WILDBERRIES

Как сделать приемку, внедрение и харденинг Open Source-решения

Алексей Федулаев
DevSecOps Team Lead

$whoami

Алексей Федулаев

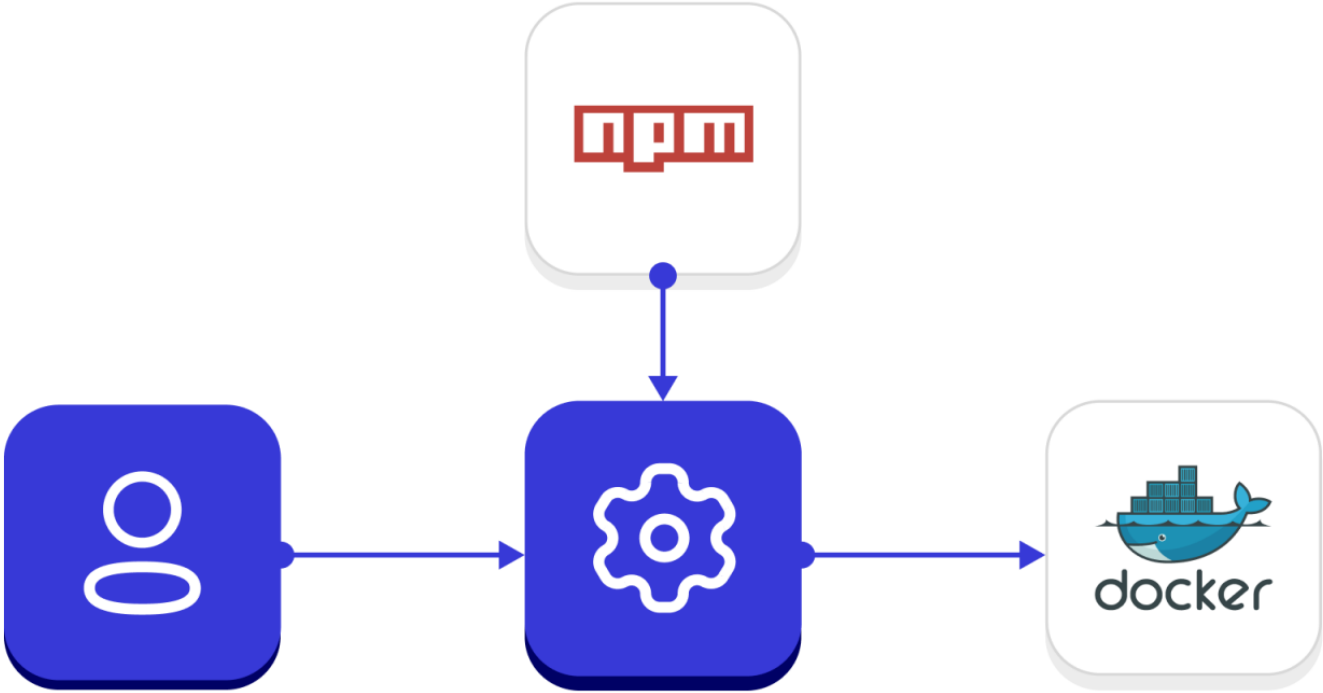
- 12+ лет в ИБ
- DevSecOps Team Lead Wildberries, Специалист по безопасности контейнерных перевозок. Спикер крупнейших российских конференций, ведущий подскастов SafeCode Live
- Tg: @int0x80h

Почему эта тема важна?

- Open Source стал неотъемлемой частью нашей жизни,
  а иногда и безальтернативным вариантом
- В то же время это привлекло большее внимание злоумышленников
- Начались постоянные атаки, в т.ч. на авторов
  Open Source-пакетов
- Как не оступиться, используя Open Source?

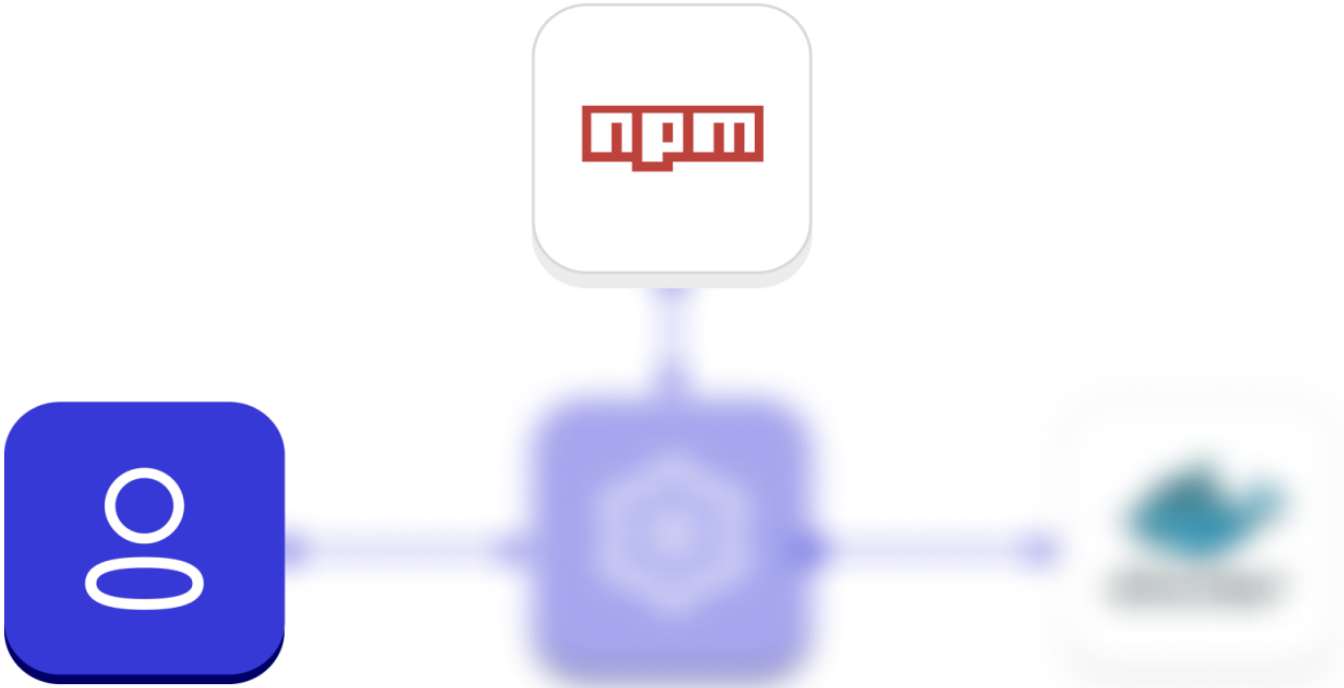
В рамках этого краш-курса мы разберемся, где стоит подстраховаться

Как работали раньше?

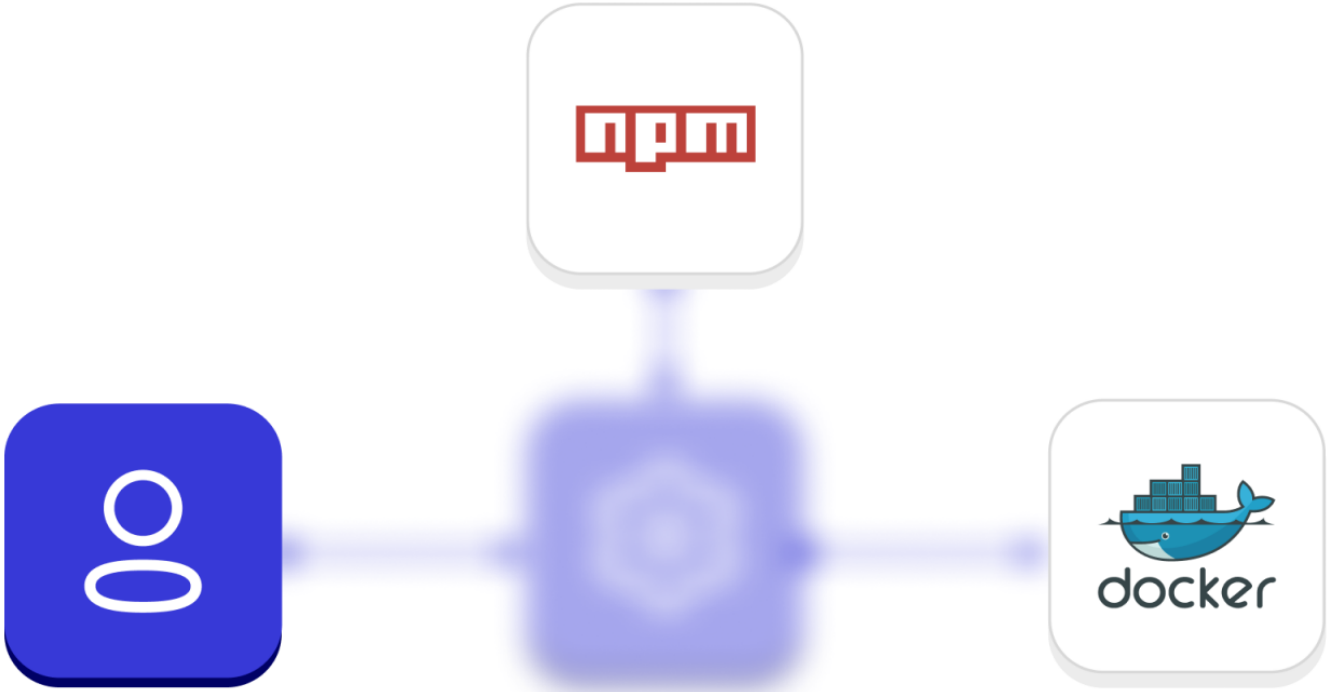


Как работали раньше?

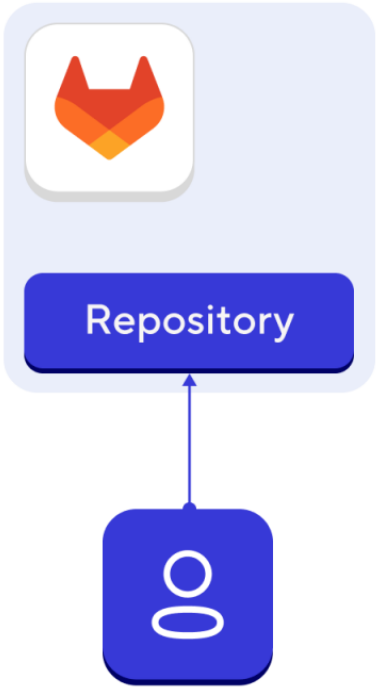
Как работали раньше?



