WHITEPAPER

Full list of CDE vendors (+ feature comparison table)

Cloudomation

Table of Contents

Executive Summary

Cloud Development Environments aim to solve the following problems faced by developers:

- A lot of time is invested in maintaining development environments
- A lot of knowledge is required to set up and maintain development environments
- A lot of computing power is required to build, test and run the software that developers work on

CDEs propose to solve this by providing development environments which are:

- Simple to set up and use - reducing the need for both time and knowledge required for set up and maintenance of development environments.
- Run on powerful servers - reducing the load on local workstations.

The CDE market is immature, with many new products having been announced in the last two years (2023, 2024). Even vendors who have been on the market for a long time are rebuilding their products and/or have only released stable versions recently.

This whitepaper gives an overview of all CDE products currently (as of August 2024) on the market and provides insight into functionality and comparability of products.

Disclaimer - who am I and why am I writing this?

As mentioned before, with this whitepaper, my goal is to provide an overview of all CDE tools that are currently on the market, as of August 2024.

My name is Margot Mückstein and I am the CEO of a CDE vendor - Cloudomation. I also include a description of our product - Cloudomation DevStack - in this whitepaper.

However, this whitepaper is not intended as a promotional piece for our product, and I neither disparage other CDE products nor do I limit my comparison to features where our product can shine.

The research I present here was done to get a deep understanding of the tools currently available to inform our product development. I want to ensure that our product provides both what users need, and what isn't already available on the market. That requires honesty, which is the basis of this whitepaper.

I will express some personal opinions in this whitepaper. Whenever I do so, I will make it clear through the use of pronouns and the active voice (e.g. „in my opinion", „I think", „it seems to me" etc.).

I welcome your thoughts and comments on this whitepaper.

Reach out to me on LinkedIn or by e-mail: margot@cloudomation.com.

What are CDEs?

CDE is short for Cloud Development Environment. It refers to products that allow developers to do their work in a cloud environment that provides all the tools a developer needs. The CDE provides an environment with compute resources, operating system, language frameworks, developer tools, and all dependencies required to write code, build, test and run the application a developer works on.

A CDE platform provides a way to customise CDE configurations, and provides CDE management functionality to deploy, start, stop and delete CDEs.

While this is the basic idea, CDE products vary a lot in what exactly they provide. There are large differences in:

- how much of a developer's workload is moved to the cloud,
- which tools are available in a CDE,
- how customisable it is, what architecture and underlying technologies are used.

CDE is short for Cloud Development Environment. It refers to products that allow developers to do their work in a cloud environment that provides all the tools a developer needs.

Not all CDEs run in the Cloud

CDE is short for Cloud Development Environment. This implies that the CDE runs in the cloud. **While many CDE vendors offer their products as SaaS, this is not true for all of them.**

Several offer the option to host the CDE platform within customer's infrastructure. Technically, this would be RDEs - remote development environments. I mention on this to make clear that RDEs and CDEs are technologically the same product, with differences only in deployment options.

How does it work?

The most common CDE setup looks like this:

- CDE is described in a config file which is kept in a source code repository. This config file can be customised to describe which dependencies should be available in the CDE.
  - There are **two established standards to describe CDEs**:

