Введение

Контейнерная виртуализация — это технология, которая позволяет изолировать процессы и их окружение в рамках одной операционной системы. В отличие от классической виртуализации, где каждая виртуальная машина имеет свою операционную систему, контейнеры используют ядро хостовой системы, что делает их более легковесными и быстрыми. В Linux контейнеры реализуются с помощью таких технологий, как **namespaces** и **cgroups**.

Namespaces

Namespaces — это механизм изоляции ресурсов. Каждый namespace предоставляет изолированное представление ресурсов системы для процессов, находящихся внутри него. Основные типы namespaces:

- **PID namespace**: Изолирует идентификаторы процессов.
- **Mount namespace**: Изолирует точки монтирования.
- **UTS namespace**: Изолирует hostname и domainname.
- **User namespace**: Изолирует пользователей и группы.
- **Network namespace**: Изолирует сетевые интерфейсы.

Cgroups

Cgroups (Control Groups) — это механизм управления ресурсами. Он позволяет ограничивать и распределять ресурсы (CPU, память, дисковый ввод-вывод) между группами процессов.

Пример реализации контейнера

Рассмотрим пример программы на языке C, которая создает контейнер с использованием namespaces. Программа изолирует процесс, создавая для него новое окружение, включая корневую файловую систему и пространство имен пользователя.

```c
int main(int argc, char **argv) {
    ....

    if (pipe(params.fd) < 0)
        die("Failed to create pipe: %m");

    int clone_flags =
            SIGCHLD |
            CLONE_NEWUTS | CLONE_NEWUSER | CLONE_NEWNS | CLONE_NEWPID;
    int cmd_pid = clone(
        cmd_exec, cmd_stack + STACKSIZE, clone_flags, &params);

    if (cmd_pid < 0)
        die("Failed to clone: %m\n");

    int pipe = params.fd[1];

    prepare_userns(cmd_pid);

    if (write(pipe, "OK", 2) != 2)
        die("Failed to write to pipe: %m");
    if (close(pipe))
        die("Failed to close pipe: %m");

    if (waitpid(cmd_pid, NULL, 0) == -1)
        die("Failed to wait pid %d: %m\n", cmd_pid);

    return 0;
}
```

```c
int cmd_exec(void *arg) {
    if (prctl(PR_SET_PDEATHSIG, SIGKILL))
        die("cannot PR_SET_PDEATHSIG for child process: %m\n");

    ...

    await_setup(params->fd[0]);
    prepare_mntns("../rootfs");

    if (setgid(0) == -1)
        die("Failed to setgid: %m\n");
    if (setuid(0) == -1)
        die("Failed to setuid: %m\n");

    char **argv = params->argv;
```

```
    char *cmd = argv[0];

    if (execvp(cmd, argv) == -1)
        die("Failed to exec %s: %m\n", cmd);
    return 1;
}

void prepare_userns(int pid) {
    ...
    sprintf(path, "/proc/%d/uid_map", pid);
    // uids mapping: inside_id outside_id length
    sprintf(line, "0 %d 1\n", uid);
    write_file(path, line);

    sprintf(path, "/proc/%d/setgroups", pid);
    // запрещаем изменять процессу свои группы
    sprintf(line, "deny");
    write_file(path, line);

    sprintf(path, "/proc/%d/gid_map", pid);
    // gids mapping
    sprintf(line, "0 %d 1\n", uid);
    write_file(path, line);
}

void prepare_mntns(char *rootfs) {
    const char *mnt = rootfs;

    if (mount(rootfs, mnt, "ext4", MS_BIND, ""))
        die("Failed to mount %s at %s: %m\n", rootfs, mnt);

    if (chdir(mnt))
        die("Failed to chdir to rootfs mounted at %s: %m\n", mnt);

    const char *put_old = ".put_old";
    if (mkdir(put_old, 0777) && errno != EEXIST)
        die("Failed to mkdir put_old %s: %m\n", put_old);

    if (syscall(SYS_pivot_root, ".", put_old))
        die("Failed to pivot_root from %s to %s: %m\n", rootfs, put_old);

    if (chdir("/"))
        die("Failed to chdir to new root: %m\n");

    if (umount2(put_old, MNT_DETACH))
```

```
        die("Failed to umount put_old %s: %m\n", put_old);
}
```

Дальнейшие работы

1. Рассмотреть cgroups
2. Рассмотреть seccomps
3. Добавить network namespace (slirp4netns для rootless выполнения)
4. Изучить интеграцию overlayfs
5. Рассмотреть интеграцию apparmor/selinux
6. Рассмотреть интеграцию: oci container runtime -> conmon -> cri