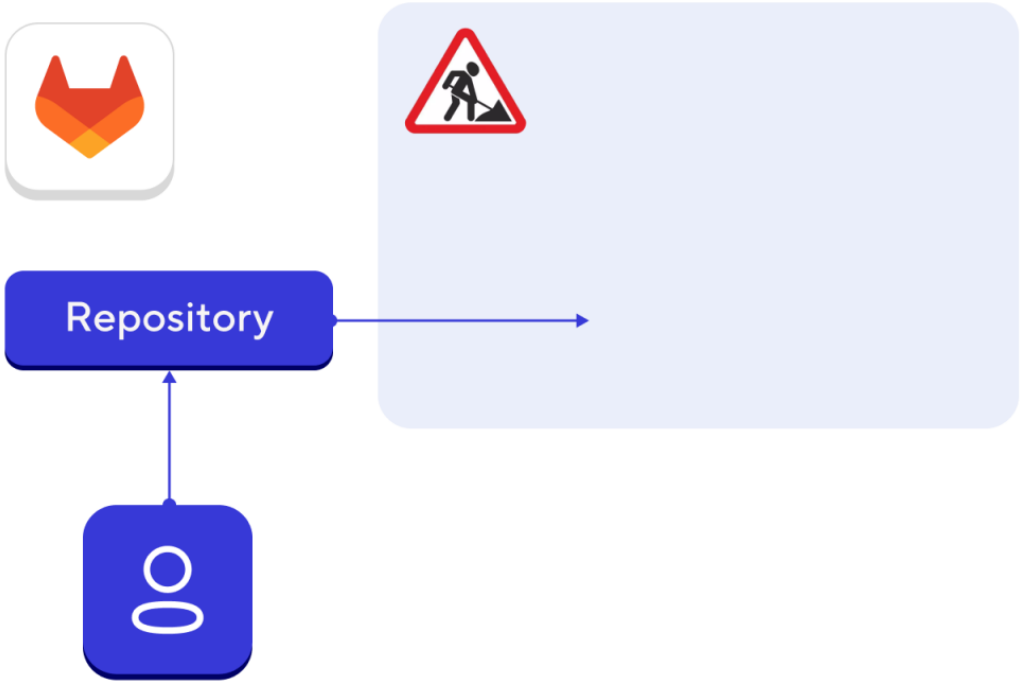
Какие проблемы?



