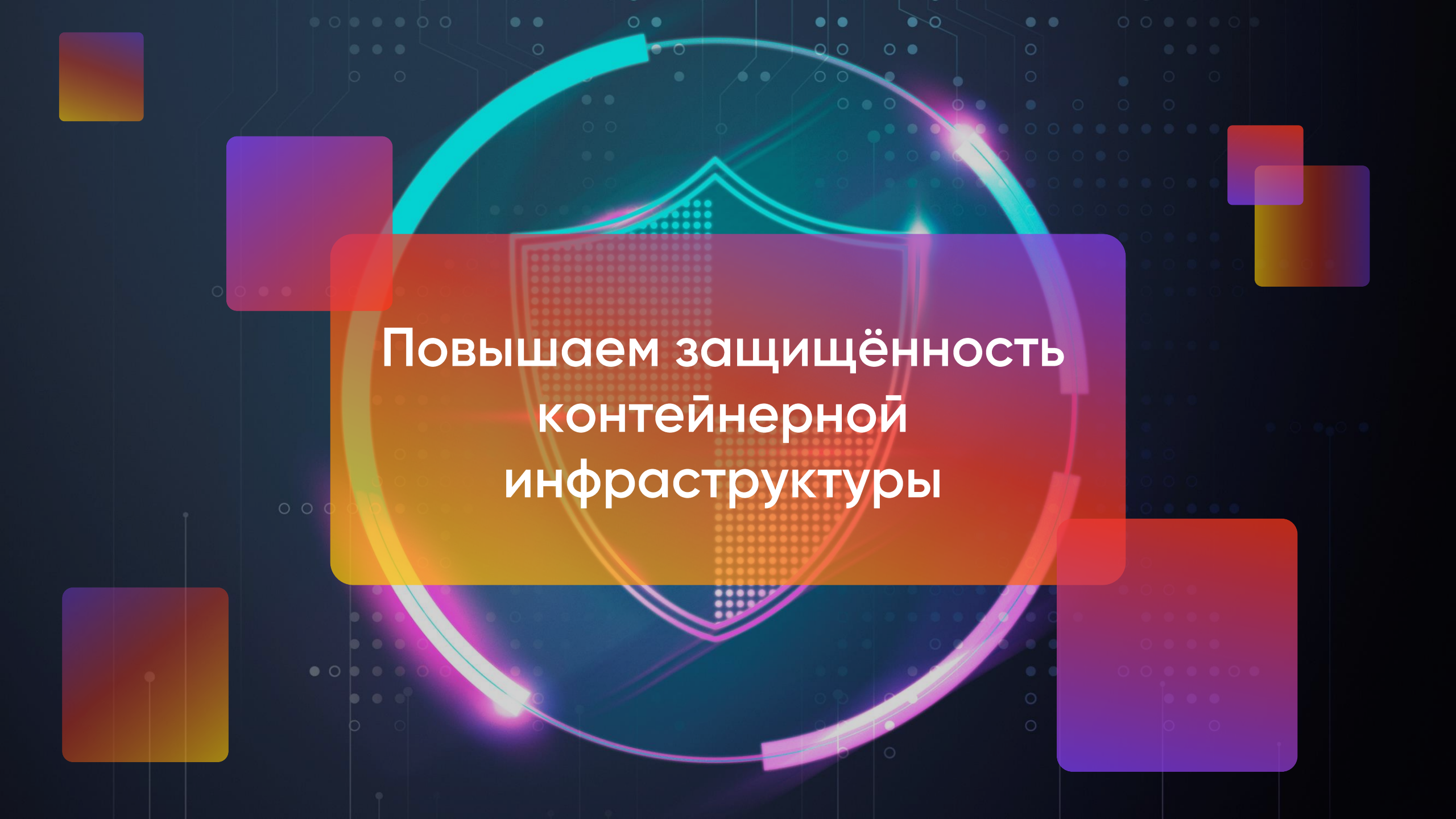


The background is a dark blue field with a subtle grid of small white dots and faint circuit-like lines. A large, glowing shield shape is centered, composed of concentric arcs in cyan, magenta, and yellow. Overlaid on this are several semi-transparent squares in various colors: a small orange one in the top-left, a purple one in the top-left, a red one in the top-left, a purple one in the top-right, a yellow one in the top-right, a purple one in the bottom-left, a yellow one in the bottom-left, a red one in the bottom-right, and a purple one in the bottom-right. The text is centered within the shield.

