CloudIDE——商业模式和服务技术

背景介绍

在云计算浪潮和无服务器Serveless的大趋势下，如何优化开发者的体验?

在过去的五年里，软件开发的生命周期发生了巨大的变化。基础设施即代码的概念以及DevOps文化的发展对软件开发产生了强烈的影响。开源运动也在影响IDE的未来方面发挥了关键作用。不断变化的动态迫使开发工具的提供者回到白板前，从头开始重新设计IDE。

以下是改变IDE面貌的几个趋势。

- 多语言化。一个应用程序只用一种语言针对单一技术栈进行编码的日子已经一去不复返了。在很长一段时间里，开发人员只在类似Java或.NET的单一技术领域内开发应用程序。在今天的世界里，开发人员被要求至少懂得三种语言才能在工作中取得成效。云和微服务使企业能够为一个应用程序选择最佳的语言和工具。
- 跨平台。在多语言之后，下一个重大变化是跨平台的兼容性。每一行代码都应该在各种桌面环境、浏览器、移动设备、平板电脑、甚至可穿戴设备上无缝运行。现有的工具不足以开发跨平台的应用程序。
- 协作。开源运动与DevOps文化相结合，使协作开发成为SDLC的重要方面。源代码管理（SCM）已经成为开发中不可或缺的一部分。Git和Github的出现使合作开发变得更加简单。

- 构建管理。持续集成和部署（CI/CD）是DevOps的一个关键支柱。集成开发环境支持一键提交到SCM，这将立即触发一个构建过程，进而启动一系列的自动测试。这种连续的反馈回路缩小了软件发货所需的周期。集成开发环境需要有效地参与到这个过程中。
- 可扩展性。在当前SDLC的背景下，IDE不仅仅是智能编辑器。它们需要支持新兴语言、脚本环境、声明性标记语言和第三方集成的插件。集成开发环境正在逐渐成熟，成为自身的平台。
- 微服务。基于微服务的应用程序是由一组模块化的服务组装而成的。从单体到微服务的转变正在影响着开发者的思维过程。他们需要在开发和测试阶段访问整个系统堆栈。
- 多目标部署。开发人员编写的代码会被送到公共云、裸机服务器、虚拟机、容器中，甚至被编译成一个独立的操作系统（Unikernel）。当然，针对X86、AMD64、ARM和其他架构的传统挑战仍然存在。亚马逊对Cloud9 IDE的收购表明了社区的发展前景。一个提供计算、存储和网络服务的IaaS供应商转变为一个最强大的平台公司，这出乎很多人的意料，但它正揭示着IDE平台的发展方向。

需求分析

配置环境是一项沉重而重复的工作

一般的开发工作流中，我们会建立一套本地环境，包括顺手的 IDE 和整套本地工具，但这种本地开发模式存在一些问题：

- 开发机性能要求高：冷编译一次 40 分钟。
- 开发环境配置复杂：工具环境能够通过容器技术或一系列版本管理工具（如 nvm）解决，但网络、安全等环境就不那么容易配置了。
- 依赖特定设备：休假可以，但是带上电脑，24 小时 On Call。
- 巨型代码库的管理难题：巨型代码库切换 Git 分支，动辄耗费半小时。

于是，**远程开发**的理念应运而生，连接远程测试服务器，直接在服务器环境完成日常开发工作，免去本地重建并维护一套测试环境的成本。

现有的远程开发模式下，工程师大多通过终端交互工具连接远程机器，并通过 vim、naro 等文本编辑器来开发。而这些编辑器通常对项目文件管理、运行任务、调试器、智能提示/补全等基础功能的支持不那么友好，并不能像本地 IDE 一样提供舒适的开发环境。开发体验下降的同时，也限制了开发效率。