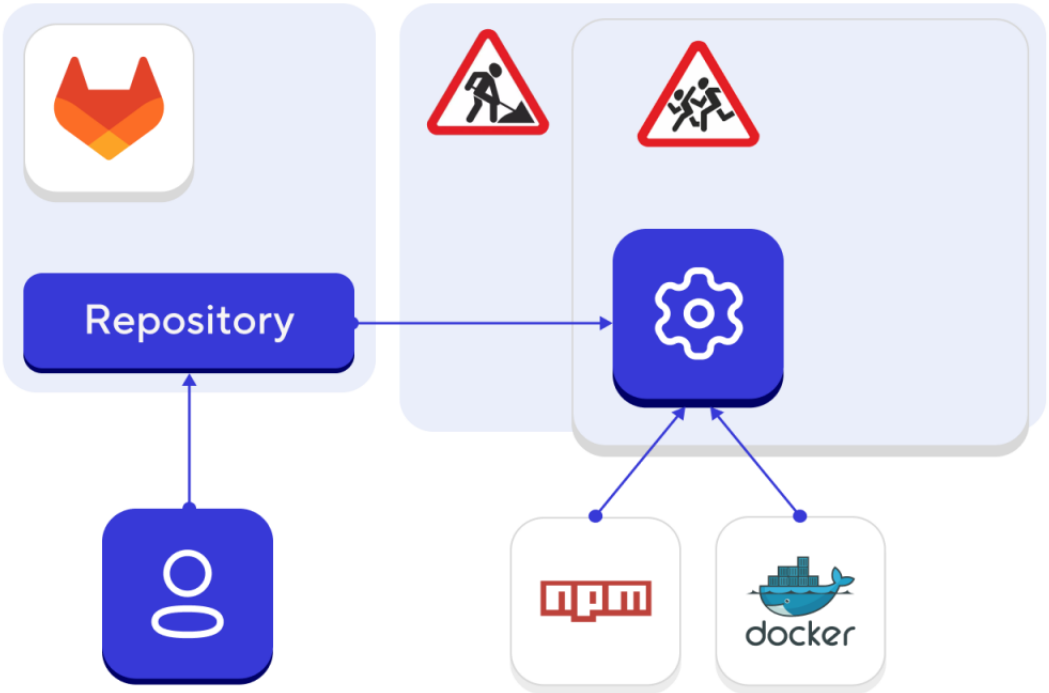
Появились SCM



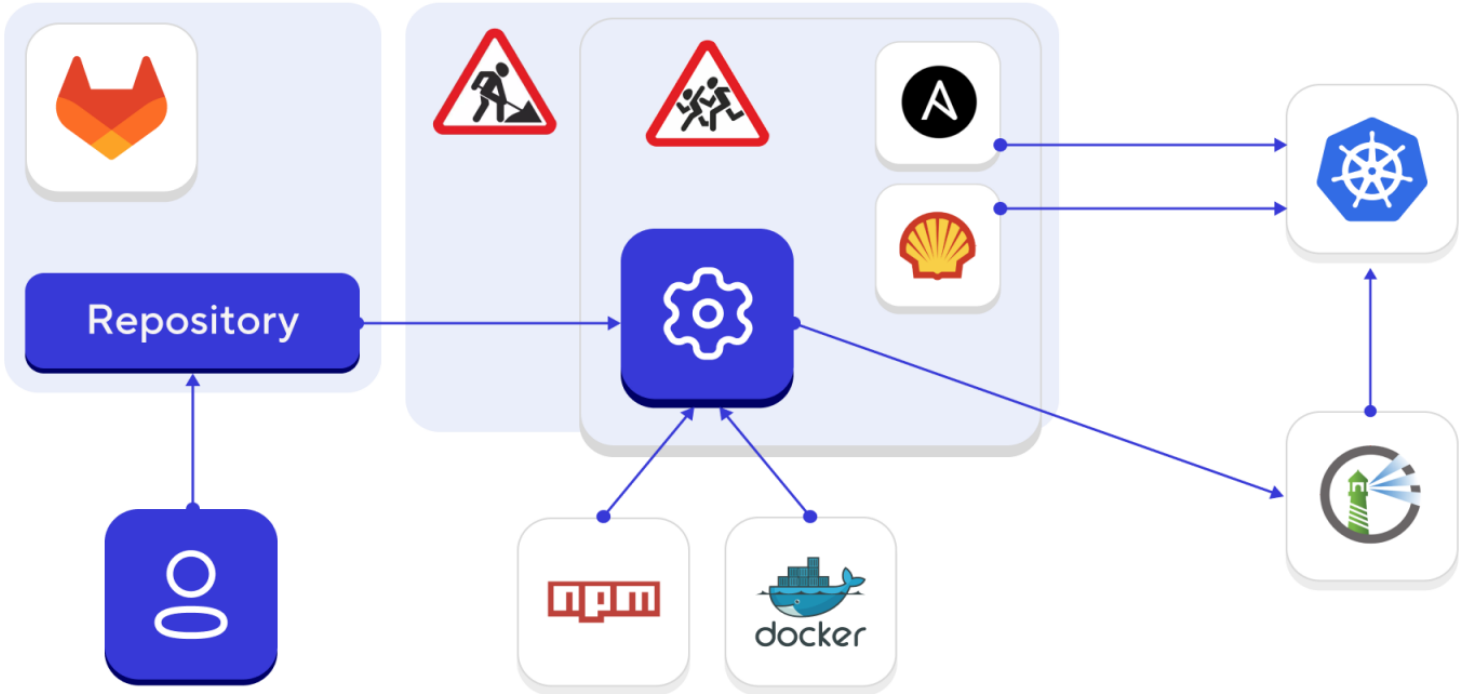
Код отправляется на воркер-ноду



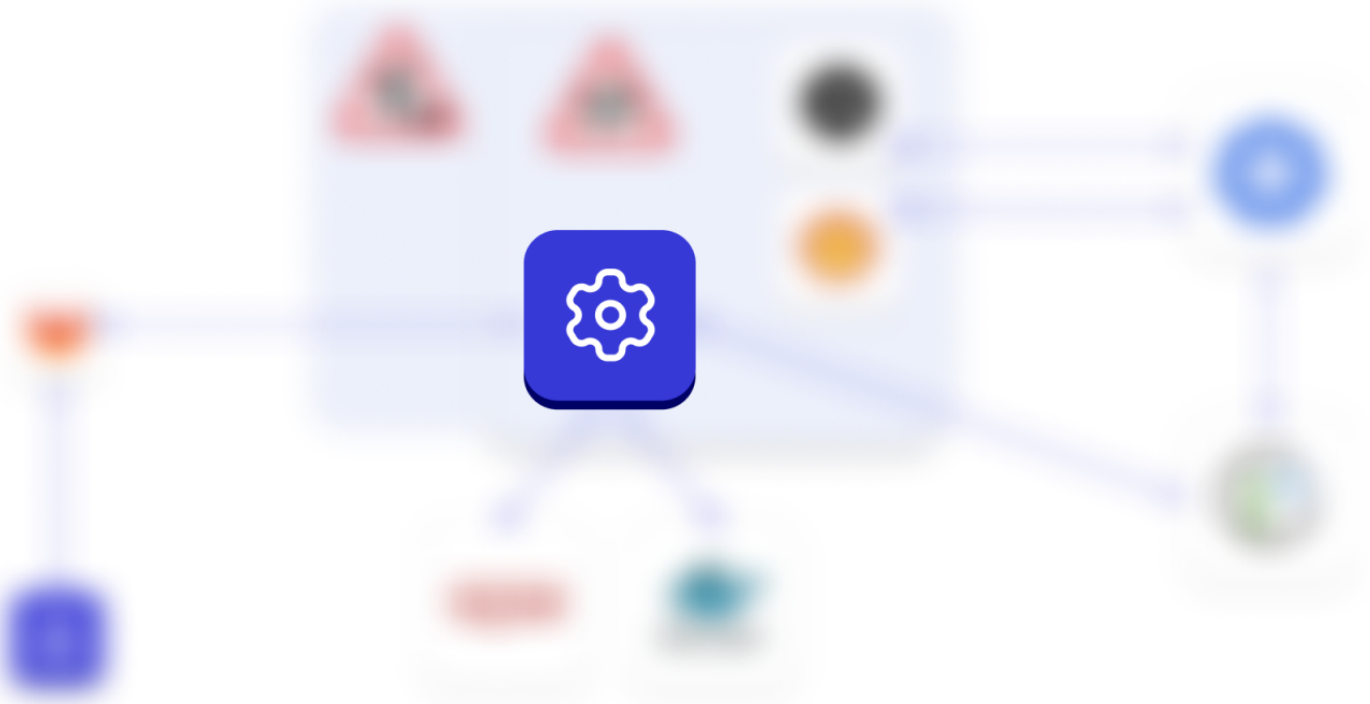
Сборка происходит удаленно



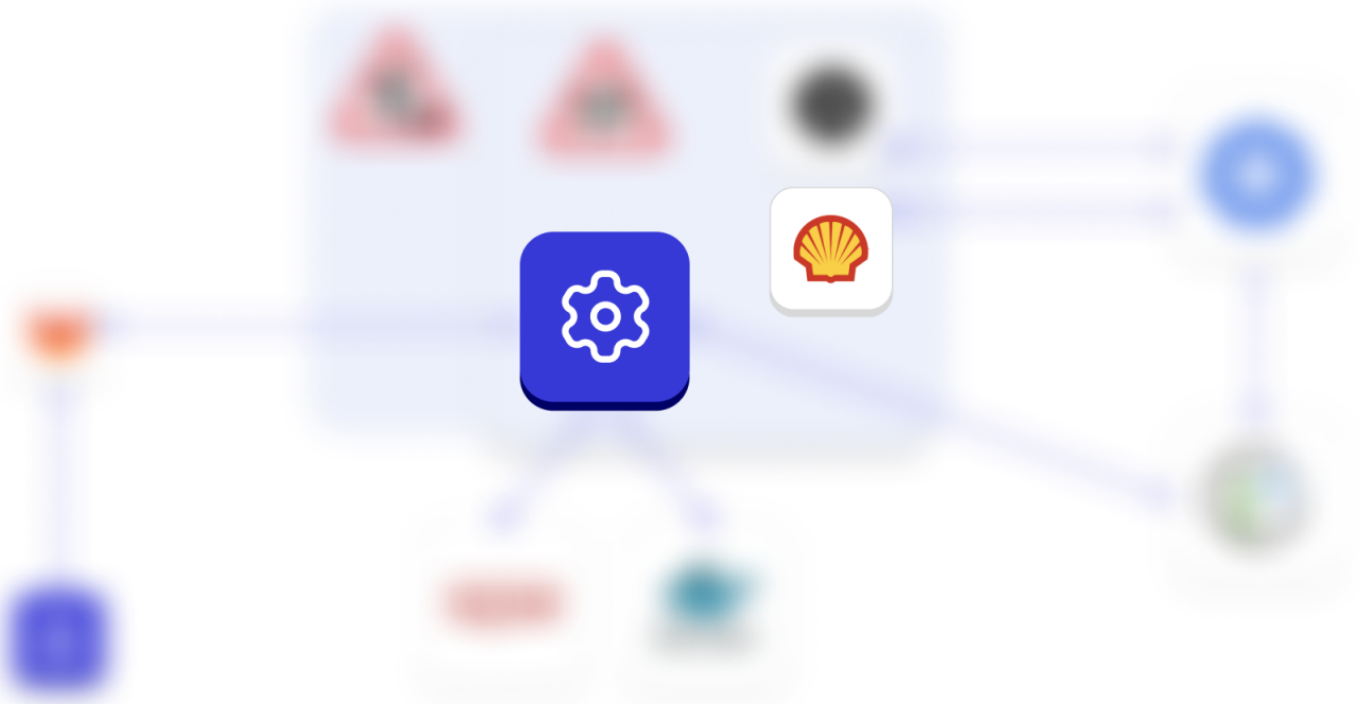
Результат выкатывается в prod



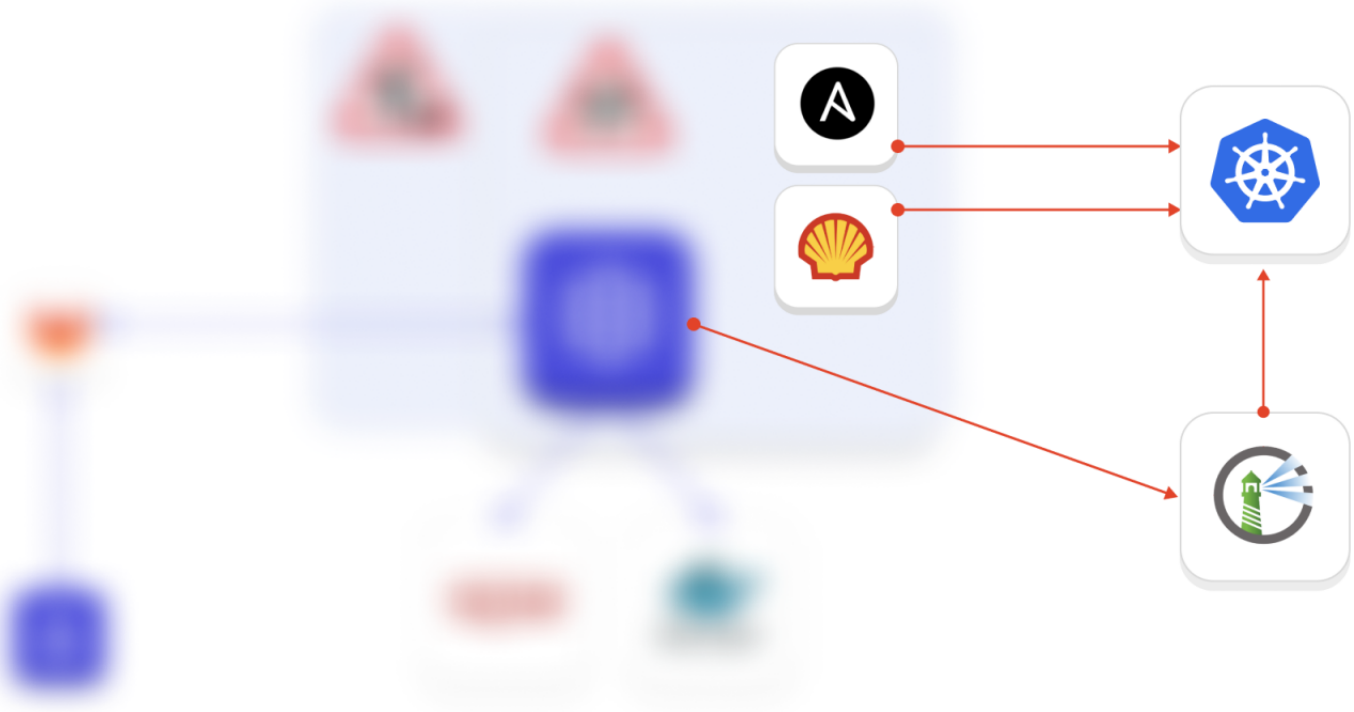
Проблема осталась



А еще у нас shell



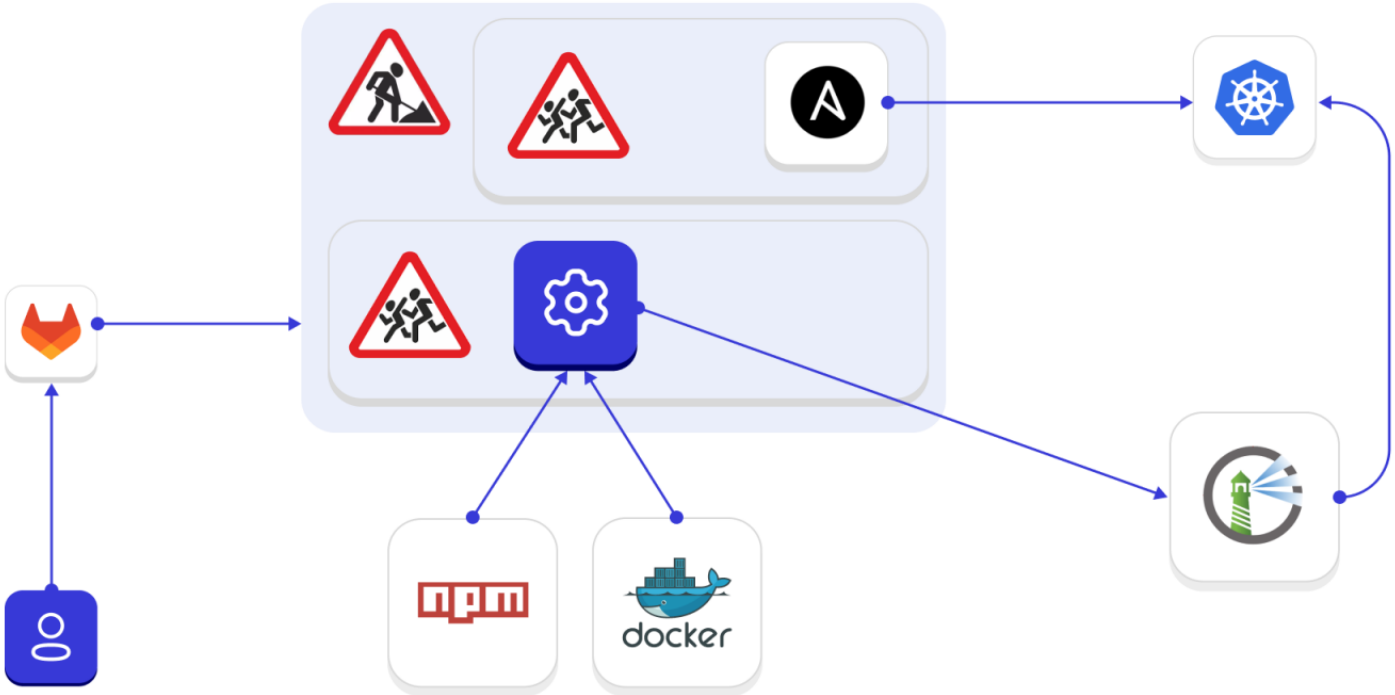
