个人信息

- 姓名： 廖翔
- 学号： U201915116
- 班级：CS1906

所选论文

- title={**Container-Based Customization Approach for Mobile Environments on Clouds**}
- author={Jiahuan Hu and Song Wu and Hai Jin and Hanhua Chen}
- booktitle={GPC}
- year={2018}

读后感

如今，利用云的弹性资源为移动应用提供服务的云平台正变得越来越流行。在构建移动云平台（MCP）时，最重要的一点是为移动应用提供一个执行环境，例如，Android移动操作系统（OS）。为了在云上建立安卓环境，已经有许多努力，如安卓虚拟机和安卓容器。但是这两种方法都有其各自的缺点。在谈这个话题前，我打算先谈一谈这两个概念。

容器中最有名的当属于Docker，而Docker与虚拟机的区别如下：

- 虚拟机是通过管理系统(Hypervisor)来模拟出CPU，内存，网络等硬件，然后在这些模拟的硬件上创建内核和操作系统，这样做的好处是虚拟机有自己的内核和操作系统,用户是无法直接使用主机上的操作系统和硬件资源。因此虚拟机也对隔离性和安全性有着更好的保证
- Docker容器是通过Linux的Namespace来进行资源隔离的，然后利用cgroups来进行资源的限制，最终实现了容器之间互不影响，由于容器的隔离仅仅是依靠内核来提供，所以容器的隔离性弱于虚拟机。

我重点提到了隔离性这个词，因为本文章提出的AndroidX就是为了解决容器的安全问题，那么Docker有哪些问题呢

Docker自身安全

- Docker作为一款容器引擎，本身也会存在一些安全漏洞，CVE目前已经记录了多项与Docker相关的安全漏洞，主要有权限提升，信息泄漏几类安全问题

镜像安全

- 由于Docker是需要镜像来启动的，因此镜像的安全直接影响到容器的安全
- 镜像软件存在安全漏洞：由于容器需要下载一些软件包，如果软件包本身就有问题，那么就会有别人侵入容器，影响其它容器或者主机的安全
- 仓库漏洞：无论是Docker的官方仓库还是自己的私有仓库，都有可能被别人侵入，然后修改你的镜像文件，当我们使用镜像时，就会成为攻击者的目标
- 用户程序漏洞：用户自己构建的软件包可能存在漏洞被别人植入恶意脚本，这样会导致容器或者主机的安全

Linux内核隔离性不够

- 尽管目前Namespace已经提供了很多种资源隔离类型，但是仍有部分关键内容没有被隔离，其中包括一些系统的关键性目录(/sys,/proc)等，这些关键性的目录可能会泄漏主机一些关键性的信息
- 而仅仅依靠Namespace的隔离是远远不够的，因为一旦内核的Namespce被攻破，使用者就可以直接获取主机的超级权限，从而影响主机安全

所有容器共享主机内核

- 由于同一主机上的所有容器共享主机内核，所以攻击者可以利用一些特殊手段使主机内核崩溃，进而导致主机宕机影响主机上的其它服务

显然，Docker有这些问题也不可能坐以待毙，因此也有许多解决这些问题的方案，例如

- Docker自身安全性改进：随着Docker的不断改进，在安全性方面已经作出了非常大的改进，目前Docker在默认配置和默认行为下是足够安全的

- 保障镜像自身安全：我们可以在私有仓库安装镜像安全扫描组件，一旦发现镜像有漏洞，就及时通知用户阻止镜像的构建和发布

- 加强内核安全和管理：由于容器仅仅依靠内核来提供资源隔离，所以我们要及时升级内核漏洞，使用Capabilities划分权限，使用安全加固组件，资源限制

- 使用安全容器：容器有着轻便快速启动的优点，虚拟机有着安全隔离的优点，有没有一种技术可以兼顾两者的优点，做 到既轻量又安全呢?答案是有，那就是安全容器。安全容器是相较于普通容器的，安全容器与普通容器的主要区别在于，安 全容器中的每个容器都运行在一个单独的微型虚拟中，拥有独立的操作系统和内核，并且有虚拟化层 的安全隔离。安全容器目前推荐的技术方案是 Kata Containers，Kata Container 并不包含一个完整的操作系统，只 有一个精简版的 Guest Kernel 运行着容器本身的应用，并且通过减少不必要的内存，尽量共享可以共 享的内存来进一步减少内存的开销。另外，Kata Container 实现了 OCI 规范，可以直接使用 Docker 的 镜像启动 Kata 容器，具有开销更小、秒级启动、安全隔离等许多优点。