环境不可共享导致大量人力的浪费

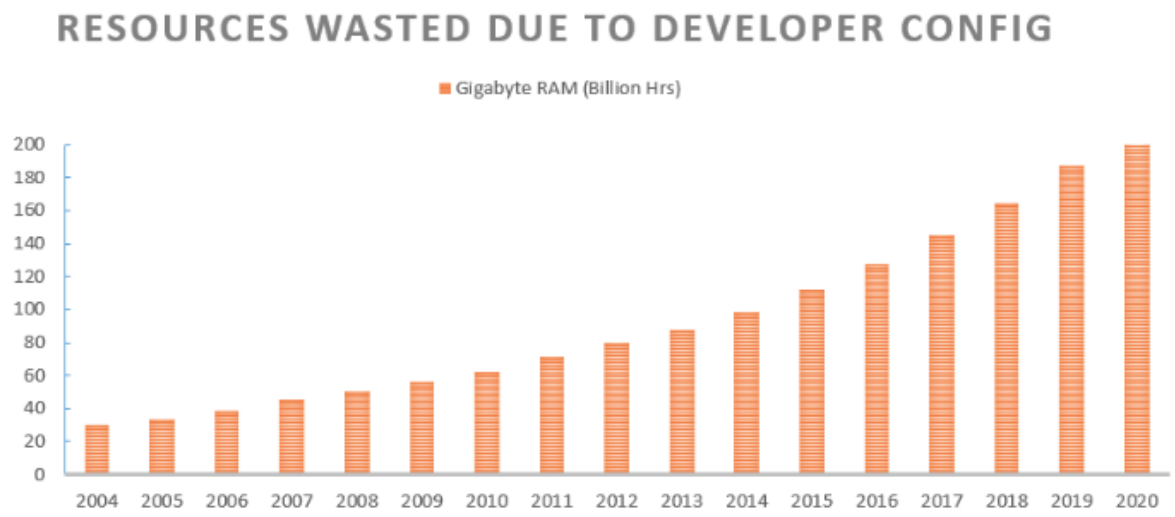
- 开发人员：希望预包装的环境可以随时编码，不需要安装。他们希望减少冷启动时间，将更多的精力放在研究写出更优雅的程序上。
- 团队：希望减少甚至消除浪费在环境配置和维护上的程序员时间。他们希望在采用持续交付&集成、开放源码、微服务和容器等等技术栈的同时，不必被庞大系统的配置之复杂所拖累。

开发环境的重复配置导致了开发人员的生产力损失，浪费了大量的开发时间；同时，由于机器的闲置，计算资源也常常不能被充分地利用。

统计数据称，在2014年：

- 世界上有1850万名开发人员
- 他们在配置上花费了157亿个小时
- 浪费了**99B Gigabyte hours**

如今开发人员总数相比2014年增长了27%，时间的浪费增长了一倍多，达到了**213B Gigabyte hours**。



1. Input: IDC 2014 Report w/ # of hobbyist / professional devs world wide through 2020.
2. Input: BLS statistics on growth of develoeprs.
3. Elastic cloud study: link to study on devs spend 13 hours / week configuring.
4. Average RAM per dev desktop: http://www.quepublishing.com/articles/article.aspx?p=1941419
5. Assumption: Hobbyist RAM is 50% of professional
6. Assumption: Pro works 2600 hours / year; hobbyist is 1040
7. Assumption: 100% of non-configuration work time is compiling + debugging
8. Assumption: Compiling + debugging time consumes 100% of available RAM

我们还可以想象另一个用户场景，programmer A将自己的一段不长的代码分享给programmer B，后者在接收代码之后发现在A的机器上perfectly work的代码在自己的环境下疯狂报错。这个季度影响开发体验和效率的场景存在的罪魁祸首是，在本地环境下进行开发时，不同开发者的环境配置或多或少有差异，然而代码可以分享，环境却不能。

解决这个问题的一个方案就是将开发环境放到云端。

云IDE是一个基于网络的集成开发平台（IDE）。

集成开发环境是一种被打包成应用程序的编程环境，通常由代码编辑器、编译器、调试器和图形用户界面（GUI）构建器组成。通常，云IDE不仅是基于云的，而且是为创建云应用而设计的。然而，有些云IDE是为创建智能手机、平板电脑和其他移动设备的本地应用而优化的。

云IDE的好处包括：

1. 可以从世界任何地方的任何兼容设备上进行访问；
2. 下载和安装最少，甚至无需任何下载；
3. 便于分散在各地的开发者之间的协作。
4. 统一了不同开发者之间的环境配置。