Произвольный код выкатывается на прод



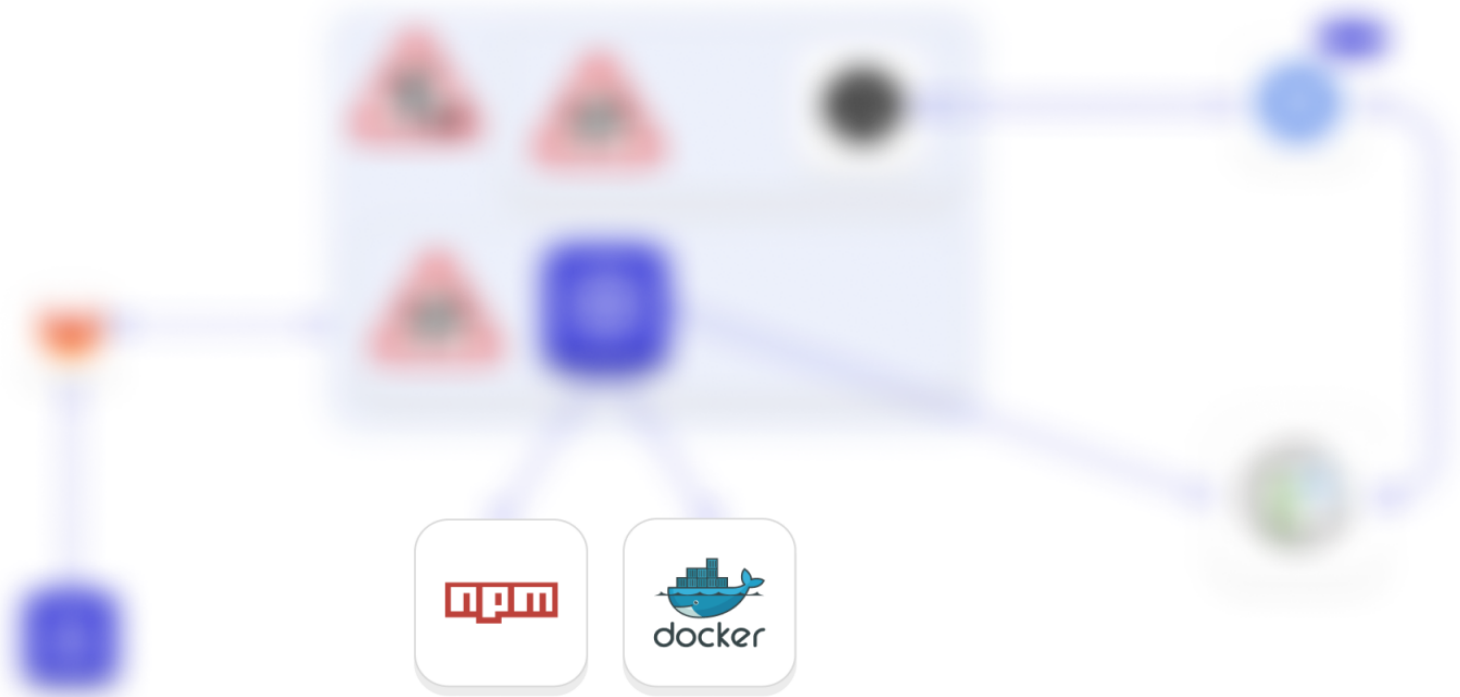
Checklist

- ✔ Не используем shell-раннеры

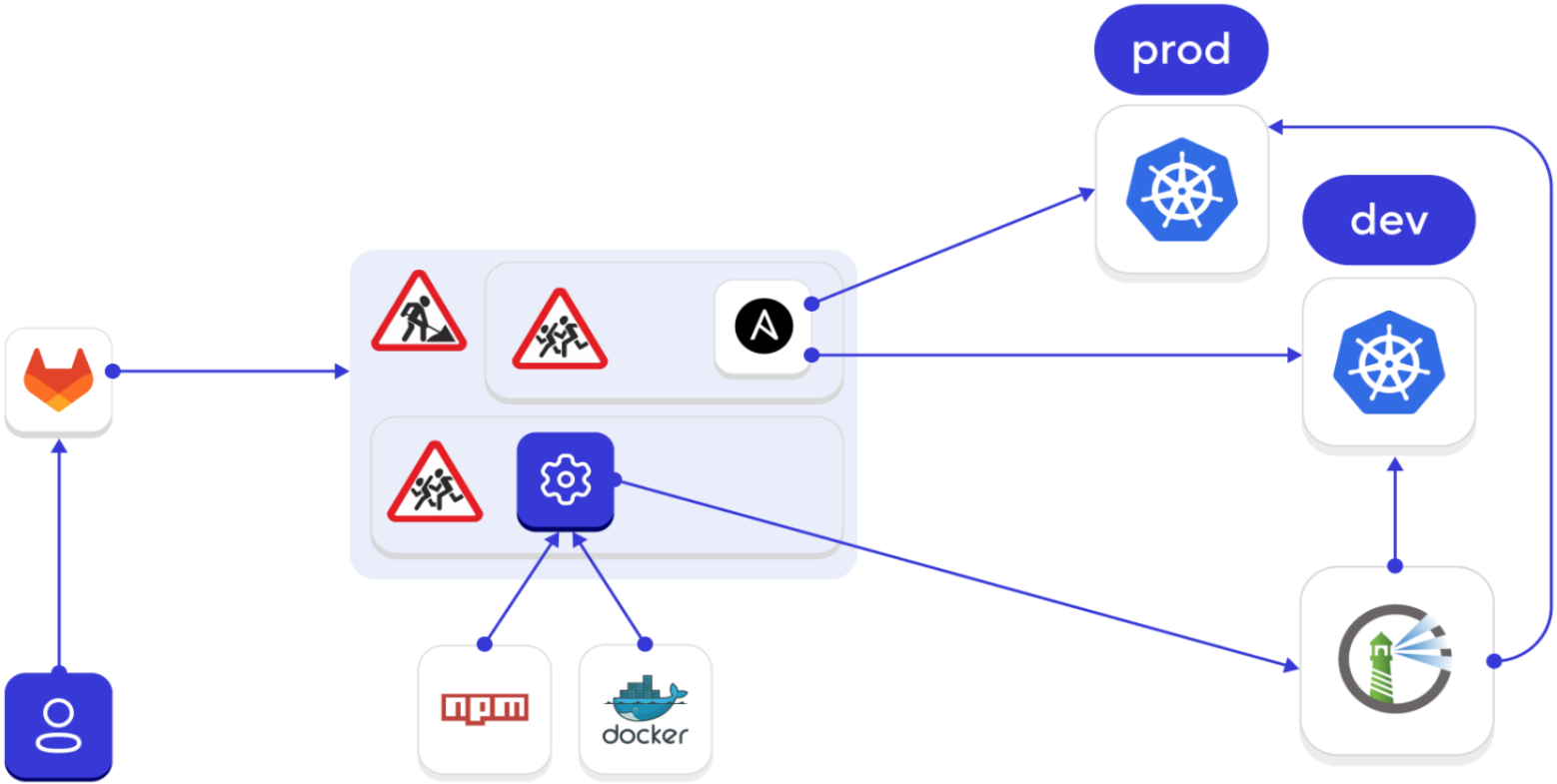
Все запускается в контейнерах



А вдруг мы скачали что-то не то?

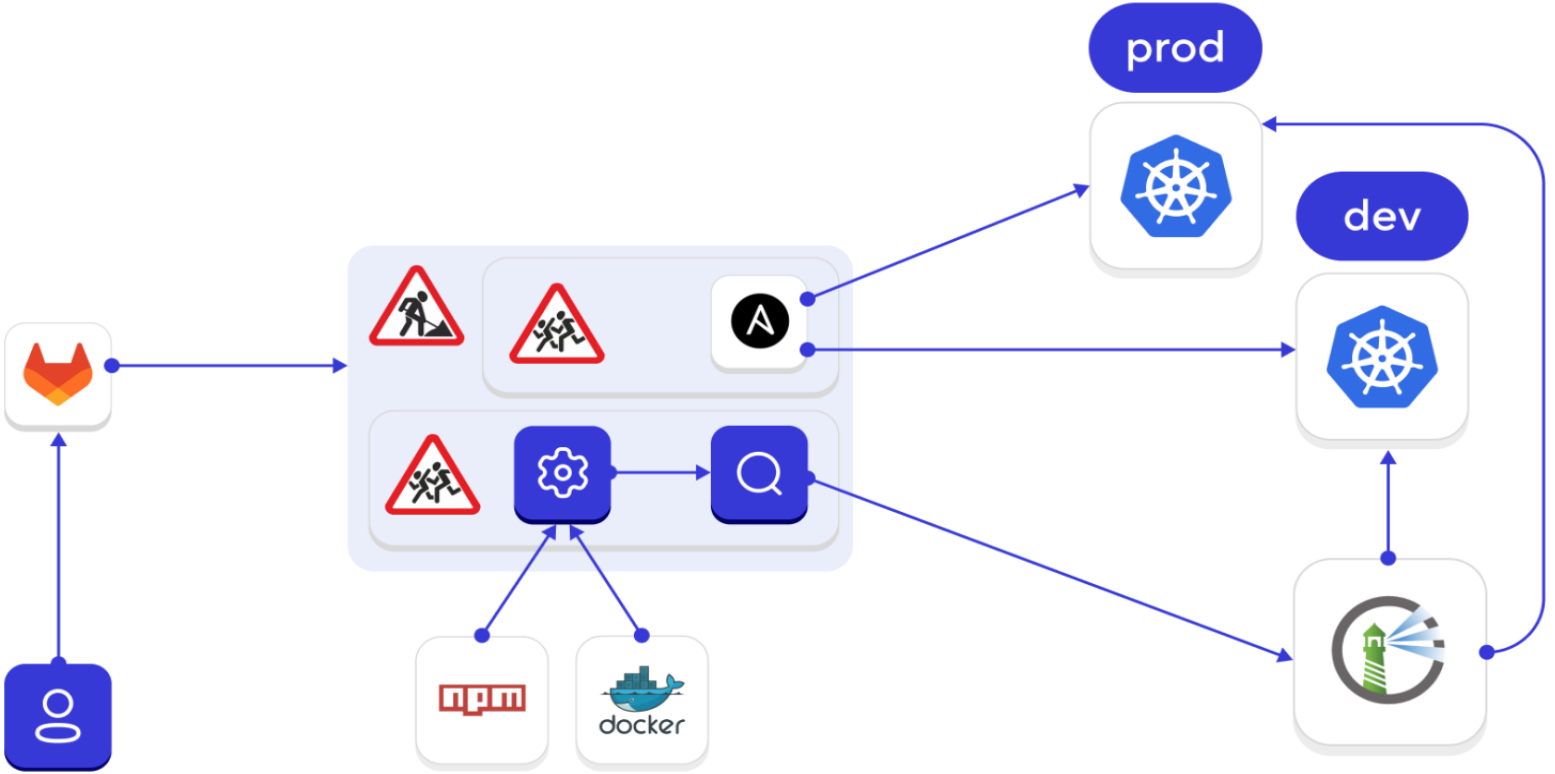


Добавляем dev-сегмент

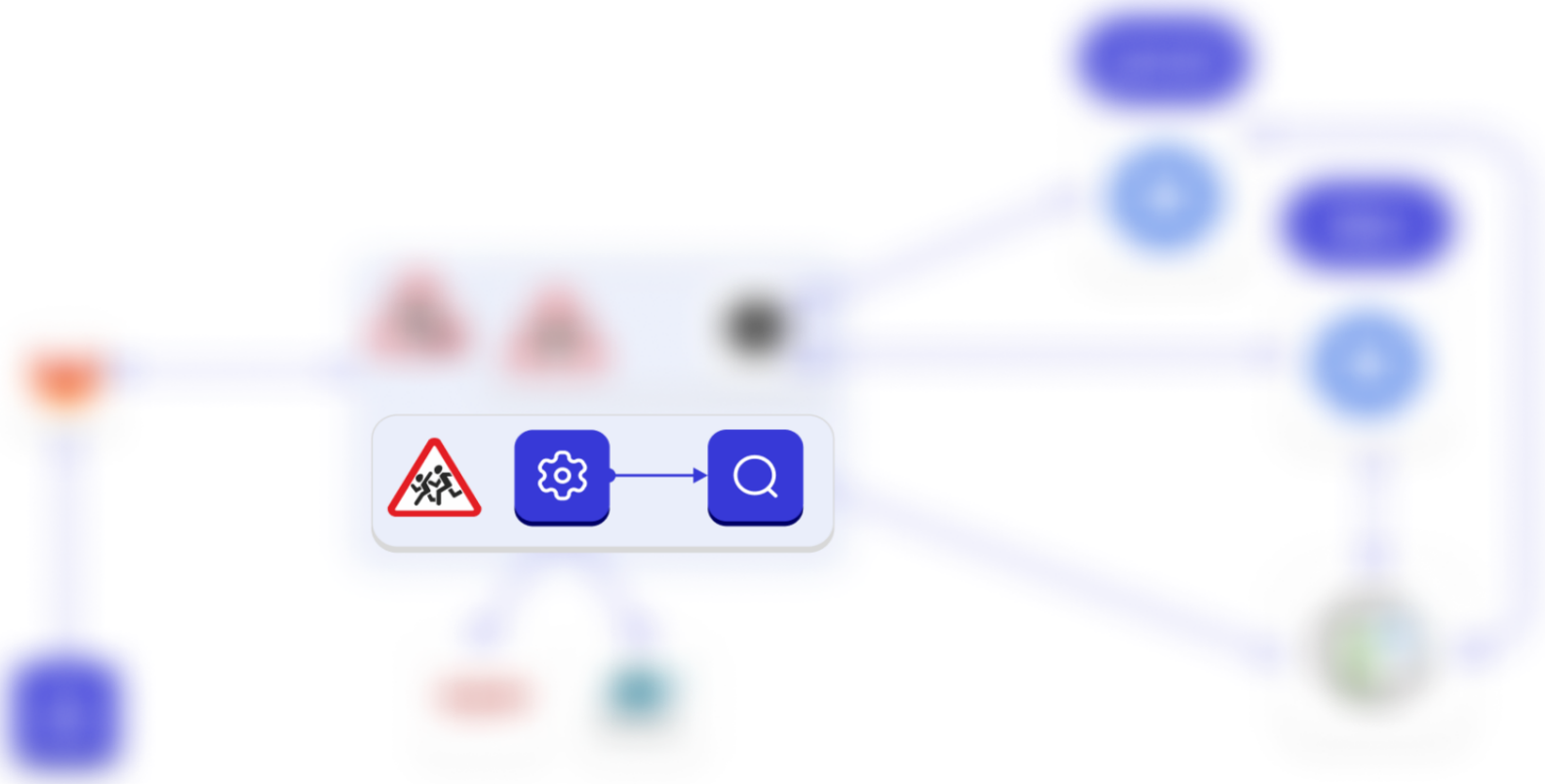


Как обезопаситься еще?

Добавляем сканеры



Аффектит ли bad dependency на сборку?



Конечно!