而本文中的AndroidX有一部分灵感就是来自于Kata Container，但是AndroidX是专门为云上各种频繁变化的场景设计的。同时也不得不提一下Unikernel,要了解什么是 Unikernel，首先需要了解什么是 kernel，kernel 是操作系统中的一个概念。应用要运行起来，是肯定要跟硬件打交道的，但是如果让应用都直接操作硬件，那一定是一场灾难。那内核就是在应用与硬件中间的一层抽象，内核提供了对底层硬件的抽象，比如把硬盘抽象成了文件，通过文件系统进行管理。传统的内核会将所有的硬件抽象都实现在其中，其中的代表就是 Linux，这样的内核被称为宏内核（Monolithic Kernel)。在宏内核中，所有的模块，诸如进程管理，内存管理，文件系统等等都是实现在内核中的。这样虽然不存在通信问题，但是任何一个模块的 bug 会使得整个内核崩溃。

于是学者们提出了微内核（Micro Kernel）的概念，在内核中只保留必要的模块，比如 IPC，内存管理，CPU调度等等。而其他，诸如文件系统，网络IO等等，都放在用户态来实现。这样会使得内核不那么容易崩溃，而且内核需要的内存小了。但是由于模块间的通信需要以 IPC 的方式进行，因此有一定的 overhead，效率不如很莽的宏内核。

那后来又有了混合内核（Hybrid Kernel)，把一部分不常使用的内核模块，或者是原本需要的时间就很长，因此 IPC 的 overhead 看起来就不是那么夸张的功能，移出内核，而其他的就会放在内核里。再后来还有 Exokernel等，在这里就不赘述了。

而Unikernel 是专用的，单地址空间的，使用 library OS 构建出来的镜像,其最大的卖点就是在，没有用户空间与内核空间之分，只有一个连续的地址空间。这样使得 Unikernel 中只能运行一个应用，而且对于运行的应用而言，没有硬件抽象可言，所有的逻辑，包括应用逻辑和操作硬件的逻辑，都在一个地址空间中。

Unikernel 镜像都很小，由 MirageOS实现的一个 DNS server 才 184KB，实现的一个 web server 674 KB，小到恐怖的程度。

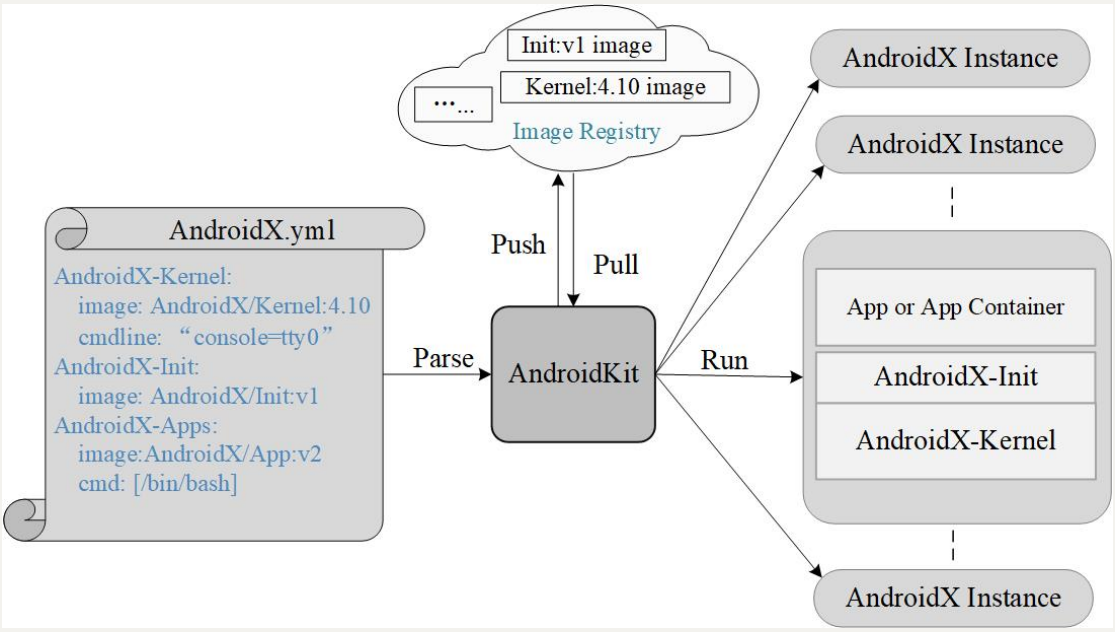
然后就是快，启动很快。因为镜像都很小，所以起停都在毫秒级别，比传统的 kernel 要快多了。