HTML 5的出现常常被认为是云IDE的一个关键推动因素，因为该标准支持基于浏览器的开发。其他关键因素包括日益增长的移动性、云计算和开放源码软件的趋势。

产品介绍

CloudIDE 是一个轻量级的即开即用的云端的集成开发环境。旨在为开发者提供轻量极速的在线编程体验,帮助开发者快速可靠交付代码,并打通整个开发、测试和运行时，解决开发环境依赖沉重，版本难以管理，多人维护同一份芜杂不清，分支合并缓慢低效等问题。

集成开发环境是一种被打包成应用程序的编程环境，通常由代码编辑器、编译器、调试器和图形用户界面（GUI）构建器组成。通常，云IDE不仅是基于云的，而且是为创建云应用而设计的。然而，有些云IDE是为创建智能手机、平板电脑和其他移动设备的本地应用而优化的。

无论本地 IDE 还是云 IDE，都具有两个基本作用：

- 提升开发效率：整合零碎的开发工具/服务，实现工具链的平台化
- 升级开发体验：无缝连接开发工作流，提供一站式体验

从开发者角度来看，**IDE 的关键在于对工具的整合与连接，不只是简单的工具集，而是让这些工具能以最自然的方式配合工作，组成高效的工作流**。即 `工作台/工作助理 >> 工具集`：

```
IDE >> 项目文件管理 + 文本编辑器 + 交互式终端 + 项目脚手架 + 运行任务 + 调试器 + 工具插件 + ...工具
```

对云服务供应商而言，能够实现从 Cloud Shell、Cloud Editor 到 Cloud IDE 的产品形态升级，**将一系列产品（云服务）与用户的工作流紧密结合起来，不仅能更好地表达产品功能，还能通过 IDE 更高效率地触达用户**：

```
                      ^ FaaS、BaaS
                     /
云服务用户 ---> 云IDE ---> 数据存储服务
                     \
                      v 计算资源
```

云IDE的好处包括：

1. 可以从世界任何地方的任何兼容设备上进行访问；
2. 下载和安装最少，甚至无需任何下载；
3. 便于分散在各地的开发者之间的协作。
4. 统一了不同开发者之间的环境配置。

应用场景

在肉眼可见的未来，云 IDE 有这样几个应用场景：

- FaaS：函数即服务，那么，函数在哪里写？ 独立的技术生态：如 React Native、小程序、可视化搭建系统等。
- 云计算产品：从提供离散的产品/服务（如 FaaS），转向提供定制开发环境和工作流。
- 源码管理平台：试想，GitHub/GitLab 即开发环境。
- 研发工作台：云计算时代的全云研发模式下，`需求–开发–测试–运维`的完整链路。