ведь нужен docker-in-docker

Техники побега из Docker

№	Название техники	Минимальные Capabilities
1	Монтирование хостовой системы	SYS_ADMIN
2	Использование примонтированного Docker Socket	_
3	Инъекция Shell-code	SYS_PTRACE
4	Инъекция модуля ядра	SYS_MODULE
5	Чтение секретов с хоста	DAC_READ_SEARCH
6	Перезапись файлов на хосте	DAC_READ_SEARCH, DAC_OVERRIDE
7	«notify_on_release» в cgroups	SYS_ADMIN, DAC_OVERRIDE

Примонтированный docker-socket

- /var/run/docker.sock
- Для Docker-in-Docker (DinD) сборок иногда монтируется напрямую в контейнер
- Позволяет нам прямо из контейнера посылать команды на хостовой Docker (например, для сборок)


НО ЕСТЬ
ОДИН
НЮАНС

```
root@ln-khakimov-vm:~# docker run -it --cap-drop=ALL -v
/var/run/docker.sock:/run/docker.sock docker:dind /bin/sh
```

```
root@ln-khakimov-vm:~# docker run -it --cap-drop=ALL -v
/var/run/docker.sock:/run/docker.sock docker:dind /bin/sh

/ # docker run -it --privileged -v /:/host/ ubuntu bash -c "chroot /host/”
```

```
root@ln-khakimov-vm:~# docker run -it --cap-drop=ALL -v
/var/run/docker.sock:/run/docker.sock docker:dind /bin/sh

/ # docker run -it --privileged -v /:/host/ ubuntu bash -c "chroot /host/”

# ls
bin   dev  home  lib32 libx32     media  opt root  sbin  srv  tmp  var
boot  etc  lib lib64 lost+found  mnt    proc  run   snap  sys  usr
```

```
root@ln-khakimov-vm:~# docker run -it --cap-drop=ALL -v
/var/run/docker.sock:/run/docker.sock docker:dind /bin/sh

/ # docker run -it --privileged -v /:/host/ ubuntu bash -c "chroot /host/”

# ls
bin   dev  home  lib32 libx32     media  opt root  sbin  srv  tmp  var boot  etc  lib
lib64 lost+found  mnt    proc  run   snap  sys  usr

# whoami
root
```

CAP обычного контейнера


```
1 meltdown@docker:~$ docker exec -it mynginx bash
2 root@test:/# capsh --print
3 Current:
  cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_
  bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap=ep
4 Bounding set
  =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net
  _bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap
5 Ambient set =
6 Current IAB:
  !cap_dac_read_search,!cap_linux_immutable,!cap_net_broadcast,!cap_net_admin,!cap_ipc_lock,!cap_ipc_o
  wner,!cap_sys_module,!cap_sys_rawio,!cap_sys_ptrace,!cap_sys_pacct,!cap_sys_admin,!cap_sys_boot,!cap
  _sys_nice,!cap_sys_resource,!cap_sys_time,!cap_sys_tty_config,!cap_lease,!cap_audit_control,!cap_mac
  _override,!cap_mac_admin,!cap_syslog,!cap_wake_alarm,!cap_block_suspend,!cap_audit_read,!cap_perfmon
  ,!cap_bpf,!cap_checkpoint_restore
```

CAP privileged-контейнера


```
meltdown@docker:~$ docker exec -it privnginx bash
root@c8756b6e1411:/# capsh --print
Current: =ep
Bounding set
=cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid
,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw,ca
p_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,ca
p_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap
_lease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_w
ake_alarm,cap_block_suspend,cap_audit_read,cap_perfmon,cap_bpf,cap_checkpoint_restore
Ambient set =
Current IAB:
```

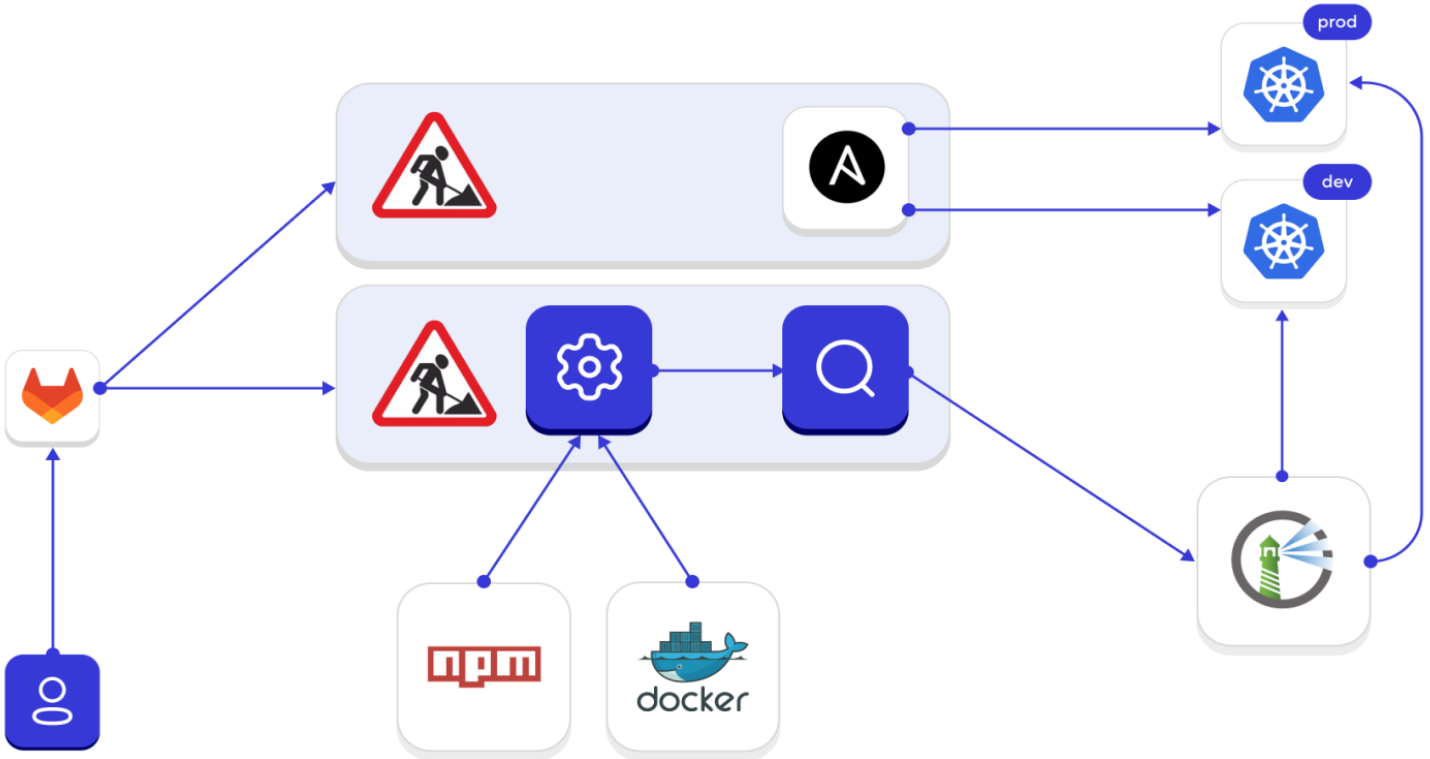
Был AppArmor и нет его


```
meltdown@docker:~/ubuntu$ docker ps --quiet --all | xargs docker inspect --format '{{ .Name }}: AppArmorProfile={{ .AppArmorProfile }}'
/privnginx: AppArmorProfile=unconfined
/unruffled_hawking: AppArmorProfile=docker-default
/mynginx: AppArmorProfile=docker-default
```

Checklist

- ✔ Не используем shell-раннеры
- ✔ Dind = нет изоляции
  (используйте kaniko)