Повышаем защищённость
контейнерной
инфраструктуры

Александр Рахманный

Главный инженер-программист, ПСБ

10 лет

в инфраструктуре,
разработка IaC

3 года в ИБ

разработка
инструментов для ИБ

Разработка сканера уязвимостей Docker
контейнеров

Спикер конференций SafeCode, Positive
Hack Days



Содержание доклада

Базовый образ

Поговорим про многоэтапные сборки и выбор базового образа.

Настройка контейнера

Обсудим основные настройки контейнеров для повышения безопасности.

Работа с секретами

Как безопаснее передать секреты приложению?

Мониторинг и аудит

Какие средства можно использовать для аудита инфраструктуры?

Сканирование

Как и на каком этапе производить сканирование образов?

Многоэтапные сборки

1

Восстанавливаем проект в SDK контейнере

Перед каждой сборкой скачиваем последние версии базовых образов либо используем обновленный корпоративный образ.

Многоэтапные сборки

1

Восстанавливаем проект в SDK контейнере

Перед каждой сборкой скачиваем последние версии базовых образов либо используем обновленный корпоративный образ.

2

Собираем приложение и зависимости

В SDK контейнере собираем приложение и все его зависимости. Проходим модульные тесты, инжектируем конфигурацию (если необходимо). Используем `.dockerignore`.

Многоэтапные сборки

1

Восстанавливаем проект в SDK контейнере

Перед каждой сборкой скачиваем последние версии базовых образов либо используем обновленный корпоративный образ.

2

Собираем приложение и зависимости

В SDK контейнере собираем приложение и все его зависимости. Проходим модульные тесты, инжектируем конфигурацию (если необходимо). Используем `.dockerignore`.

3

Переносим результат в рантайм контейнер

Переносим в рантайм контейнер только те артефакты сборки, которые необходимы для работы приложения.

Пример dockerbuild

```
1  # Stage Build
2  FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build # Используем SDK для сборки проекта
3  ARG BUILD_CONFIGURATION=Release # Переменная для конфигурации сборки
4  WORKDIR /src
5  COPY ["WebAppDemo/WebAppDemo.csproj", "WebAppDemo/"]
6  RUN dotnet restore "./WebAppDemo/WebAppDemo.csproj"
7  COPY . .
8  WORKDIR "/src/WebAppDemo"
9  RUN dotnet build "./WebAppDemo.csproj" -c $BUILD_CONFIGURATION -o /app/build # Собираем проект
10
11  RUN dotnet test /src/tests # Запускаем тесты
12
13  # Stage Publish
14  FROM build AS publish
15  ARG BUILD_CONFIGURATION=Release
16  RUN dotnet publish "./WebAppDemo.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false # Публикуем проект в виде бинарных файлов
17
18  # Stage Final
19  FROM mcr.microsoft.com/dotnet/aspnet:8.0-cbl-mariner-distroless AS final
20  WORKDIR /app
21  COPY --from=publish /app/publish . # Копируем бинарники проекта в образ
22  ARG UID=10001 # Переменная для UID пользователя
23  RUN adduser \
24      --disabled-password \
25      --gecos "" \
26      --home "/nonexistent" \
27      --shell "/sbin/nologin" \
28      --no-create-home \
29      --uid "${UID}" \
30      appuser
31  USER appuser # Пользователь, от имени которого будет запускаться приложение
32  ENTRYPOINT ["dotnet", "WebAppDemo.dll"]
```