以云 IDE 为中心的高效研发模式，可能是这样的：

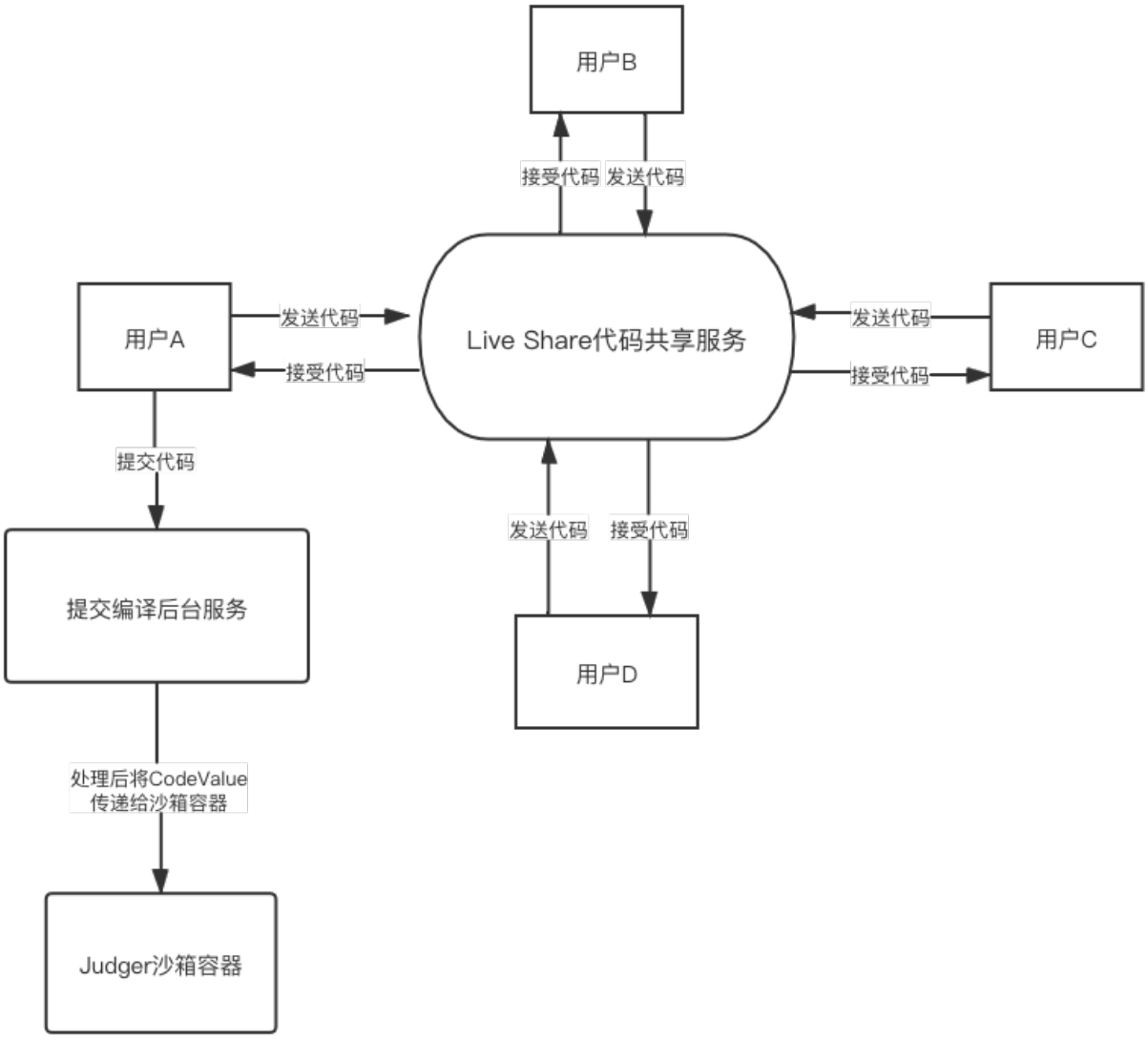
- 统一的开发环境：借助容器技术，开发环境也能作为项目的一部分，像源码一样管理起来（基础设施即代码，Infrastructure as Code），代码风格约束也能更好地落实。
- 专用 IDE：通过定制开源 IDE，提供更贴合产品/业务的专用 IDE。

- 完整的工程化链路：编辑-构建-运行-调试-测试-运维。
- 飞快的构建速度：得益于云计算的弹性调配能力，编译时长能被大幅缩短。
- Code anywhere：开发环境也能像云计算服务一样触手可得，随时随地，想码就码。
- 实时协作：在线 Review，手把手教学，共享工作空间、一键分享代码。
- AI 助力开发：基于全源码的智能提示、甚至代码生成、质量分析等。

在技术走向 techless 的同时，**研发模式或将迎来 tool-less 时代**。

服务架构与系统设计

架构图



总体设计

Cloud IDE的后端由三个服务组成：

- Code Live Share
- 代码提交
- 编译运行沙箱

Code Live Share 是一个基于WebSocket的消息转发服务器，其功能类似于一个多人聊天室。负责接收一个端点的消息并向该群组的每个客户端发送。

代码提交负责简单的消息拆包，信息传递的功能，作为客户端和Judger之间的中转站，同时处理用户相关的功能和做一些统计。

编译运行沙箱是

工作原理

Code Live Share

利用Django Channels为Django添加WebSocket功能。使用 Channel Layers管理用户群组，实现消息群发/转发。

> Channels是一个将Django扩展到HTTP之外的项目--处理WebSockets、聊天协议、IoT协议等等。它是建立在ASGI的Python规范上。
>
> Channels建立在Django自v3.0以来提供的原生ASGI支持之上，并为Django v2.2提供了一个外挂的ASGI实现。Django仍然处理传统的HTTP，而Channels让你可以选择以同步或异步的方式处理其他连接。