最后是安全，一般来讲，小的东西相对而言比较安全。Unikernel 中没有 Shell 可用，没有密码文件，没有多余的设备驱动，这使得 Unikernel 更加安全。

在我看来Unikernel是另一种形式上的容器。在一个 Unikernel 中，只能运行一个应用，这与容器的哲学不谋而合。但现在容器最吸引人的特性并不是它的便捷，而是在它的分发。Docker 让我们看到了，原来应用的分发可以这么无痛。而 Unikernel 与容器相比，虽然可以做的更小更安全，而且也不需要有 Docker Daemon 这样的后台程序存在，甚至不需要 Host OS，或者 Hypervisor，但是它一是与传统的软件过程有较大的出入，二是在分发等等方面不能做到像容器那样方便。所以它目前肯定不会成为主流的应用分发方式，还需要进一步探索。

AndroidX

说了这么多，就是为了引出这篇论文所提出的基于容器个性化云端执行环境的方法Android X，相对应的还有一个工具包AndroidKit来帮助用户完善体验。



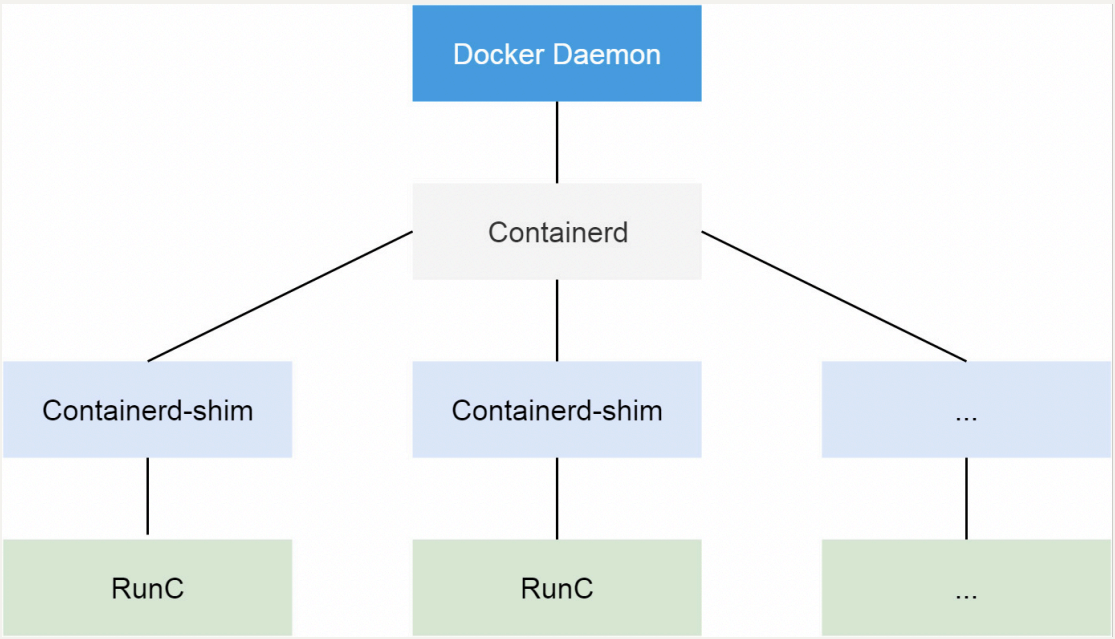
这张图即为整篇论文的核心，我们可以清楚地看到，AndroidX有两个核心组件。

一个是带有命令行界面的工具集，称为AndroidKit，用于解析输入的yaml配置文件并生成Docker镜像。

另一个是一个与管理程序无关的runtime，它能够用一个新的虚拟机实例启动镜像。

在这个图中的AndroidX实例的内部，一个被称为AndroidX-Kernel的极简的Android Kernel直接由管理程序启动。AndroidX-Kernel采用了一个微小的Android初始化服务，称为AndroidX-Init，从主机加载图像，然后启动它们。通过在独立的虚拟机实例和内核空间中包含应用程序，AndroidXs能够提供比容器更出色的工作负载隔离，以及像虚拟机一样的安全优势，这非常适用于多用户云环境。