devile.yml

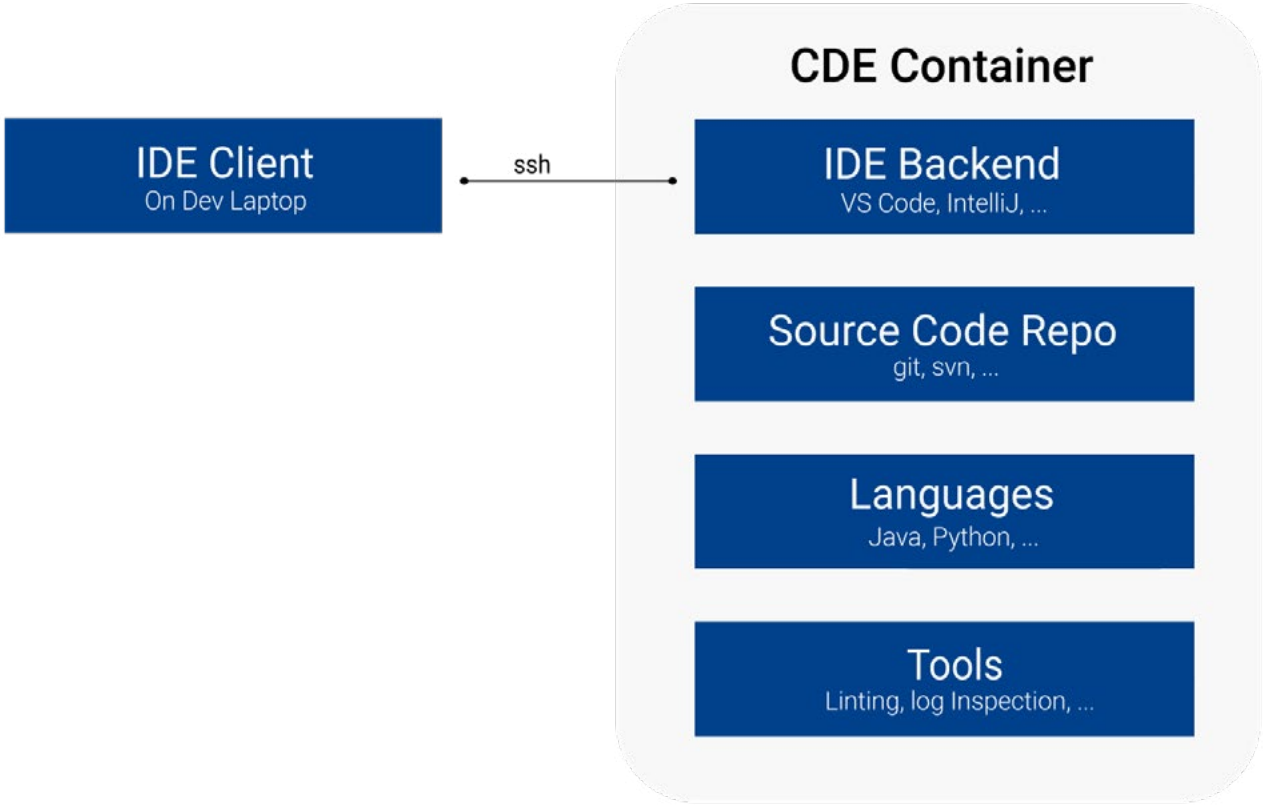
devcontainer.json

  - Both assume that the CDE is a single container and reference Docker images, Docker files or Docker-compose files, which describe a container base image. On top of this, devfile and devcontainer allow to specify development-specific additional properties for the container, such as IDE extensions or scripts that should be run after the container has started.

I wrote a blog post explaining the difference between devfile and devcontainer and which additional features they provide on top of docker-compose and dockerfiles:

https://cloudomation.com/en/cloudomation-blog/devfile-devcontainer-vs-dockerfile-docker-compose/

    - *Variants*: Due to limitations of these standards, some vendors have developed their own configuration standards that are used to describe CDEs. I wrote a blog post explaining the limits of devfile and devcontainer and why some vendors use a proprietary configuration format despite the existence of standards: https://cloudomation.com/en/cloudomation-blog/devfile-and-devcontainer-as-standards-for-configuring-cloud-development-environments-cdes/
- One CDE is one container.
    - *Variants*: While the majority of CDEs are containers, this is not true for all of them. Cloudomation DevStack, for example, allows to deploy CDEs as containers or to deploy directly to a CDE VM. Another common variant are CDE platforms that deploy several containers next to each other, with one container designated as the CDE container which contains the source code and IDE backend, which is deployed alongside several application containers.
- CDEs are consumed as SaaS.
    - *Variants*: Some CDE platforms are available as self-hosted on-premise, or as vendor-managed on-premise.
- Developers access CDEs via:
    - an ssh-capable IDE which connects to IDE backend in the CDE and a browser portal where CDEs are started, stopped, removed etc.
    - *Variants*: Many CDEs allow developers direct ssh access to the CDE. Some CDEs provide a CLI to facilitate CDE access and management.