Пример dockerbuild

```
1  # Stage Build
2  FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build # Используем SDK для сборки проекта
3  ARG BUILD_CONFIGURATION=Release # Переменная для конфигурации сборки
4  WORKDIR /src
5  COPY ["WebAppDemo/WebAppDemo.csproj", "WebAppDemo/"]
6  RUN dotnet restore "./WebAppDemo/WebAppDemo.csproj"
7  COPY . .
8  WORKDIR "/src/WebAppDemo"
9  RUN dotnet build "./WebAppDemo.csproj" -c $BUILD_CONFIGURATION -o /app/build # Собираем проект
10
11 RUN dotnet test /src/tests # Запускаем тесты
12
13 # Stage Publish
14 FROM build AS publish
15 ARG BUILD_CONFIGURATION=Release
16 RUN dotnet publish "./WebAppDemo.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false # Публикуем проект в виде бинарных файлов
17
18 # Stage Final
19 FROM mcr.microsoft.com/dotnet/aspnet:8.0-cbl-mariner-distroless AS final
20 WORKDIR /app
21 COPY --from=publish /app/publish . # Копируем бинарники проекта в образ
22 ARG UID=10001 # Переменная для UID пользователя
23 RUN adduser \
24     --disabled-password \
25     --gecos "" \
26     --home "/nonexistent" \
27     --shell "/sbin/nologin" \
28     --no-create-home \
29     --uid "${UID}" \
30     appuser
31 USER appuser # Пользователь, от имени которого будет запускаться приложение
32 ENTRYPOINT ["dotnet", "WebAppDemo.dll"]
```


Пример dockerbuild

```
1  # Stage Build
2  FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build # Используем SDK для сборки проекта
3  ARG BUILD_CONFIGURATION=Release # Переменная для конфигурации сборки
4  WORKDIR /src
5  COPY ["WebAppDemo/WebAppDemo.csproj", "WebAppDemo/"]
6  RUN dotnet restore "./WebAppDemo/WebAppDemo.csproj"
7  COPY . .
8  WORKDIR "/src/WebAppDemo"
9  RUN dotnet build "./WebAppDemo.csproj" -c $BUILD_CONFIGURATION -o /app/build # Собираем проект
10
11  RUN dotnet test /src/tests # Запускаем тесты
12
13  # Stage Publish
14  FROM build AS publish
15  ARG BUILD_CONFIGURATION=Release
16  RUN dotnet publish "./WebAppDemo.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false # Публикуем проект в виде бинарных файлов
17
18  # Stage Final
19  FROM mcr.microsoft.com/dotnet/aspnet:8.0-cbl-mariner-distroless AS final
20  WORKDIR /app
21  COPY --from=publish /app/publish . # Копируем бинарники проекта в образ
22  ARG UID=10001 # Переменная для UID пользователя
23  RUN adduser \
24      --disabled-password \
25      --gecos "" \
26      --home "/nonexistent" \
27      --shell "/sbin/nologin" \
28      --no-create-home \
29      --uid "${UID}" \
30      appuser
31  USER appuser # Пользователь, от имени которого будет запускаться приложение
32  ENTRYPOINT ["dotnet", "WebAppDemo.dll"]
--
```

Выбираем базовый образ для среды

Компонент	Dev	Prod/Pre-prod
Dev Tools, SDK	+	-
Root	+	-
Пакетный менеджер	+/-	-
Shell	+	-

Выбираем базовые образы для приложений на .NET

- Сравнение содержимого базовых образов
- Поиск уязвимостей в компонентах базового образа
- Сравнение быстродействия разных базовых образов



Настройка контейнера



Пользователь с минимальным набором прав

Это помогает минимизировать риски, связанные с эксплуатацией уязвимостей, так как приложение будет работать с ограниченными привилегиями.



Не использовать `docker.sock` и флаг `--privileged`

Проброс `docker.sock` внутрь контейнера и установка флага `--privileged` позволяют получить `root` доступ к хостовой ОС.

Контейнеры необходимо запускать в режиме запрета эскалации привилегий `--security-opt=no-new-privileges`.



Readonly монтирование директорий и томов

Предотвращаем случайные и несанкционированные изменения важных данных и конфигурации.

Настройка контейнера



Пользователь с минимальным набором прав

Это помогает минимизировать риски, связанные с эксплуатацией уязвимостей, так как приложение будет работать с ограниченными привилегиями.



Не использовать `docker.sock` и флаг `--privileged`

Проброс `docker.sock` внутрь контейнера и установка флага `--privileged` позволяют получить `root` доступ к хостовой ОС.

Контейнеры необходимо запускать в режиме запрета эскалации привилегий `--security-opt=no-new-privileges`.



Readonly монтирование директорий и томов

Предотвращаем случайные и несанкционированные изменения важных данных и конфигурации.

Настройка контейнера



Пользователь с минимальным набором прав

Это помогает минимизировать риски, связанные с эксплуатацией уязвимостей, так как приложение будет работать с ограниченными привилегиями.