显然，AndroidX克服了容器饱受诟病的安全性缺点，同时又不像虚拟机那么的重，有很广泛的应用场景。此外，文章还提到了containerd, runC这些Docker重要组件,那么首先要看到Docker的架构



当我们要创建一个容器的时候，现在 Docker Daemon 并不能直接帮我们创建了，而是请求 `containerd` 来创建一个容器，containerd 收到请求后，也并不会直接去操作容器，而是创建一个叫做 `containerd-shim` 的进程，让这个进程去操作容器，我们指定容器进程是需要一个父进程来做状态收集、维持 stdin 等 fd 打开等工作的，假如这个父进程就是 containerd，那如果 containerd 挂掉的话，整个宿主机上所有的容器都得退出了，而引入 `containerd-shim` 这个垫片就可以来规避这个问题了。

然后创建容器需要做一些 namespaces 和 cgroups 的配置，以及挂载 root 文件系统等操作，这些操作其实已经有了标准的规范，那就是 OCI（开放容器标准），`runc` 就是它的一个参考实现，这个标准其实就是一个文档，主要规定了容器镜像的结构、以及容器需要接收哪些操作指令，比如 create、start、stop、delete 等这些命令。`runc` 就可以按照这个 OCI 文档来创建一个符合规范的容器，既然是标准肯定就有其他 OCI 实现，比如 Kata、gVisor 这些容器运行时都是符合 OCI 标准的。

所以真正启动容器是通过 `containerd-shim` 去调用 `runc` 来启动容器的，`runc` 启动完容器后本身会直接退出，`containerd-shim` 则会成为容器进程的父进程，负责收集容器进程的状态，上报给 containerd，并在容器中 pid 为 1 的进程退出后接管容器中的子进程进行清理，确保不会出现僵尸进程。

而 Docker 将容器操作都迁移到 `containerd` 中去是因为当前做 Swarm，想要进军 PaaS 市场，做了这个架构切分，让 Docker Daemon 专门去负责上层的封装编排，当然后面的结果我们知道 Swarm 在 Kubernetes 面前是惨败，然后 Docker 公司就把 `containerd` 项目捐献给了 CNCF 基金会，这个也是现在的 Docker 架构。

最后，我也跟着https://github.com/CGCL-codes/AndroidX这个github地址尝试去follow整个instruction，最重要的就是其中的yaml文件

```
kernel:
  image: zjsyhjh/androidx-kernel:3569d0dc175cfd163380ddbccee44784
  cmdline: "console=tty0"
init:
  - zjsyhjh/androidx-init:417cf515633e8af3f819d155cea87f2e
trust:
  org:
    - zjsyhjh
```

这是官方仓库的一个yaml文件模版,具体到其中的对应项如下

- `AndroidX-Kernel` specifies a kernel Docker image, containing a kernel and a filesystem tarball, eg containing modules.
- `AndroidX-Init` is the base `init` process Docker image, which is unpacked as the base system, containing `init`, `containerd`, `runc` and et al.