The CDE itself typically contains:

- The source code of the software that is being developed
- Tools required to work with the source code:
  - Version control system
  - Editor / IDE backend
- All dependencies needed to build, run and debug the software that is being developed. There are large differences between CDE products regarding what can be run on the CDE. The most powerful CDEs support running the entire stack:
  - Language runtimes and software development kits (SDKs) (for example: Node.js or .NET Framework)
  - Build scripts
  - All components of the software itself: backend, frontend, database
  - All other libraries and software packages that are required by the software to run (for example: Docker)
  - Debugger, linter (usually part of the IDE)
  - Other tools as required (for example: log inspection tools)
- Anything else the developer fancies :)

On all the mentioned points, there are exceptions and variants.

Very brief history of CDEs

Startups started to tinker with the idea of CDEs in the early 2010s. Adoption has been slow for the better part of a decade. In the early 2020s, there has been a sudden explosion of new products, from established companies moving into the CDE market as well as a host of new startups announcing CDE products in 2022, 2023 and 2024.

Why now?

In my personal opinion, a congruence of three factors determined the timing:

1. **Scarcity of talent**: More and more software developers are needed, which has been fuelling a long-running trend towards increasing developer experience and productivity.
2. **Availability of technological prerequisites**: Cheap and abundant cloud computing, as well as ubiquity of high-quality internet connection, in addition to maturation of CDE-specific technologies such as configuration standards have made it technologically feasible to provide work environments to developers remotely on scale.

3. **The Covid pandemic** led to increased awareness of, openness for, and focus on technologies that enable remote work. Both other factors are long-running trends- I think the Covid pandemic was the trigger event for the sudden release of so many CDE products by many different vendors.

Here you can read why we decided to develop a CDE product now:
https://www.linkedin.com/pulse/why-cloudomation-went-rde-margot-m%C3%BCckstein/

With so many new products on the market, it is becoming increasingly difficult to understand the differences between them. In this whitepaper, I want to provide a full list of available CDE products as of August 2024, and give an overview of the differences between them.

IDEs and CDEs

The main difference between "just" an IDE and a CDE is that the **IDE allows only to work with the source code, while a CDE also provides a runtime environment to build and run the software that is being developed.**

Since the CDE market is fairly young, this distinction is often not made. There are IDE vendors labeling their products as CDEs and a lot of content out there that confuses the two product categories. I wrote about this issue in our blog: The difference between IDEs, local development environments and remote development environments.

In reality however an IDE and a CDE provide different feature sets that make them complementary. IDEs are typically used in combination with CDEs. CDEs usually do not come with an IDE but offer compatibility or integration with a set of IDEs.

Most CDEs can be used with ssh-capable IDEs. Ssh-capable IDEs enable a developer to only run a thin client locally, connect to the IDE backend via ssh, and outsource all compute-intensive tasks like debugging to the ssh backend, which runs on the CDE. The CDE is the runtime environment for the IDE backend as well as the software that the developer works on.

A CDE platform makes it possible to configure, customise and manage these remote runtime environments, while the IDE provides functionality related to code editing.

CDE configuration standards

There are two established configuration standards for CDEs: devfile.yaml and devcontainer.json. In addition, the nix package manager provides functionality that is somewhat adjacent to CDE configuration by allowing to package dependencies that make up a development environment.

I wrote about the difference between devfile, devcontainer, dockerfile and docker-compose in our blog, and also explain a bit more about devfile and devcontainer as configuration standards for CDEs.

Dev Container

Dev Containers is a partially open source standard for describing containerised development environments. The devcontainer project includes a small command-line tool (CLI) that allows developers to locally deploy a development container based on a devcontainer.json config file. It doesn't include tooling to run dev containers in the cloud or to connect to Dev Containers remotely. That is also not the goal of the Dev Containers project: it primarily seeks to define a standard for defining containerised development environments.