Не использовать `docker.sock` и флаг `--privileged`

Проброс `docker.sock` внутрь контейнера и установка флага `--privileged` позволяют получить `root` доступ к хостовой ОС.

Контейнеры необходимо запускать в режиме запрета эскалации привилегий `--security-opt=no-new-privileges`.



Readonly монтирование директорий и томов

Предотвращаем случайные и несанкционированные изменения важных данных и конфигурации.

Создание пользователя с минимальным набором прав

На этапе сборки

```
ARG UID=10001 # Переменная для UID пользователя
RUN adduser \
  --disabled-password \
  --gecos "" \
  --home "/nonexistent" \
  --shell "/sbin/nologin" \
  --no-create-home \
  --uid "${UID}" \
  appuser
USER appuser # Пользователь, от имени которого будет запускаться приложение
```

Или на этапе запуска

```
docker run -d -p 8080:8080 -u 10001 --name webappdemo webappdemo:latest
```

Флаг `-u` в команде `docker run` используется для указания пользователя и группы, от имени которых будет запущен контейнер.

В docker-compose

```
version: '3'
services:
  app:
    image: webappdemo
    user: "${UID}:${GID}"
```

Запрет эскалации привилегий

setuid – позволяет запускать исполняемые файлы с правами владельца файла, а не текущего пользователя.

Пример использования:

```
✓ RUN cp /bin/bash /bin/setuidbash && chmod 4755 /bin/setuidbash # Устанавливаем права на запуск bash
```

4 – устанавливает бит setuid и позволяет запускать bash с правами владельца (root)

Запрет эскалации привилегий

setuid – позволяет запускать исполняемые файлы с правами владельца файла, а не текущего пользователя.

Пример использования:

✓ `RUN cp /bin/bash /bin/setuidbash && chmod 4755 /bin/setuidbash` # Устанавливаем права на запуск bash

4 – устанавливает бит setuid и позволяет запускать bash с правами владельца (root)

Эксплуатация:

```
$ whoami
appuser
$ /bin/setuidbash -p
setuidbash-5.2# whoami
root
setuidbash-5.2#
```

Запрет эскалации привилегий

setuid – позволяет запускать исполняемые файлы с правами владельца файла, а не текущего пользователя.

Пример использования:

```
✓ RUN cp /bin/bash /bin/setuidbash && chmod 4755 /bin/setuidbash # Устанавливаем права на запуск bash
```

4 – устанавливает бит setuid и позволяет запускать bash с правами владельца (root)

Эксплуатация:

```
$ whoami
appuser
$ /bin/setuidbash -p
setuidbash-5.2# whoami
root
setuidbash-5.2#
```

Запуск с ключом `--security-opt=no-new-privileges:true`

```
$ whoami
appuser
$ /bin/setuidbash -p
appuser@3faa5f9cac0c:/app$ whoami
appuser
appuser@3faa5f9cac0c:/app$
```

Readonly монтирование директорий и томов

Readonly режим

```
✓ # Запуск контейнера в ReadOnly режиме
  docker run --read-only webappdemo
  # Подключение tmpfs к контейнеру
  docker run --read-only --tmpfs /tmp webappdemo
```

Аналог в compose

```
version: '3.7'
services:
  app:
    image: webappdemo
    read_only: true
    tmpfs:
      - /tmp
```

Подключение тома в режиме readonly

```
version: '3.7'
services:
  app:
    image: webappdemo
    read_only: true
    volumes:
      - /var/data:/app/data:ro
```



Docker Security Cheat
Sheet

Работа с секретами

1

Инжект на этапе CI/CD

Внедрение секретов на этапе сборки в CI. Наименее безопасный вариант. Необходима повторная сборка при изменении секретов.

Работа с секретами

1

Инжект на этапе CI/CD

Внедрение секретов на этапе сборки в CI. Наименее безопасный вариант. Необходима повторная сборка при изменении секретов.

2

Инжект на этапе запуска контейнера

Внедрение секретов механизмом Docker/k8s Secrets/External Secret Storage. Оптимальный вариант для большинства сценариев. Необходим передеплой приложения при изменении секретов.

Работа с секретами

1

Инжект на этапе CI/CD

Внедрение секретов на этапе сборки в CI. Наименее безопасный вариант. Необходима повторная сборка при изменении секретов.