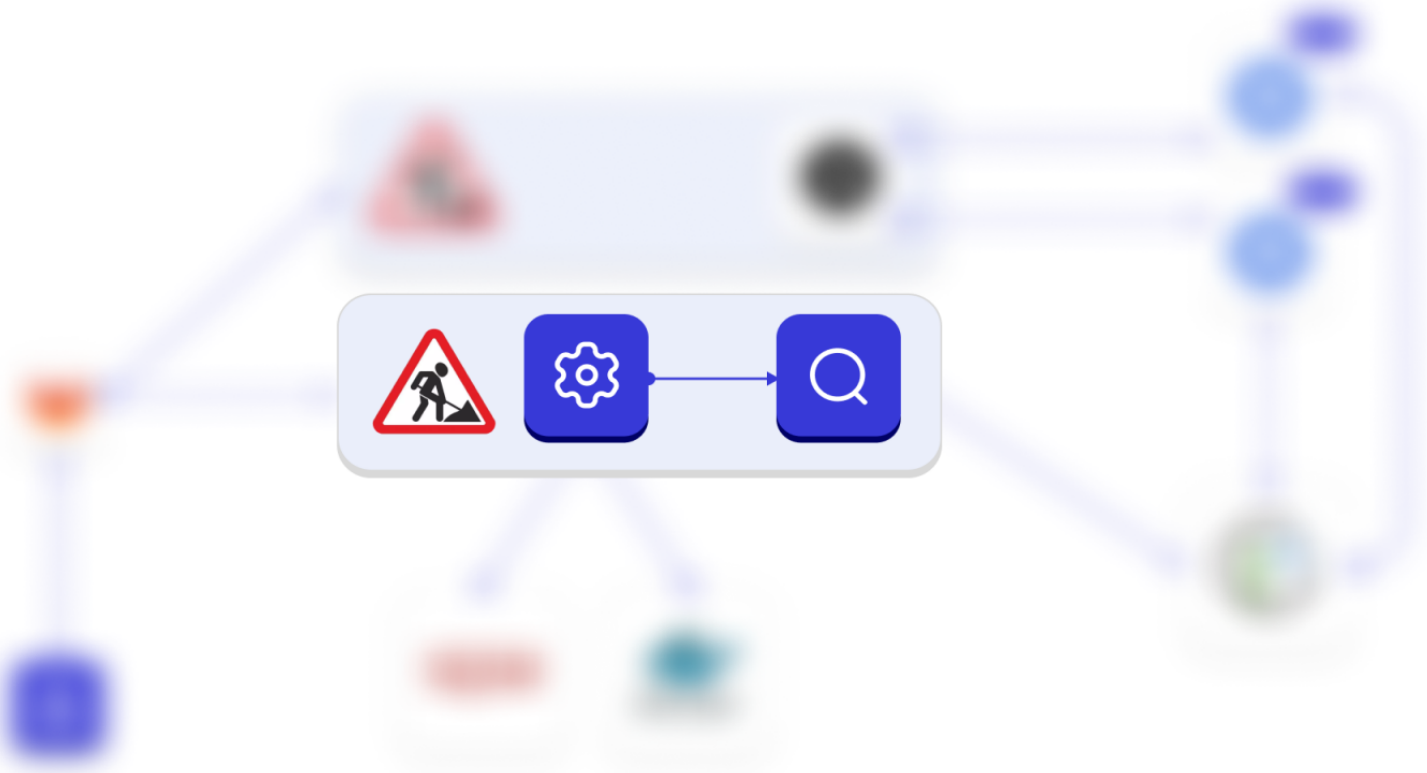
Разделим воркер-ноды



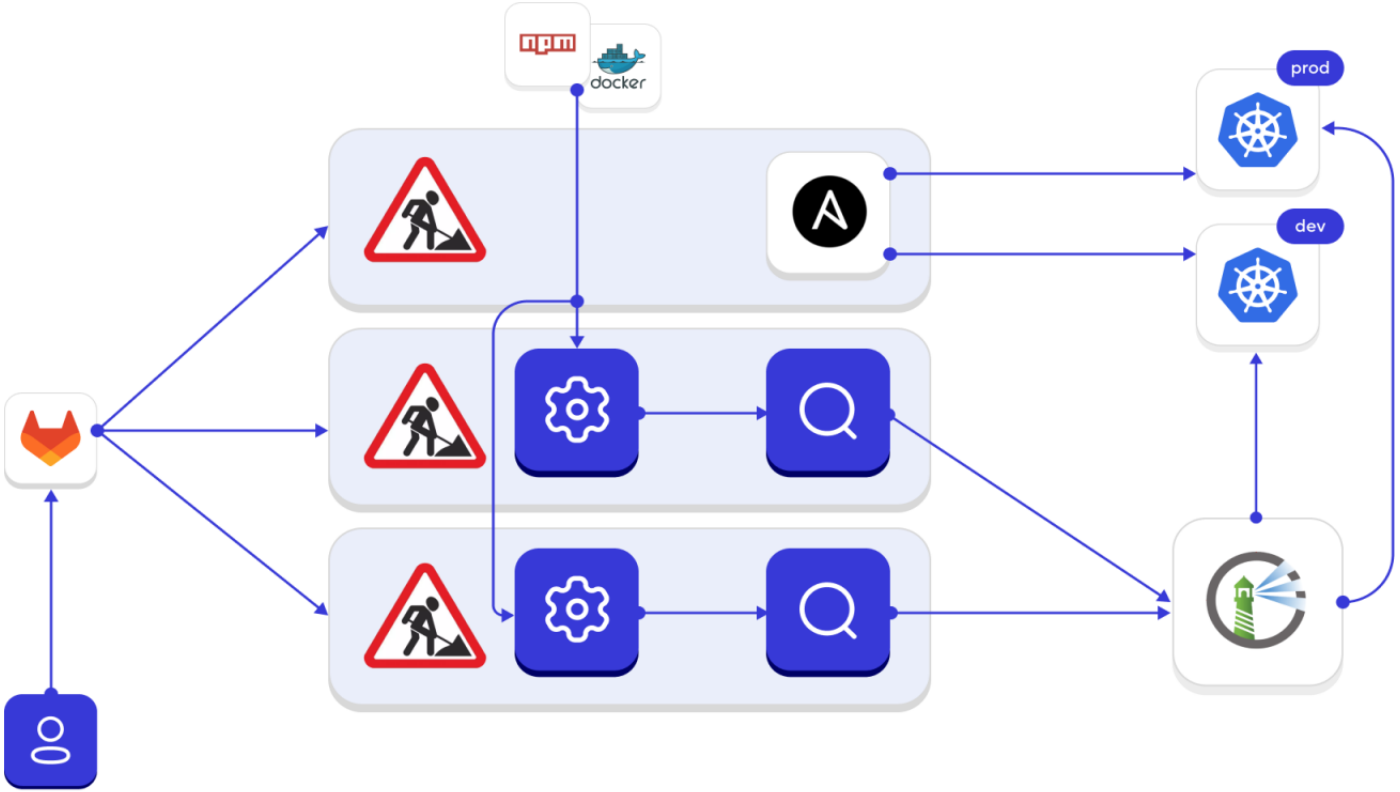
Checklist

- ✔ Не используем shell-раннеры
- ✔ Dind = нет изоляции
  (используйте kaniko)
- ✔ build и deploy
  на разных воркер-нодах

Но все еще код из dev может попасть в prod



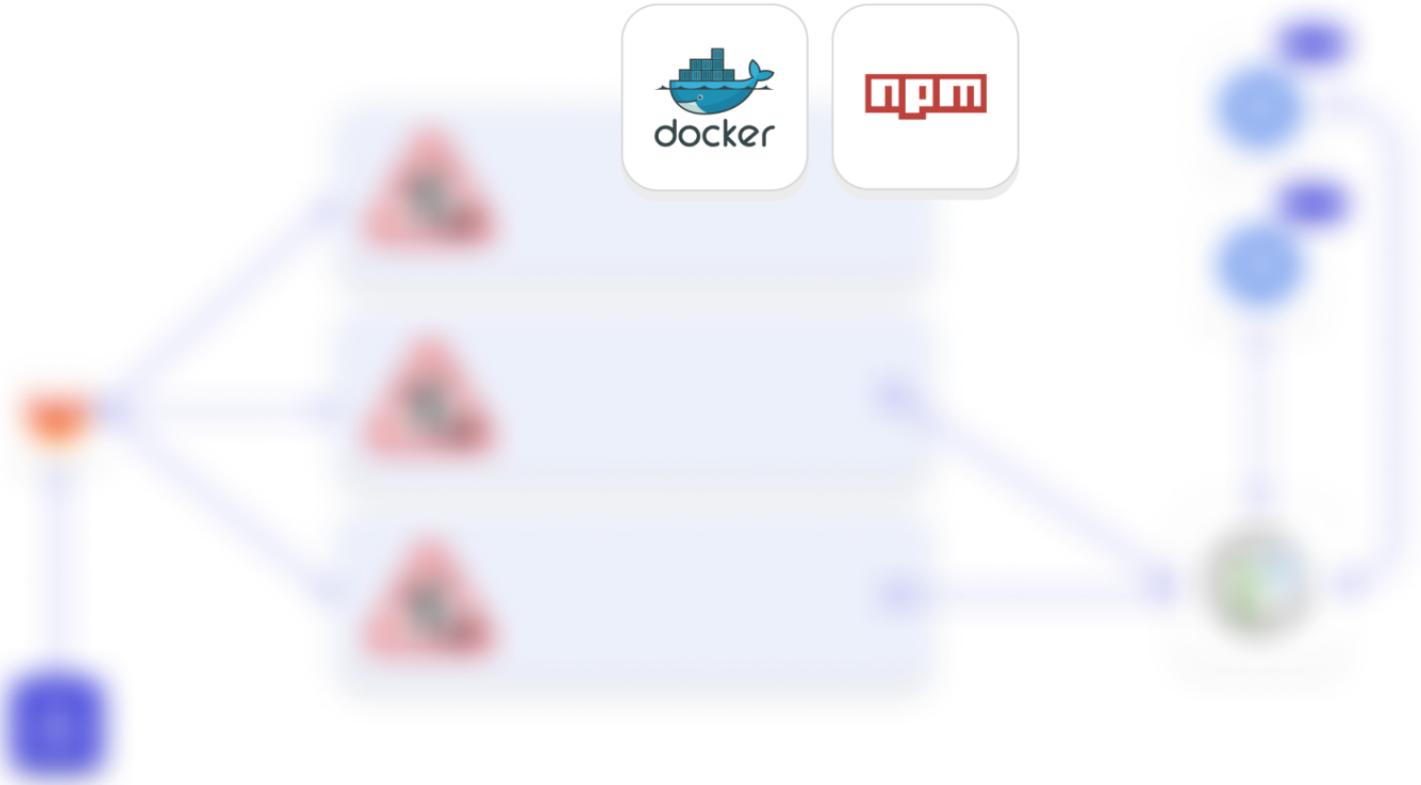
Разделим build-среды



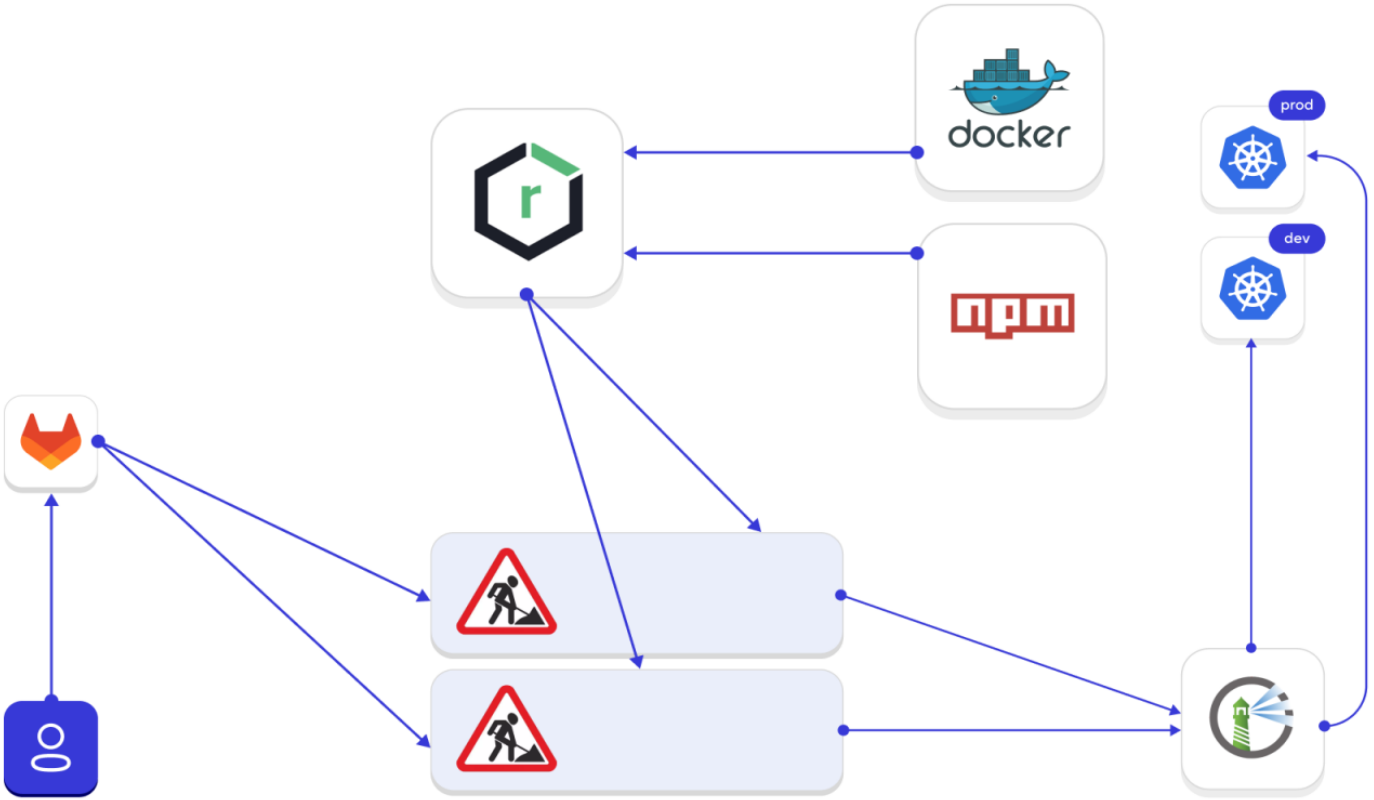
Checklist

- ✔ Не используем shell-раннеры
- ✔ Dind = нет изоляции
  (используйте kaniko)
- ✔ build и deploy
  на разных воркер-нодах
- ✔ раздельные ноды для build-сред

Как управлять зависимостями?