More info: https://containers.dev/

Devfiles

Devfile is an open source standard for defining containerised development environments. Similar to the devcontainers standard, it is not itself a CDE solution but a standard that many CDE products use for configuration of CDEs.

More info: https://devfile.io/
Docs: https://devfile.io/docs/

Nix package manager

Like the name suggests, Nix is a package manager and not a CDE tool. A package manager allows defining bundles of libraries into a package that contains all dependencies for an application to run. As it happens, it is possible to define development environments as such a package. Nix showcases this here: https://nixos.org/#asciinema-demo-example_3. There are other resources out there describing how Nix can be used to define and distribute development environments as Nix packages.

The USP of Nix is that it creates isolated packages. This means each package contains everything it needs, and defining a package requires explicitly stating every dependency in its required version. This makes it possible for different packages to use different versions of the same library, removing dependency conflicts.

Since dependency conflicts are often a big problem when working on different versions of the software you develop (which might require different versions of dependencies), this can be solved by creating Nix packages for your different versions.

However, being a package manager, Nix doesn't provide a platform to manage CDEs, or infrastructure where they can be run.

More info: https://nixos.org/
Docs: https://nix.dev/

Main differences between CDE products

Below, I describe the categories I use in the feature comparison table further down.

USP

Where discernible, I point out the USP(s) of CDE products as presented on their websites.

Customizability: What can run on the CDE

I like to group the CDE landscape into **three broad buckets** which can be placed along an axis of increasing customizability and "power" of the CDE products. This refers mostly to which software can run in a CDE, and how much the deployment logic of the CDE and the software that runs on it can be customised.

Categories for the feature comparison table

(A) **Highly opinionated:** Only one specific type of software is supported. The CDE deployment is largely hidden from the user. The user can interactively customise their CDE after deployment, but has no or only limited options to customize the CDE template, often via a graphical user interface. Examples are CDEs for Windows Desktop Software, or that support exclusively node.js projects.
(B) **Intermediate**: These CDEs have a fixed deployment model that the user cannot customise, but allow flexible customisation of the software that runs within the CDE. Since this is the largest group of CDE vendors, I added three sub categories which I explain in the chapter below: containers vs. VMs

- **B1**: Single container
- **B2**: Multi-container (without Kubernetes, containers running directly in Docker)
- **B3**: Kubernetes

(C) **Agnostic**: (Almost) any type of software is supported. This includes Docker-only setups for multi-container applications, as well as cluster-based, single-container, single VM or multi-VM deployments, or serverless applications, as well as hybrid setups where e.g. some services are shared between CDEs, while others are unique to each CDE. Users have full control over the deployment logic and can customize fully what constitutes a CDE. Typically, users have access to infrastructure or deployment automation software where they can fully customise templates, or create their own CDE deployment logic.

Containers vs. VMs

In the previous version of this whitepaper, one of the distinctions I made between CDE products was whether they are deployed as VMs or containers. I argued that restricting developers to a single container as their work environment was limiting, making it impossible to deploy several containers in their CDE.

I removed this distinction, because the majority of container-based CDEs use technologies like Sysbox (or similar container runtime technologies) which provide the experience of using a VM, while working with a container. **This means that Docker, Kubernetes and other system-level software can be run without issues within those containers, while maintaining the isolation of environments that containers provide, and allowing for more efficient resource utilisation and scaling than VMs. These are the ones I categorize as B2.**

B (Intermediate): Fixed deployment, but flexible customization of the software that runs within the CDE.

However there are still some CDE vendors who use **"standard" containers as CDEs, which are not suitable to run Docker and several containers within them. Those are the ones I categorise as B1.**

B3 containers also provide one standard single container as the developer's work environment, which contains the source code and an IDE backend. However these containers are deployed alongside other containers that make up the application that the developer works on, removing the need for the developer to deploy several containers within their CDE. This has the downside of separating the developer from the other containers, making it more complicated to access logs and inspect services running in those other containers. But it has the upside of being closer to a production environment- assuming that the application the developer works on runs in Kubernetes in production.