2

Инжект на этапе запуска контейнера

Внедрение секретов механизмом Docker/k8s Secrets/External Secret Storage. Оптимальный вариант для большинства сценариев. Необходим передеплой приложения при изменении секретов.

3

Использование внешнего хранилища

Приложения сами обращаются в хранилища за секретами. Максимальная защищенность секретов.

Внедрение секретов в CI

Пример копирование секретов при сборке образа:

```
✓ # Stage Final
FROM mcr.microsoft.com/dotnet/aspnet:8.0-cbl-mariner-distroless AS final
WORKDIR /app
COPY --from=publish /app/publish . # Копируем бинарники проекта в образ
COPY /secretsore/webappdemo/launchSettings.json /app/launchSettings.json # Копируем файл с секретами в образ
```

Плюсы:

- Простота настройки пайплайна сборки

Минусы:

- Необходимость повторного билда при изменении секретов
- Свой образ для каждой среды
- Возможность утечки секретов вместе с образом

Использование внешнего хранилища приложением

Использование в Docker Compose

```
version: '3.7'
services:
  app:
    image: webappdemo
    secrets:
      - webappdemo_secrets # Получение секретов из файла
    volumes:
      - ${SecretPath2}:/app/secrets:ro # Монтирование тома с секретами
    environment: # Передача секретов через переменные окружения
      - SECRET_1=${SECRET_1}
      - SECRET_2=${SECRET_2}
secrets:
  my_secret:
    file: ${SecretPath1}.txt
```

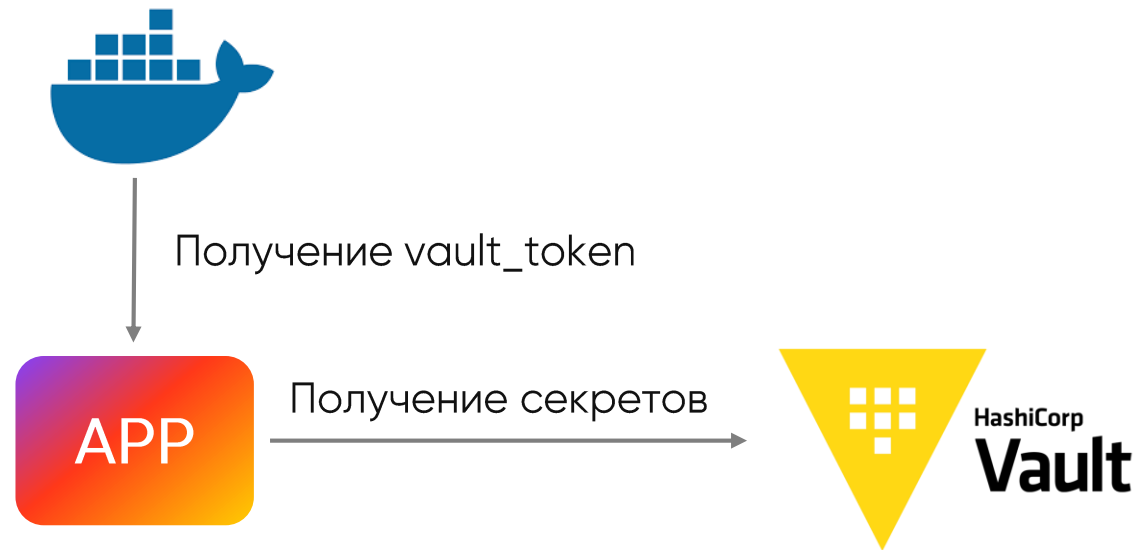
Плюсы:

- Образы не содержат секретов
- Можно использовать один compose для нескольких сред, передавая в него разные секреты

Минусы:

- Секреты хранятся в незашифрованных файлах на нодах или внешних ресурсах
- Сложность настройки deploy пайплайнов

Внедрение секретов в среде выполнения



Плюсы:

- Инфраструктура не знает секретов, риск утечки минимален
- Возможна реализация изменений секретов без перезапуска контейнера

Минусы:

- Необходимо наличие внешнего хранилища и правильная настройка
- Каждое приложение должно реализовывать подключение к хранилищу

Мониторинг и аудит

1

Мониторинг Docker и k8s

dockprom – решение, объединяющее Prometheus, Grafana, cAdvisor, NodeExporter AlertManager