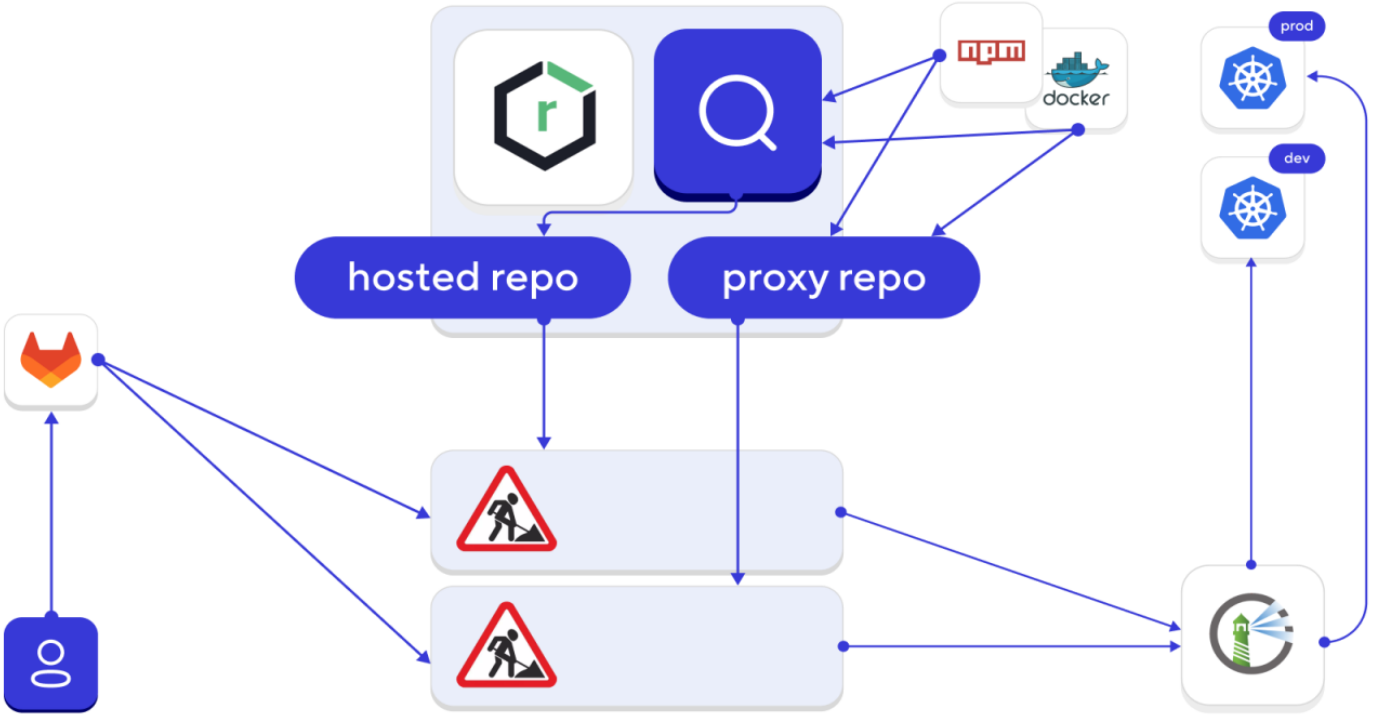


Храним зависимости локально



Как защитить прод?

Делаем прокси и хостед

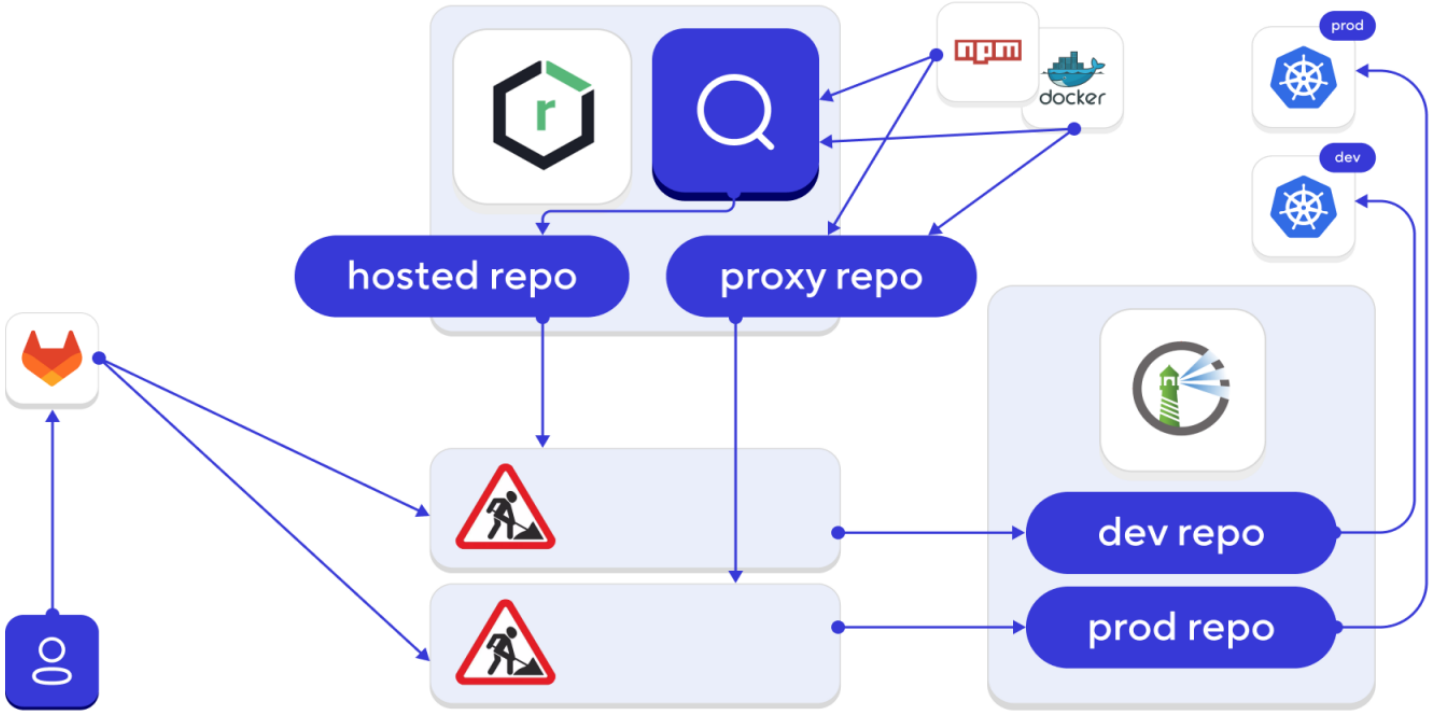


Checklist

- ✔ Не используем shell-раннеры
- ✔ Dind = нет изоляции
  (используйте kaniko)
- ✔ build и deploy
  на разных воркер-нодах
- ✔ раздельные ноды для build-сред
- ✔ отдельные репы для прод-
  и билд-сборок

Про что мы забыли?

Раздельные репозитории в registry

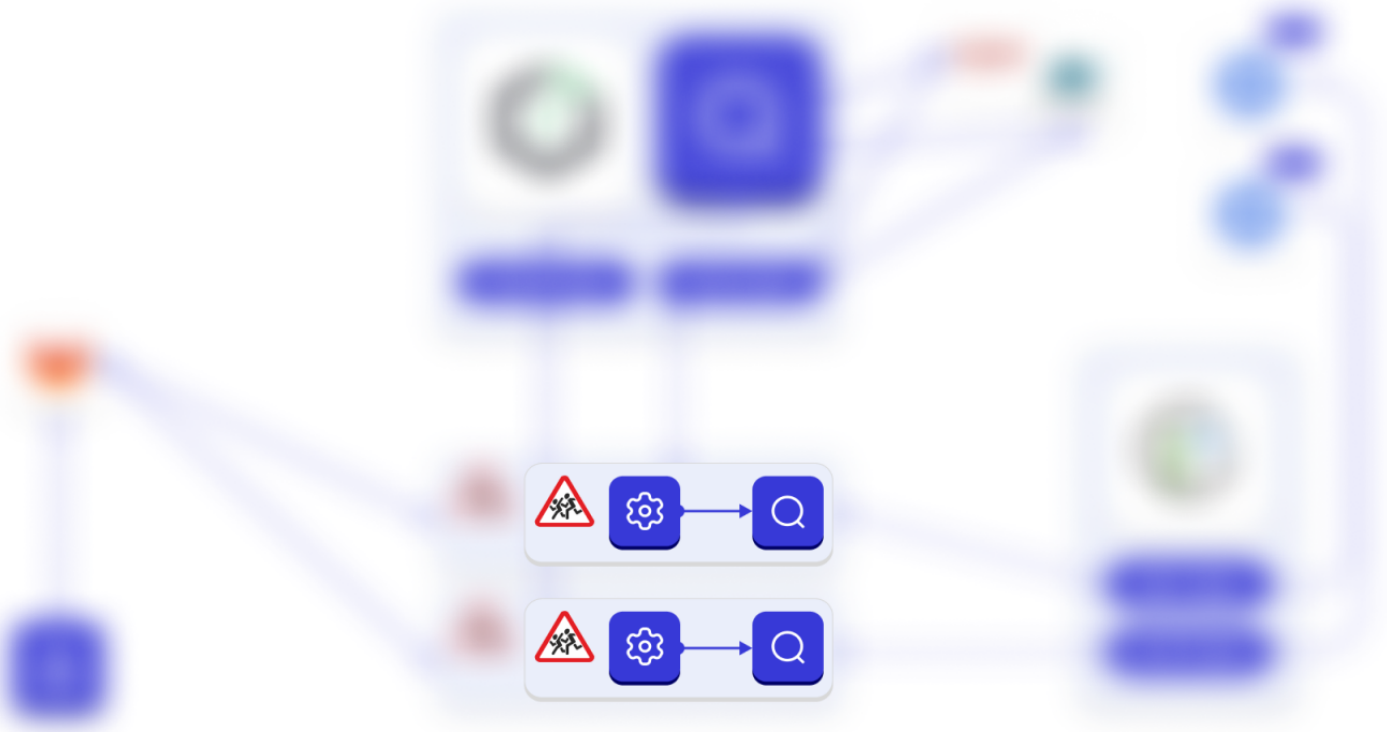


Checklist

- ✔ Не используем shell-раннеры
- ✔ Dind = нет изоляции
  (используйте kaniko)
- ✔ build и deploy
  на разных воркер-нодах
- ✔ раздельные ноды для build-сред
- ✔ отдельные репы для прод-
  и билд-сборок
- ✔ отдельные репозитории
  и отдельные ключи! для
  репозиториев registry


One more thing...

Мечтают ли сканеры просканироваться?