Configuration

There are (at least) three layers of configuration that make up a CDE:

- **Infrastructure**: This relates to the infrastructure unit(s) that make up a CDE. All CDEs allow to specify the available resources (cores, memory) for the CDE. Some CDEs allow further configuration of the underlying infrastructure, e.g. by specifying whether the CDE is a VM or serverless or one container or several containers (C), or by configuring the cluster on which the CDE runs (B3). In the comparison table, I list the technology available to the user to configure the infrastructure layer. One difference between CDE products is the ability or inability to template the infrastructure layer. Many CDE products exclude the infrastructure layer from their templating options (A, B1, B2), meaning that a developer has to choose the required resources for a CDE every time when creating a new CDE (though it is often possible to set a default). That is not because it makes sense for the developer to choose this, but rather because it cannot be included in the software-layer, which is the only place that most

CDE products allow to template. Those CDE products that allow broader customisation of the infrastructure layer typically also allow to fully template resource specs.

- **Software**: This relates to the software that runs within a CDE and concerns the operating system (OS) and everything above. The line between infrastructure and software is a bit blurry when it comes to containers: Deploying several containers into a cluster in addition to a CDE container is typically configured at the infrastructure level, while the deployment of several containers within one CDE (Docker-only) is typically configured on the software layer of the CDE. The software layer is the one that is typically fully templated, meaning that developers spin up a CDE without having to make any decisions about the software layer.
- **User-specific**: This relates to any configuration that is done on top of the CDE template(s) which is unique to a specific user. User-specific configuration is typically applied in one of three ways:
  - *Interactive*: Anything that is done manually by the user and has to be repeated for each new CDEis considered interactive user-specific configuration. This includes choices the user has to make whenever they deploy a new CDE. For example, the user may have to choose which IDE they want to use. It might (or might not) be possible to set default values for these choices. Interactive user configuration also includes everything the user does manually after the CDE has been created, to set it up for themselves. For example, the user might customise port forwardings, or alter linter settings etc. Interactive configuration is poor practice and should be minimized as much as possible. While it is often good and sensible to enable users to deviate from default values, any configuration that is expected to be applied regularly should be templateable.
  - *User settings*: Many CDE products store some user-specific information within the CDE platform. For example, the user typically can upload their public ssh key(s) to the CDE platform, which is then automatically injected to each CDE the user deploys.
  - *Templated*: Some CDE products allow users to apply their own settings via configuration files. For example, some CDE products allow developers to specify dotfiles that store different types of user preferences, such as IDE settings like color schemes etc. Other CDE products allow users to store some preferences in custom configuration files.

User-specific configuration is done in many different ways and is not a distinguishing feature for most CDE products. I therefore don't go into detail about user-specific configuration in the feature comparison table.

Compatible IDEs

Most CDE products list specific IDEs their products work well with. Since this is a distinguishing feature, I include a **list of compatible IDEs in my feature comparison table. In the table, I list only those IDEs that vendors mention on their website, which are IDEs their products are tested with and/or have specific plugins for.**

There are fundamentally different ways of using IDEs with a CDE:

- **Purely browser-based**: Almost all CDEs offer at least one IDE that can be opened in the browser and allows developers to work with the CDE without any dependencies on the local device (besides a browser). In the majority of cases, this is (some fork of) VS code.
- **Ssh-based with local thin client**: Many CDEs also offer the option to use their CDEs with a range of ssh-capable IDEs. Such IDEs consist of two components: the IDE backend, which runs on the CDE, and an IDE thin client, which the developer installs on their laptop and configures to connect to the IDE backend on the CDE via ssh.
- **As far as we know, only Cloudomation DevStack offers a third option**: The source code is mirrored between the CDE and the developer's laptop. The developer can use any IDE or editor they like, fully locally. Developers can keep using existing workflows, have the source code in their local file system, can work on the code even when offline, and outsource only compute-intensive tasks like building, testing and running the application to the CDE. As an agnostic solution, DevStack supports a purely browser-based workflow using VS code in the browser, as well as ssh-capable IDEs in addition to the option of mirroring the source code locally.

