмент L2
`cpuAllocationRatio`	10 по умолчанию
Длина имени пространства имён	до 63 символов

Имена, которые надо знать дословно

Варианты установки: `isp-full`, `isp-full-generic`, `isp-hosted`, `default`. Старые имена `paas-full` и `distro-full` заменены в версии 1.5 — в вариантах ответа это ловушка.

Переключатели тенанта: `etcd`, `monitoring`, `ingress`, `seaweedfs`.

Метки доступа: `policy.cozystack.io/allow-to-apiserver`, `policy.cozystack.io/allow-to-etcd`. Значение — строка `"true"`.

Объекты резервных копий: `BackupClass`, `Plan`, `BackupJob`, `Backup`, `RestoreJob`. Готовый класс — `cozy-default`.

Кубконфиг тенанта собирается из **токена**, **CA-сертификата** и **адреса сервера**: первые два — из одноимённого секрета тенанта, адрес — из кубконфига администратора.

`suspend` у HelmRelease — контроллер перестаёт сверять; поды продолжают работать, ручные правки сохраняются до возврата сверки.

Цепочки

```
Слои: Talos → Kubernetes → Cozystack
Заказ: объект → HelmRelease → чарт → оператор → поды
Метрика: под → vmagent → VictoriaMetrics → Grafana
Журнал: под → fluent-bit → VictoriaLogs → Grafana
Оповещение: VMAlert → Alerta → почта и мессенджеры
Имя namespace: tenant- + цепочка предков без корня
```

Что есть, а чего нет

Есть: живая миграция машин, снимки состояния, проброс видеокарт, резервные копии по расписанию, отдельные сети тенантов.

Нет: автоматической балансировки между узлами, как у DRS. Инкрементальных копий виртуальных машин. Возможности отключить изоляцию тенантов. **Автоперезапуска виртуальной машины на другом узле после отказа** — для этого нужен фенсинг, которого в поставке нет.

Изменилось в 1.5: автоматизирована **подготовка узлов** под проброс видеокарт — разметка узлов и список разрешённых устройств. Сам проброс был и раньше.

План подготовки на пять дней

День	Что делать
1	Урок 7 — он про идеи, на которых держится всё. Потом урок 1
2	Урок 2 — самый плотный. Разобрать пропущенное из первого дня
3	Уроки 3 и 4
4	Уроки 5 и 6
5	Пробные вопросы несколько раз, добор по слабым темам, эта шпаргалка

Если пяти дней нет — читайте подряд за вечер, прогоняйте пробные вопросы до тех пор, пока не перестанете ошибаться, и идите сдавать.

Как устроен экзамен

60 вопросов, 90 минут. На английском — если он вам не родной, попросите дополнительные 30 минут заранее. Часть вопросов с несколькими верными ответами, и они засчитываются только целиком. Результат — сдал или нет, с общим баллом и оценкой по темам. Две попытки, между ними неделя.

Проходной балл не публикуется. Готовьтесь знать материал, а не попадать в число.