Взламываем через Snyk

Snyk напрямую выполняет команды из gradlew на целевой системе

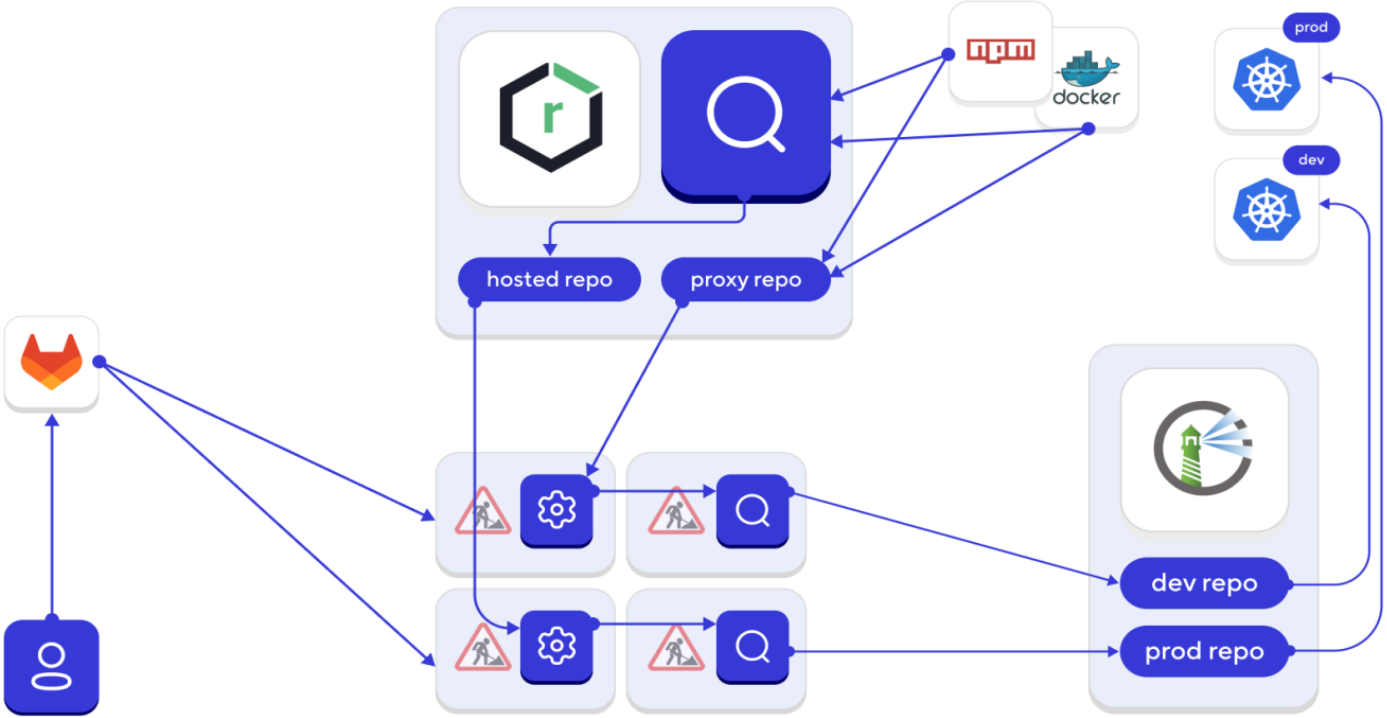
Взламываем через Snyk

Snyk напрямую выполняет команды из gradlew на целевой системе

```
$ gradlew
1    #!/bin/sh
2    touch hacked_snyk_gradle
3
```

```
meltdown@ALEKSEYs-MacBook-Pro snyk_gradle % ls
build.gradle    gradlew
meltdown@ALEKSEYs-MacBook-Pro snyk_gradle % snyk test
Gradle Error (short):


===== DEBUG INFORMATION START =====

No line prefixed with "JSONDEPS " was returned; full output:

===== DEBUG INFORMATION END =====

Error running Gradle dependency analysis.

Please ensure you are calling the `snyk` command with correct arguments.
If the problem persists, contact support@snyk.io, providing the full error
message from above, starting with ===== DEBUG INFORMATION START =====.
meltdown@ALEKSEYs-MacBook-Pro snyk_gradle % ls
build.gradle            gradlew                 hacked_snyk_gradle
meltdown@ALEKSEYs-MacBook-Pro snyk_gradle %
```

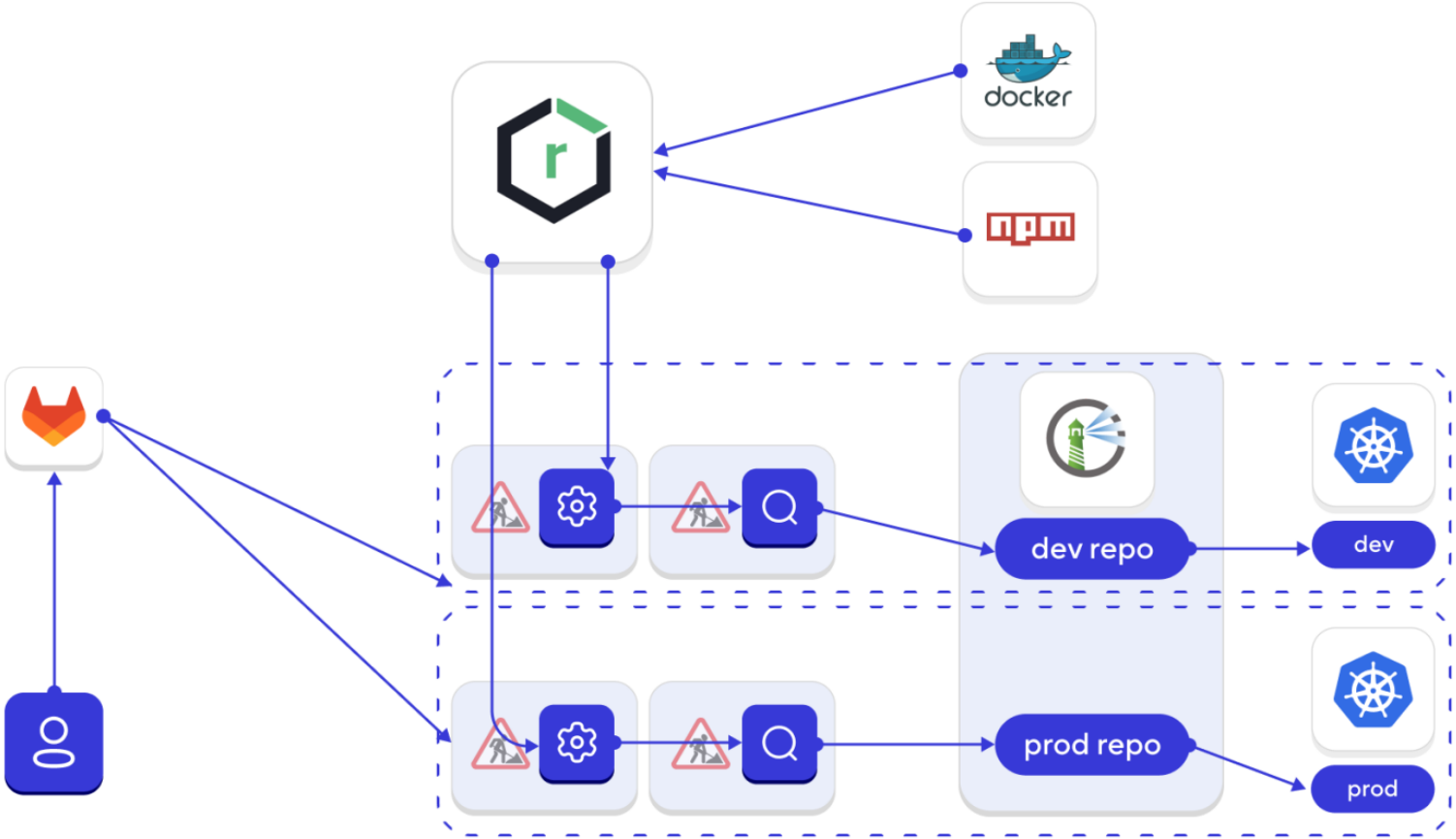
Отделяем сканеры



Checklist

- ✔ Не используем shell-раннеры
- ✔ Dind = нет изоляции (используйте kaniko)
- ✔ build и deploy на разных воркер-нодах
- ✔ раздельные ноды для build-сред
- ✔ отдельные репы для прод- и билд-сборок
- ✔ отдельные репозитории и отдельные ключи! для репозиториев registry
- ✔ на build-ноде только сборки! сканеры отдельно

Не забываем про FW



Stage окружение в dev кластере, которое смотрит в prod базы (с)

Checklist

- ✔ Не используем shell-раннеры
- ✔ Dind = нет изоляции (используйте kaniko)
- ✔ build и deploy на разных воркер-нодах
- ✔ раздельные ноды для build-сред
- ✔ отдельные репы для прод- и билд-сборок
- ✔ отдельные репозитории и отдельные ключи! для репозиториев registry
- ✔ на build-ноде только сборки! сканеры отдельно
- ✔ изолируем контуры друг от друга

Эфемерные (временные) раннеры

Checklist

- ✔ Не используем shell-раннеры
- ✔ Dind = нет изоляции (используйте kaniko)
- ✔ build и deploy на разных воркер-нодах
- ✔ раздельные ноды для build-сред
- ✔ отдельные репы для прод- и билд- сборок
- ✔ отдельные репозитории и отдельные ключи! для репозиториев registry
- ✔ на build-ноде только сборки! сканеры отдельно
- ✔ изолируем контуры друг от друга
- ✔ эфемерные раннеры

Приемка Open Source

- Добавление зависимостей в репозиторий только доверенными лицами

Приемка Open Source

- Добавление зависимостей в репозиторий только доверенными лицами
- Использование SCA/OSA

Приемка Open Source

- Добавление зависимостей в репозиторий только доверенными лицами
- Использование SCA/OSA
- Проверка зависимостей при добавлении (происхождение зависимости, количество загрузок, наличие CVE, Issues, контроль целостности зависимости)

Приемка Open Source

- Добавление зависимостей в репозиторий только доверенными лицами
- Использование SCA/OSA
- Проверка зависимостей при добавлении (происхождение зависимости, количество загрузок, наличие CVE, Issues, контроль целостности зависимости)
- Тестирование зависимостей (использование в песочнице, отслеживание syscalls и т.д.)

Приемка Open Source

- Добавление зависимостей в репозиторий только доверенными лицами
- Использование SCA/OSA
- Проверка зависимостей при добавлении (происхождение зависимости, количество загрузок, наличие CVE, Issues, контроль целостности зависимости)
- Тестирование зависимостей (использование в песочнице, отслеживание syscalls и т.д.)
- Проверка соответствия зависимостей в окружении

Приемка Open Source

- Добавление зависимостей в репозиторий только доверенными лицами
- Использование SCA/OSA
- Проверка зависимостей при добавлении (происхождение зависимости, количество загрузок, наличие CVE, Issues, контроль целостности зависимости)
- Тестирование зависимостей (использование в песочнице, отслеживание syscalls и т.д.)
- Проверка соответствия зависимостей в окружении
- Хардкор: дизассемблер и декомпилятор

Checklist

- ✔ Не используем shell-раннеры
- ✔ Dind = нет изоляции (используйте kaniko)
- ✔ build и deploy на разных воркер-нодах
- ✔ раздельные ноды для build-сред
- ✔ отдельные репы для прод- и билд-сборок
- ✔ отдельные репозитории и отдельные ключи! для репозиториев registry
- ✔ на build-ноде только сборки! сканеры отдельно
- ✔ изолируем контуры друг от друга
- ✔ эфемерные раннеры
- ✔ тестируем зависимости

Не забываем про подпись

- Подпись коммитов
- Подпись зависимостей
- Проверка аутентичности артефактов при сборке приложения
- Подпись самого приложения
- Проверка артефактов при сборке образа и подпись образа

Kyverno Cosign Policy

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: check-image
spec:
  validationFailureAction: enforce
  background: false
  webhookTimeoutSeconds: 30
  failurePolicy: Fail
  rules:
    - name: check-image
      match:
        any:
        - resources:
            kinds:
              - Pod
      verifyImages:
      - imageReferences:
        - "your.registry/id"
        attestors:
        - count: 1
          entries:
          - keys:
              publicKeys: |-
                <содержимое_cosign.pub>
```

Checklist

- ✔ Не используем shell раннеры
- ✔ Dind = нет изоляции (используйте kaniko)
- ✔ build и deploy на разных воркер нодах
- ✔ раздельные ноды для build сред
- ✔ отдельные репы для прод и билд сборок
- ✔ отдельные репозитории и отдельные ключи! для репозиториев registry
- ✔ на build ноде только сборки! сканеры отдельно
- ✔ изолируем контура друг от друга
- ✔ эфемерные раннеры
- ✔ тестируем зависимости
- ✔ не забываем про подпись и валидацию

Спасибо за внимание!

Алексей Федулаев
DevSecOps Team Lead
Wildberries
Tg: @int0x80h