<https://github.com/stefanprodan/dockprom>

k8s: набор приложений для мониторинга –
[prometheus-community/kube-prometheus-stack](https://github.com/prometheus-community/kube-prometheus-stack)

2

Аудит Docker

Выполнение CIS Benchmark для Docker

<https://github.com/docker/docker-bench-security>

3

Аудит k8s

Выполнение CIS Kubernetes Benchmark

<https://github.com/aquasecurity/kube-bench>

4

Kaspersky Container Security

Комплексный инструмент аудита k8s соответствиям стандартов, защита среды выполнения, сканирование на уязвимости контейнеров и образов

5

Aqua Security

Аналогичное решение, продвинутое и гибкие политики безопасности. Импортное решение.

Мониторинг и аудит

1

Мониторинг Docker и k8s

dockprom – решение, объединяющее Prometheus, Grafana, cAdvisor, NodeExporter AlertManager

<https://github.com/stefanprodan/dockprom>

k8s: набор приложений для мониторинга –
[prometheus-community/kube-prometheus-stack](https://github.com/prometheus-community/kube-prometheus-stack)

2

Аудит Docker

Выполнение CIS Benchmark для Docker

<https://github.com/docker/docker-bench-security>

3

Аудит k8s

Выполнение CIS Kubernetes Benchmark

<https://github.com/aquasecurity/kube-bench>

4

Kaspersky Container Security

Комплексный инструмент аудита k8s соответствиям стандартов, защита среды выполнения, сканирование на уязвимости контейнеров и образов

5

Aqua Security

Аналогичное решение, продвинутое и гибкие политики безопасности. Импортное решение.

Мониторинг и аудит

1

Мониторинг Docker и k8s

dockprom – решение, объединяющее Prometheus, Grafana, cAdvisor, NodeExporter AlertManager

<https://github.com/stefanprodan/dockprom>

k8s: набор приложений для мониторинга –
[prometheus-community/kube-prometheus-stack](https://github.com/prometheus-community/kube-prometheus-stack)

2

Аудит Docker

Выполнение CIS Benchmark для Docker

<https://github.com/docker/docker-bench-security>

3

Аудит k8s

Выполнение CIS Kubernetes Benchmark

<https://github.com/aquasecurity/kube-bench>

4

Kaspersky Container Security

Комплексный инструмент аудита k8s соответствиям стандартов, защита среды выполнения, сканирование на уязвимости контейнеров и образов

5

Aqua Security

Аналогичное решение, продвинутое и гибкие политики безопасности. Импортное решение.

Мониторинг и аудит

1

Мониторинг Docker и k8s

dockprom – решение, объединяющее Prometheus, Grafana, cAdvisor, NodeExporter AlertManager

<https://github.com/stefanprodan/dockprom>

k8s: набор приложений для мониторинга –
[prometheus-community/kube-prometheus-stack](https://github.com/prometheus-community/kube-prometheus-stack)

2

Аудит Docker

Выполнение CIS Benchmark для Docker

<https://github.com/docker/docker-bench-security>

3

Аудит k8s

Выполнение CIS Kubernetes Benchmark

<https://github.com/aquasecurity/kube-bench>

4

Kaspersky Container Security

Комплексный инструмент аудита k8s соответствиям стандартов, защита среды выполнения, сканирование на уязвимости контейнеров и образов

5

Aqua Security

Аналогичное решение, продвинутое и гибкие политики безопасности. Импортное решение.

Мониторинг и аудит

1

Мониторинг Docker и k8s

dockprom – решение, объединяющее Prometheus, Grafana, cAdvisor, NodeExporter AlertManager

<https://github.com/stefanprodan/dockprom>

k8s: набор приложений для мониторинга –
[prometheus-community/kube-prometheus-stack](https://github.com/prometheus-community/kube-prometheus-stack)

2

Аудит Docker

Выполнение CIS Benchmark для Docker

<https://github.com/docker/docker-bench-security>

3

Аудит k8s

Выполнение CIS Kubernetes Benchmark

<https://github.com/aquasecurity/kube-bench>

4

Kaspersky Container Security

Комплексный инструмент аудита k8s соответствиям стандартам, защита среды выполнения, сканирование на уязвимости контейнеров и образов

5

Aqua Security

Аналогичное решение, продвинутое и гибкие политики безопасности. Импортное решение.