核心代码示例：

```
class ChatConsumer(WebsocketConsumer):

    ...

    def receive(self, text_data):
        async_to_sync(self.channel_layer.group_send)(
            "chat",
            {
                "type": "chat.message",
                "text": text_data,
            },
        )

    def chat_message(self, event):
        self.send(text_data=event["text"])
```

Judger

使用`seccomp`和Docker安全策略限制系统调用和资源占用等等。

使用Judger示例代码：

```
>>> import _judger
```

```
>>> _judger.VERSION

[2, 0, 1]

>>> _judger.run(max_cpu_time=1000,
...             max_real_time=2000,
...             max_memory=128 * 1024 * 1024,
...             max_process_number=200,
...             max_output_size=10000,
...             max_stack=32 * 1024 * 1024,
...             # five args above can be _judger.UNLIMITED
...             exe_path="/bin/echo",
...             input_path="/dev/null",
...             output_path="echo.out",
...             error_path="echo.out",
...             # can be empty list
...             args=["HelloWorld"],
...             # can be empty list
...             env=["foo=bar"],
...             log_path="judger.log",
...             # can be None
...             seccomp_rule_name="c_cpp",
...             uid=0,
...             gid=0)

{'cpu_time': 0, 'signal': 0, 'memory': 4554752, 'exit_code': 0, 'result': 0,
'error': 0, 'real_time': 2}
```

技术选型与具体实现

选型

- 前端使用Vue@2.6.2，借助Vue-cli@2.9.6开发。编辑器采用Monaco Editor。UI设计采用ElementUI。
- 后端使用Python(3.8.5)语言开发，使用的Web框架为Django(2.2.16)。
- 为了降低系统的耦合度，使系统更加健壮，将后端的**代码提交**和**WebSocket同步**两个功能分离为两个服务。
- 数据库使用Redis(6.2.1)。
- 部署方式为Nginx(阿里云应用型负载均衡ALB)+uWSGI。
- 编译运行在沙箱Judger中完成。Judger是独立在后端系统之外的服务，使用Flask进行Web封装，同样使用uWSGI部署。Django后台使用Requests调用Judger。