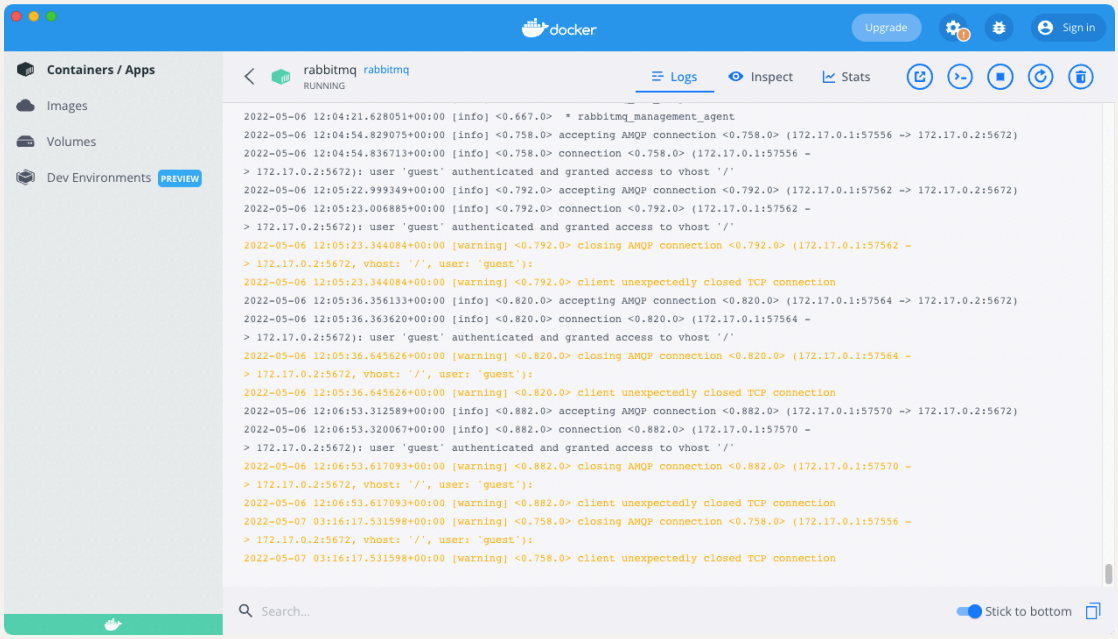
那么如何保证镜像安全了，这就需要DCT了，总体来说，Docker 镜像的发布者可以在将镜像推送到库中时对其进行签名。使用者可以在拉取镜像时进行校验，或进行构建或运行等操作。长话短说，DCT 确保使用者能够得到他们想要的镜像。DCT 实现的是客户端的签名和验证，意味着由 Docker 客户端执行它们。显然，类似这样的密码机制，对于确保在互联网上拉取和推送的软件的可信性是非常重要的，其在整个技术栈的各个层次，以及软件交付流水线的各个环节都在发挥越来越重要的作用。在不久的将来，这种加密信任机制将有望在交付链的各个方面发挥作用。

除此之外，官方仓库还提供了许多script来帮助用户理解AndroidX的运行原理。

总的来说，虽然我并不太熟悉安卓应用，但我因为在大厂作为测试实习生，还是能够理解文章中所提到的许多痛点。作为测试，需要把应用在不同的环境下都在测试一遍，例如不同的Android版本，不同的手机型号，不同的屏幕，不同的配置等等，在公司的许多测试都是需要真机，也就是需要去采购机器回来在机器上面实测，显然花费还是有点高，但是如果能够通过直接构建运行环境的方式来提供一个环境，虽然说不能完全满足测试需求，但是在一定程度上也能够节约成本。

课程体会

整个课程下来，老师的讲授内容主要分为两类，一类是云计算，另一类就是容器技术。云计算显然是现在非常热门的一个话题，对于一些用户的需求，云服务器显然是非常具有优势的。而容器技术，显然是为了克服虚拟机的许多困难诞生的，但容器并不能够完全的取代虚拟机，在有些应用场景下，还是需要用户自己去做一个取舍。我个人用Docker比较多，比如在自己的机器上要实现一个能够通讯的服务器，就可以通过Docker去创建对应的容器，例如



这就是我之前学习微服务所创建的一个容器，整个过程十分快速，并且不用担心破坏到本机的一些配置等等，非常方便。

总的来说，云计算和容器都有极大的发展前景，在学术界和工业界都会有极为广泛的应用。

后文为我阅读论文时所做的一些笔记。

笔记

摘要

提供应用执行环境的重要性。

大部分的解决方案是安卓虚拟机和安卓容器。而个性化的安卓执行环境一直被忽略，并且也不利于升级和维护。

这篇论文提出了一个在云端个性化安卓环境的方法，并且开发出了AndroidKit来自动化地产生特定的安卓系统组件。通过AndroidKit，能够启动全新的安卓虚拟机实例AndroidXs。而这些实例就是通过前面的组件来组成的，因此非常的方便。