In addition to what CDE vendors list on their websites, I want to point out that many CDE products are compatible with any ssh-capable IDE, which includes VS Code, JetBrains IDEs, Eclipse Theia, Jupyter, RStudio Server and probably others.

However, configuring those CDEs to work with an ssh-capable IDE that is not natively supported may, while possible, require some effort. How difficult it is varies a lot between CDE products and is generally easier with agnostic CDEs than with more opinionated ones.

As a purely terminal-based editor, vim is a special case that can be used with any CDE that allows direct ssh access to the CDE via a terminal.

CLI

Yes (y) / No (n) in the feature comparison table

Command Line Interfaces (CLIs) **allow management of CDEs directly from a developer's local terminal**. CLI features vary but generally include the ability to create, start, stop and delete CDEs via the command line, as well as connection to the CDE via ssh. Other features can include port forwarding to localhost, tailing of logs of applications that run on the CDE, and other, product-specific features.

The main benefits of a CLI are:

- Ability to script / automate CDE management
- More seamless way of working for the developer, especially for developers used to working with the terminal

Note that a CLI is a local tool that a developer needs to install on their laptop. Most CDEs aim to provide a work environment that is fully independent of the local device, so the CLI is always an optional "add-on" that developers can, but don't have to use.

Source Code Security

I use the the following categories of source code security to compare CDEs:

Categories for the feature comparison table

1. **The entire source code is stored on each developer's laptop.** This is the standard for git-based source control systems and has the benefit of each developer's laptop doubling as a backup. However, this also means that theft or loss can happen fairly easily. This can be addressed with standard security tools like disk encryption etc. For most companies, this is sufficient.
2. **Source code is accessed via ssh**, e.g. by the developer directly via a terminal or through an ssh-capable IDE, or through a browser-based IDE. Malware would have to be very clever to get at source code via ssh, however a targeted attack could access it. Access to the source code can be removed centrally. Developers can easily copy or download source code files to their local file system.
3. **Developers work via a remote desktop software.** This can be set up so that nothing can be copied between the remote and the local machine, and the source code is streamed as images through the remote desktop client. It will be very hard to steal the source code bar recording the screen and having AI extract text from the recorded images.

Most CDEs offer level 2 security: remote access to the source code. If danger of loss or theft is a relevant concern, this is already quite a good level of security.

So source code security can really only help to reduce the risk of theft or accidental loss- for example, malware that steals the source code, or loss through the loss of a laptop with the source code on it.

In my opinion, there has to be some degree of trust between developers and their employers. Developers naturally need access to the source code which they work on. No CDE can prevent a motivated developer from stealing source code if they want to. So source code security can really only help to reduce the risk of theft or accidental loss- for example, malware that steals the source code, or loss through the loss of a laptop with the source code on it.

SaaS and on-prem

Yes (y) / No (n) in the feature comparison table. For on-premise, a (K) is added when a Kubernetes cluster is required.

The available deployment options for CDE products vary, with many offered only as SaaS, others only as on-premise, and some offering both options. **For CDE platforms available on-premise, I added a (K) for those that require a Kubernetes cluster to run**, since this has significant implications for self-hosting. Note that some other on-premise CDE platforms can optionally be run with a Kubernetes cluster, but do not require one. I added the (K) only to those that cannot be run without Kubernetes.

Pricing model

This describes what the user is charged for. Most CDE products charge per hour, some charge per user, and some have more complex pricing models that combine hourly and per-user charges with other usage metrics.

Approx. price / dev / month

Even though per-hour pricing is used by most CDE vendors, **the details of what an hourly price includes and which other charges are applied vary a lot. To make this more comparable, I calculated the approximate cost per month per developer for each CDE product in the comparison table, based on a standard usage model.** For some vendors, extensive research was necessary to arrive at the pricing estimates presented here. **They are all based on the following assumptions:**