核心技术栈介绍

- Monaco Editor Web代码编辑器

  Monaco Editor 是VS Code代码编辑器的引擎 。

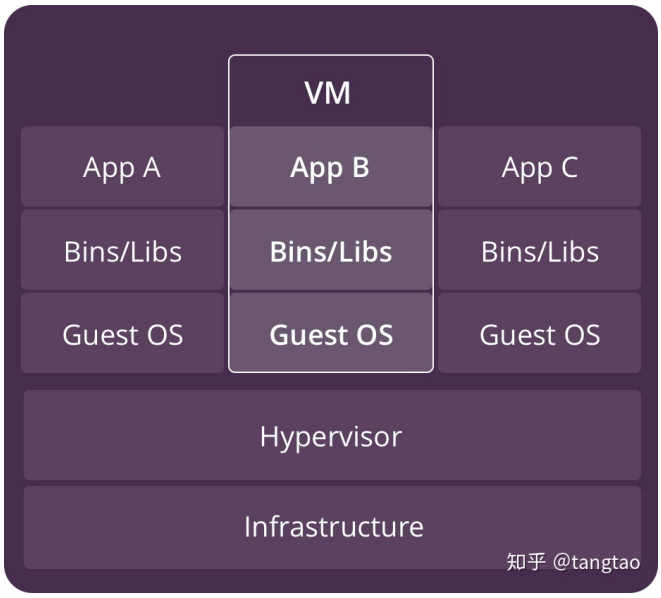
- Docker容器虚拟化技术

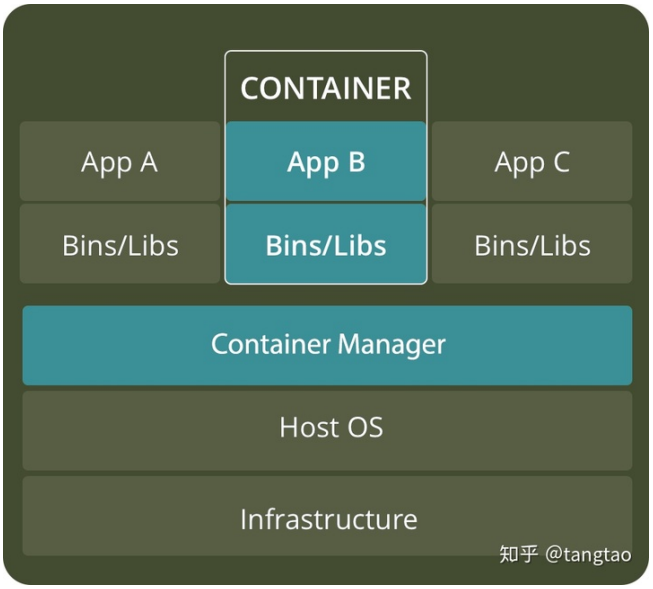
  什么是容器

  使用容器，而不是像虚拟机（VM）那样虚拟化底层计算机，只是虚拟化操作系统。

  容器位于物理服务器及其主机操作系统之上 - 通常是Linux或Windows。每个容器共享主机操作系统内核，通常也包括二进制文件和库。共享组件是只读的。共享操作系统资源（如库）可以显着减少重现操作系统代码的需要，并且意味着服务器可以通过单个操作系统安装来运行多个工作负载。因此容器非常轻 - 它们只有几兆字节，只需几秒钟即可启动。与容器相比，VM需要几分钟才能运行，并且比同等容器大一个数量级。

  与VM相比，容器所需的全部功能都足以支持程序和库以及运行特定程序的系统资源。实际上，这意味着您可以将容器上的应用程序的容量设置为使用容器的两到三倍，而不是使用VM。此外，使用容器，您可以为开发，测试和部署创建可移植，一致的操作环境。





图为容器与VM差异示意图。

容器的优势

- 可移植性

目前容器技术的现代形式主要体现在应用程序容器化（如 Docker）和系统容器化（如 LXC）中。这两种形式的容器都能让 IT 团队从底层架构中抽象出程序代码，从而实现跨各种部署环境的可移植性。

- 轻量级

容器通常位于物理服务器及其主机操作系统之上。它可以通过单个操作系统安装来运行多个工作环境。因此容器特别 “轻” —— 它们只有几兆字节，只需几秒钟即可启动。

- 降低成本

与虚拟机相比，内存，CPU 和存储效率的提高是容器技术的关键优势。由于可以在同一基础架构上支持更多容器，那么这些资源的减少就可以转化为巨大的成本节省，同时还可以减少管理开销。使用容器技术，可以降低性能开销并在同一台服务器部署上千个微服务。

- 可扩展性

容器技术与微服务架构可谓是天作之合，对实现微服务架构中各服务模块性能弹性伸缩，降低整个系统的耦合度起到重要作用。

为什么选择容器?

- Docker 允许应用程序在笔记本电脑，内部服务器，公共云或私有云等上运行，从而实现灵活性和可移植性。管理和部署应用程序要容易得多。
- Docker 容器能够快速唤起，满足Cloud IDE即开即用的便捷性的需要。
- Docker容器为运行代码的环境做了天然的隔离，是成熟的即取即用的沙盒环境。

不足与预期

不足

1. 性能方面。Python不是一个好的实现Code Live Share的编程语言。具体原因是其全局解释器锁模式下的多线程并发软肋，以及本身的效率问题。在Code Live Share的访问量大时应当采用Go来做代码重构。
2. 安全性方面。沙盒目前仅对CPU时间，内存占用，文件读写做了基本的限制。这些都是Runtime 安全。编译器安全也不容忽视。比较主要的编译器安全隐患有以下几种：
   - 引用某些可以无限输出的文件，比如 `#include</dev/random>`，编译器会一直读取, 导致卡死
   - 让编译器产生大量的错误信息，比如下面一段代码，可以让 g++ 编译器产生数 G 的错误日志