简介

首先介绍mobile cloud在当前环境的重要性。包括一些软件测试都是通过云端完成的。

建立这样一个平台的最主要的挑战就是建立一个移动机执行环境。作者以Android OS为例，虽然是基于Linux，但是两系统run GUI的方式是不一样的。并且安卓存在一些kernel feature是linux没有的。因此，在MCP上部署安卓应用是具有挑战性的，因为大部分MCP都是基于GNU/LINUX。

主流方案就是用虚拟机。虽然虚拟机避免了许多硬件上的问题，但是它是极其消耗资源的，因为它完全复制了一个OS并且需要高虚拟化。作为对比，比如Docker，就非常的轻虚拟化。虽然当前市场上已经有了许多例子，但是他们面临着同样一个问题就是只有一少部分安卓应用能够正常地运行。

对于经常变化的使用场景，个性化的model是非常需要的。当前的一些解决方案同样不能满足广泛的需求。目前的解决方案都是通过个性化一个系统，这样会导致后期的升级与维护非常难做，也无所满足如今的大量需求。

基于上述的分析，本篇论文主要解决前文提到的两个问题。受前期个性化OS的启发，我们提供了一个个性化的安卓执行环境AndroidX。AndroidX由系统映像构建而来然后与他们一起运行。从文中的图例可以看到，AndoirdX的两个实例分别用了Kernel-1和Kernel-2镜像，但是他们共同使用Init-1镜像。这种联合最大化地利用了镜像，同时简化了维护和升级。通过Docker镜像的许多优点，AndroidX能够轻松的为各种场景个性化。除此之外，AndroidX将硬件系统的强制独立与虚拟机的兼容性和容器的灵活性联合在一起。

总结来说，这篇文章的两个主要贡献为：

- 我们提供了个性化安卓系统的规范式方法。这个方法提供了一个基于容器来构建安卓系统镜像的解决方案。跟随这个方法的指导，我们提供了一个自动个性化构建的工具辅助包AndroidKit，对于当前的广泛需求更加合适
- 我们提供了一个原型实现叫AndroidX。

研究动机

首先，构建MCP时，MCP需要能够提供一个能够运行各个应用的缓降。个性化的安卓系统有需求是因为不同场景有它们各自的运行环境。例如，因为灵活性和交互性是计算分载成功的关键因素，个性化的安卓系统应该尽可能地轻，只保留计算分载所需要的功能部分。作为对比，基于云的移动计算，需要MCP去提供许多版本的移动操作系统来供测试使用。因此，对于云上各种各样的场景，个性化的安卓系统有很大需求。

其次，当前的一些个性化安卓系统的努力只集中于特殊硬件的设备或者特殊驱动的程序。这些方法有很多问题，例如对于不同的设备和平台需要个性化整个系统镜像，导致对操作系统的镜像利用率很低。并且这些还需要开发者对于安卓系统非常熟悉，但这其实是不现实的。另外，因为缺乏规范化的验证手段，对于这个操作系统组件很难保证可维护性和安全性。

最后，安卓虚拟机过高的虚拟化和缓慢的启动速度，导致它很难满足当前云上一些需要低时延，高互动性和灵活性的场景。虽然安卓容器克服了上述的一些困难，但是对于云上多租户的部署是不合适的。因此，需要一个像虚拟机一个独立而又兼容，同时也像容器一样足够灵活的个性化执行环境。

系统设计

总览

这个部分，我们介绍AndroidX，一个可以个性化的安卓执行环境来在云上运行安卓应用。同时还有AndroidKit，一个工具辅助来帮助我们在特殊的应用之间创建安卓虚拟机镜像。

AndroidX有如下几个主要目标：

- 个性化组件：操作系统组件的个性化是非常重要的，而AndroidX就是用来个性化操作系统
- 易维护，易迭代
- 不变的基础设施

我们使用Linux平台作为运行和测试环境。AndroidX有两个核心组成，一个是AdnroidKit，用来解析yaml配置文件并生成Docker镜像，另一个是能够通过一个虚拟机实例来部署这个Docker镜像。

有四个困难是我们要克服的：

- 将Android系统的个性化过程尽可能的简化，并且使迭代和维护尽可能得快
- 对于Docker容器，要兼容OCI
- 尽可能地减少运行开支
- 既有容器的速度，又有虚拟机的独立性