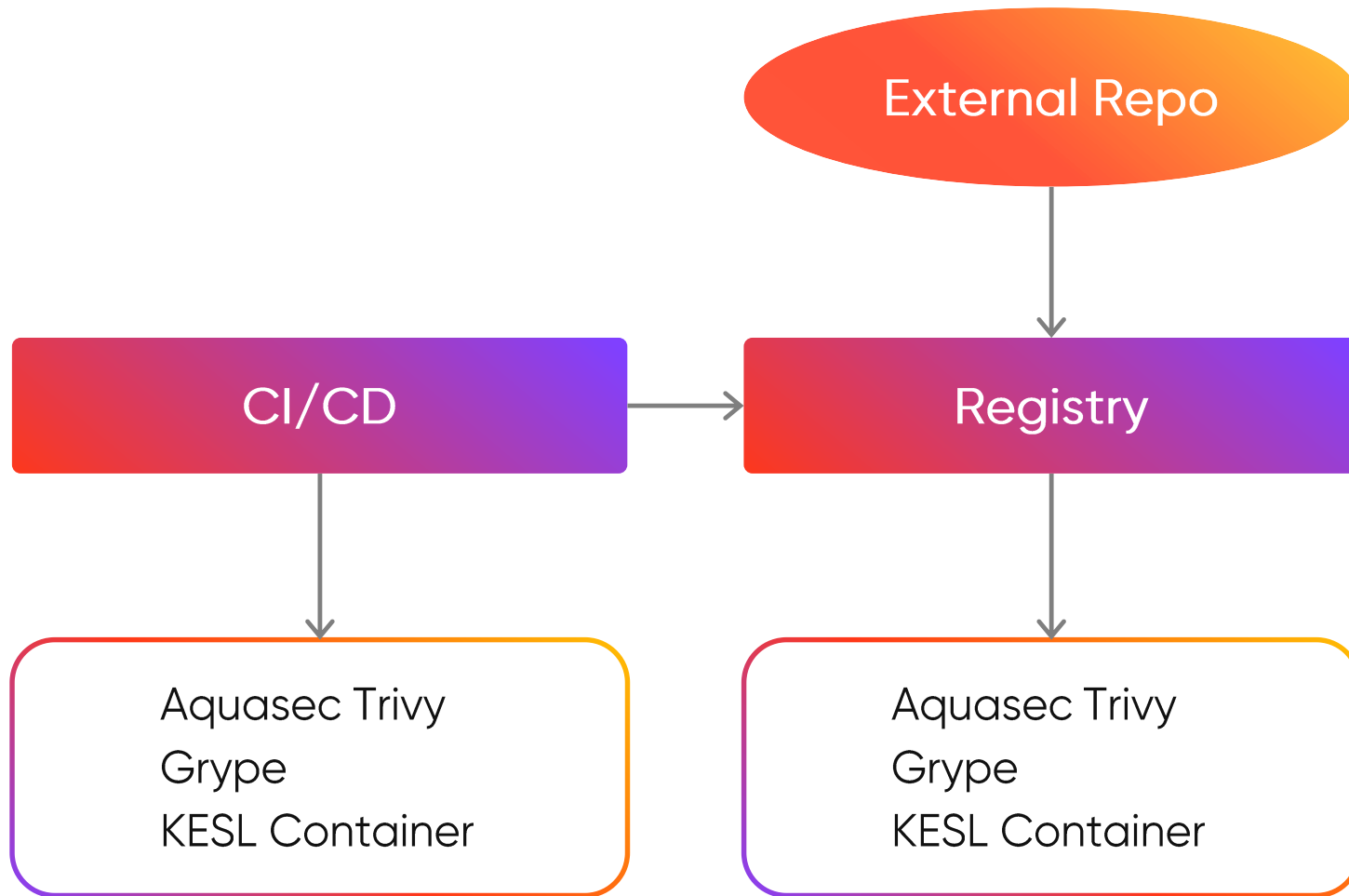
Сканирование образов

CI/CD

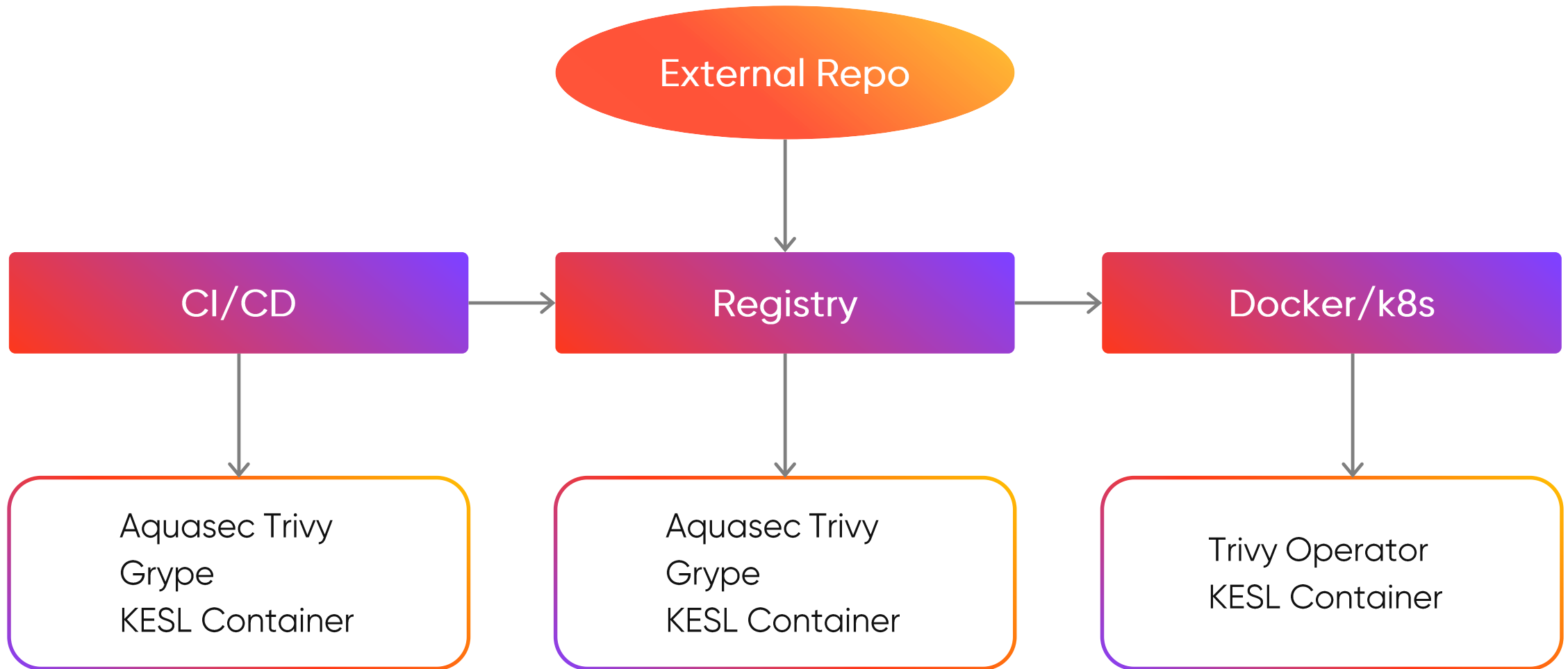
```
graph TD; A[CI/CD] --> B["Aquasec Trivy<br/>Grype<br/>KESL Container"]; style A fill:#ff6600,stroke:#ff6600,stroke-width:2px; style B fill:#fff,stroke:#ff6600,stroke-width:2px; linkStyle 0 stroke:#ff6600,stroke-width:2px; linkStyle 0 arrowhead:open
```

Aquasec Trivy
Grype
KESL Container

Сканирование образов



Сканирование образов



Выводы

1

Минимальный рантайм для приложения

Приложение работает в контейнере с минимальным набором компонентов

2

Настройка среды исполнения

Приложение работает с минимальным набором привилегий

3

Секреты

Контейнеры не содержат секретов и чувствительной информации

4

Настроен регулярный мониторинг и аудит

Инфраструктура содержит компоненты для специфичного мониторинга событий среды контейнеризации

5

Сканирование образов на предмет уязвимостей и вредоносов

Образы проверяются на этапе сборки, хранения и работы в среде контейнеризации.

Спасибо за внимание!



t.me/arahmanny



a.rahmanny@gmail.com