     ```
     int main() {
      struct x struct z<x(x(x(x(x(x(x(x(x(x(x(x(x(y,x(y>
     <y*,x(y*w>v<y*,w,x{}
      return 0;
     }
     ```

     处理方法就是编译器运行的时候也要控制 CPU 时间和实际运行时间还有最大输出，同时使用编译器参数 `-fmax-errors=N` 来控制最大错误数量
   - C++ 的模板元编程，部分代码是编译期执行的，可以构造出让编译器产生大量计算的代码。类似的有 Python 的编译器常量优化等等。
   - 引用一些敏感文件可能导致信息泄露，比如 `#include</etc/shadow/>` 或者测试用例等，会在编译错误的信息中泄露文件开头的内容。需要给编译器和运行代码设置单独的用户。
3. 功能方面。真正的并发编辑与协作。目前的Cloud IDE只是互相实时共享了代码。要做到像石墨文档、腾讯云文档那样的良好的合作编辑体验需要考虑到分布式编程的异步性和幂等性等等。

商业模式

用户场景

- 个人用户
  - 环境克隆：传递、克隆和共享整个环境。
  - 社区、旅行、紧急情况：避免拖着一台需要精确复杂配置的笔记本电脑到处跑。
  - 学习/教学：预先配置好的工作空间用于教学。
  - Chromebook开发：Chromebook在开发者中有一定的热度且在逐年增加，对这部分这部分用户而言，Cloud IDE进行开发是最佳选择。
- 团队用户
  - 黑客马拉松：简化团队协调合作的流程
  - 教学：教师创建练习和网上教学
  - 结对编程：谷歌文档风格的代码编辑
- 企业
  - 入职培训：快速整合和引导新加入代码库的开发人员

    - 职责分离。让管理员控制工作区的安装、配置和管理。管理员配置，经理监控，开发人员编码。
    - 多职能团队。沙盒后端、前端、文档、支持和QA环境。
    - 安全工作区。通常由自由职业者使用。公司可以创建一个隔离的工作空间，然后将安全资产放入其中，然后允许外部开发人员在监控和限制下进入该系统。
    - 招募面试。设置并观察开发人员的代码
    - 监控外包团队。衡量谁在哪里做什么。
    - 消除VDI/TS：200万开发人员仍然被迫在桌面IDE面前使用VDI/TS。云IDE更快、更便宜，并为企业提供更多控制

- ISVs
    - 技术发布。通过创建按需环境来简化API和SDK消费者的环境设置，以实际使用技术编码。
    - 商品化插件。在网上提供一个类似于Eclipse的插件环境，但不是让他们准备下载，而是可以通过分享收入来实现商品化。
    - 定制云IDE。使用云IDE引擎来构建一个新的云IDE。
    - IDElets。云IDE服务，如重构，可以作为Web服务提供，允许第三方产品重用这些服务，创建嵌入其他产品的开发者工作流。

总结

总结上述用户场景，我们相信，云IDE存在以下商业模式：

- 资格订阅。对于公共云IDE，为个人销售付费的使用资格或增值功能。
- 资源订阅。对于公共云IDE，出售对越来越多的硬件的访问，以支持构建、运行和其他资源密集型功能。
- 组织订阅。对于希望在公共云IDE中管理人口的团队和组织，你可以出售组织权限，授予隔离代码、超级管理员功能和专门的监控。
- 内部云IDE。对于许多企业来说，他们不能让任何代码离开防火墙，所以有机会在企业内部销售许可证和订阅。
- ISV包。允许ISV OEM核心功能的软件包，重新使用公共云IDE服务，托管一个插件，与云IDE供应商分成收入，或提供管理服务。例如，云IDE Codenvy提供管理工厂，这是一项允许ISVs为他们自己的用户按需启动ISVs的服务，无需认证。这些特殊的启动包括额外的能力，如更多的硬件，特殊的报告，给IDE加徽章，开发的可追溯性，等等。