ТЕХНИЧЕСКОЕ РУКОВОДСТВО • ЧАСТЬ 3

KATA CONTAINERS: KUBERNETES, CI/CD И КАСТОМНОЕ ЯДРО

Интеграция через `kata-deploy`, расчет `PodOverhead`, облачные кейсы AKS, CI/CD изоляция и пошаговая сборка гостевого ядра

Глава 7. Интеграция Kata Containers с Kubernetes (`kata-deploy`)

7.1. Автоматизированное развертывание через DaemonSet-оператор

В production-кластерах Kubernetes ручная установка рантайма заменяется официальным оператором `kata-deploy`, который доставляет бинарники, ядра и конфигурации на узлы:

```
# Применение манифеста kata-deploy в кластере
kubectl apply -f https://raw.githubusercontent.com/kata-containers/kata-
containers/main/tools/packaging/kata-deploy/k8s/kata-deploy.yaml

# Проверка статуса развертывания подов демона:
kubectl get pods -n kube-system -l name=kata-deploy
```

7.2. Настройка `RuntimeClass` и учет оверхеда памяти (`PodOverhead`)

Каждая микро-ВМ Kata требует дополнительную память для гостевого ядра и гипервизора. Планировщик Kubernetes должен учитывать это через секцию `overhead` в объекте `RuntimeClass`:

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: kata-dragonball
handler: kata-dragonball
overhead:
  podFixed:
    cpu: "50m"
    memory: "120Mi"
```

Глава 8. Безопасные CI/CD-ранеры без privileged на хосте

Традиционные CI/CD системы (GitLab CI, GitHub Actions) для сборки образов через `docker build` или Docker-in-Docker часто требуют режим `privileged`, открывая угрозу побега. Использование Kata Containers с флагом `privileged_without_host_devices = true` в `containerd` запускает сборочный демон внутри изолированной микро-ВМ со своим ядром Linux, защищая хост.

Глава 12. Пошаговая настройка и сборка кастомного гостевого ядра Linux

Стандартное гостевое ядро Kata оптимизировано под минимальный размер. Если вашему приложению требуются подсистема **eBPF**, VPN-модуль **WireGuard** или специфические драйверы, необходимо собрать ядро самостоятельно.

12.1. Клонирование репозитория и подготовка окружения (osbuilder)

Инструменты сборки находятся в репозитории Kata Containers. Устанавливаем зависимости (`flex` , `bison` , `libelf-dev`) и настраиваем исходный код:

```
git clone https://github.com/kata-containers/kata-containers.git
cd kata-containers/tools/packaging/kernel
# Запуск первоначальной настройки и загрузки исходников ядра
./build-kernel.sh setup
```

12.2. Настройка параметров ядра через `make menuconfig`

После распаковки исходников в рабочей директории скрипта переходим к интерактивной настройке конфигурации:

```
cd kata-linux-*
make menuconfig
```

В появившемся графическом меню включаем необходимые опции:

- **Подсистема eBPF:** активируем `CONFIG_BPF=y` , `CONFIG_BPF_SYSCALL=y` , `CONFIG_NET_CLS_BPF=y` .
- **Модуль WireGuard:** включаем сетевой протокол VPN в разделе `Networking support -> WireGuard secure network tunnel` (`CONFIG_WIREGUARD=m` или `y`).
- **Оптимизация виртуализации:** проверяем наличие `CONFIG_KVM_GUEST=y` и `CONFIG_VIRTIO_FS=y` .

12.3. Сборка бинарника `vmlinux.container` и привязка к `configuration.toml`

Сохраняем конфигурацию и запускаем сборку с помощью официального скрипта:

```
cd ../
./build-kernel.sh build
sudo ./build-kernel.sh install
```

Скрипт автоматически скомпилирует ядро и положит готовый бинарник в `/opt/kata/share/kata-containers/vmlinux.container` . Остается прописать путь к нему в конфигурационном файле:

```
[hypervisor.dragonball]
kernel = "/opt/kata/share/kata-containers/vmlinux.container"
image = "/opt/kata/share/kata-containers/kata-containers.img"
```

Глава 11 (Дополнение). Низкоуровневая отладка через `kata-runtime exec`

Для диагностики сбоев внутри изолированной микро-ВМ используется прямой интерактивный вход:

```
sudo /opt/kata/bin/kata-runtime exec <SANDBOX_ID> sh
```