AndroidKit

我们的目标是实现简单的工具化和维护。基于此，我们提供了AndoirdKit这一个工具来进行镜像建立。这个工具包括一个自动构建系统，能够对于特殊的应用创建一个尽可能简单的Docker镜像。总的来说，它能够根据Dockerfile来创建镜像。我们可以看到这个工具使用一个yaml配置文件模版作为输入，这个文件包括特定的需要个性化的系统。

因为我们最大程度地利用了Docker容器，允许用户把一个应用所需要的所有包都打包起来并作为一个唯一的包，对于个性化一个Android系统，我们能够确保生成镜像是最新且完整的。多亏Docker提供的丰富的工具，用户对于另一个应用的需求去添加和更新一个已存在的组成是非常容易的。

根据yaml配置文件构建完镜像后，AndroidKit能够运行这个镜像通过一个叫AndroidX的新虚拟机镜像。深入到AndoirdX实例内部，一个最小化的Android内核AndroidX-Kernel通过管理程序直接启动，这个内核通过部署Android初始化服务AndroidX-Init去加载Docker镜像。通过将不同应用分在不同的虚拟机里，AndroidX能够比容器提供更多的独立性，和安全性，对于云上的多租户环境更加实用。

AndroidX

我们的另一个目标是能够部署Docker镜像在一个新的Android虚拟机上，也就是AndroidX。通过用户自己定义的Docker镜像，AndroidKit能够收集并加入到启动镜像中，之后它们能通过管理系统直接部署。AndroidX通过益处Guest OS的中间层来确保基础设施不变。镜像就像初始ram文件系统一样运行，能够直接地在外部升级系统组件。这使得AndroidX不可变，并且可以通过加-driver参数来增加存储。

AndroidX的个性化过程通过AndroidX-Kernel，AndroidX-Init， Adnroid-Apps来组件，这些可以在yaml配置文件中看到。

AndroidX-Kernel

这部分定义了内核配置。因为容器与host分享内核，因此如果用户想要增加一个新的内核feature，就必须得更改host kernel并且重新编译，对于不熟悉内核的程序员来说，这项工作非常困难。除此之外，对于云端多用户环境，因为容器之间的弱隔离性。这样的操作也并不合适。相反的，AndroidX有自己独立的内核，并且提供了一系列有用的工具来简化个性化内核的过程。

为了构建AndroidX-Kernel，我们将内核配置文件根据选项分成两部分。一部分是最基本的，另一部分是用户自己选择的配置项，根据不同硬件平台和应用要求可以更换。当创建一个个性化的镜像时，AndroidKit能够通过我们提供的merge_config脚本来产生一个最简单的配置文件来构建内核。这使得AndroidKit能够创建效率极高的镜像。

AndroidX-Init

这部分包括一些初始化所需要的，例如临时文件系统。这部分和Kernel能够直接在管理系统上启动。不像其他基于Linux的系统，Android使用自己的初始化程序，通过init.rc这一脚本来启动许多必须成功或功能。

原始的runC程序不能部署应用容器因为pivot_root系统调用，在ramfs和tmpfs上行不通。AndroidX率先的创建tmpfs作为根文件系统，然后使用switch_root系统调用来更换根文件系统。因为这一切都在启动Android初始化程序之前，runC能够正确运行。除此之外，runC提供了一个原生的Go实现来创建容器。通过我们的努力，AndroidX-Init能够带来工业标准的容器环境，并且用runC来运行应用容器。

AndroidX-Apps

这部分展示了一系列运行应用和服务的镜像。它包括特定的应用和服务需要在AndroidX启动时部署。AndroidX-Apps的最终目标就是特定的应用和服务能够直接启动。因为AndroidX-Init本就原生支持运行Android应用，所以不需要其他的工作。

因为我们希望一个AndroidX实例能够尽可能多地共享系统组件，我们使用安卓系统的只读特性，可以通过-drive参数来与其他实例共享AndroidX实例。得益于此，AndroidX实例所使用的硬盘空间减少。因为我们基于不可变基础设施来设计AndroidX，AndroidX-Apps可以通过使用SD card来存储，并且可以在启动时被识别。