- 160 hoursof CDE runtime per month
- 4 core / 8GB memory CDEs
- A team of 20 developers. This is relevant for products that charge a flat-rate fee which is independent of the number of developers using the CDE platform, such as support or a fee for a single Kubernetes cluster which can be shared by several developers. Such fees are divided by 20 to approximate the cost for an individual developer.
- “Standard” usage in other metrics such as storage, networking etc., which are relevant for only very few CDE product’s pricing and make up a small portion of the total price
- No “extras”, e.g. no GPUs, prebuilds or other product-specific extras
- All estimates are based on publicly available pricing information. Many vendors indicate that custom pricing packages are available for enterprise customers, which may provide different conditions for larger numbers of developers.
- Where possible, infrastructure and license cost were calculated separately (possible mostly for on-premise products)

- **For on-premise products**, infrastructure cost was based on Azure on-demand pricing for Sweden Central in August 2024
  - *For VM-based CDEs*:
    > 1 standard tier Linux A4 v2 node with 4 Cores and 8GB RAM = € 27,26
    > 1 Standard HDD S4 32GB = € 1,44
    > **TOTAL: € 28,70**
  - *For Kubernetes based CDEs*:
    > 1 standard tier AKS cluster: € 68,34 / month, divided by 20 to approximate cost for individual developers in a shared cluster = € 3,42
    > 1 Linux A4 v2 node with 4 Cores and 8 GB RAM for 160 hours = € 27,26
    > 1 Standard HDD S4 32 GB = € 1,44
    > **TOTAL: € 32,12**

As can be seen in the table, there are large differences in the prices of different CDE products. Since the CDE products also vary quite a lot in what they offer, differences in price make sense though they do not clearly reflect the quality or power of the underlying solution, but rather the pricing power of the vendor and/or their underlying cost.

This shows that the market is still immature. **No common price points have been established yet, though there is some convergence at a price point of approximately € 30-50 / dev / month, excluding infrastructure cost.**

The fact that some SaaS vendors offer their CDEs for a similar price point including infrastructure cost indicates a starting price war between the vendors of very similar SaaS solutions who may be unable to compete on functionality. Such pricing is probably subsidised by venture capital and/or startup credits from the large cloud providers and are probably not sustainable.

Infrastructure costs can vary a lot depending on hosting options. Some CDE vendors charge for infrastructure separately, with hefty surcharges on top of the underlying compute costs.

The fact that there are also large differences in the underlying pricing models makes price comparisons difficult, though there seems to be a convergence towards hour-based pricing models.

Initial release

Here, I mention the year of the initial release of the product's current version. Some CDE vendors have significantly rebuilt or completely relaunched their products. In such cases, I will note the release year of the current / latest product.

Vendor HQ

Being a European vendor ourselves, I thought it interesting to show how concentrated the CDE vendor market is geographically.

Comment

In the comment column, I point out specific characteristics of individual CDE vendors or products that apply to only a few and therefore do not warrant individual columns.

Excluded: version control

Many CDE vendors list prominently on their website which IDEs and which version control systems their CDE product is compatible with. I exclude version control from my feature comparison table because it is a bit nonsensical - any version control system that works on Linux can be used on most of the CDEs I describe here (exception: Microsoft Devbox, which is not Linux based), including svn and all git-based version control systems. Git-based version control systems include Gitlab, Github, Bitbucket, AWS CodeCommit, Azure Repos, Google Cloud Source Repositories, git itself and many others.

Feature comparison table

The table is ordered by the timeline of the CDE products first release. I exclude the following CDE products from my comparison, since they are not under active development / have been discontinued. I do mention each of them briefly in the list of vendors below: Koding, Nimbus, Hocus.

			Configuration								Approx. cost / dev / month					
Vendor	**USP**	**Customiseability**	**Infrastructure**	**Software**	**Compatible IDEs**	**CLI**	**Source Code Security**	**SaaS**	**On-Prem**	**Pricing model**	**Licence**	**Infrastructure (4 core, 8 GB RAM)**	**Total**	**Initial release**	**Vendor HQ**	**Comment**
Eclipse Che	CDEs in Kubernetes	B3	CheCluster.yml	devfile	VS Code, JetBrains IDEs, Eclipse Theia	y	2	n	y	Open Source	0	€ 32,12	€ 32,12	2016	Canada (Eclipse Foundation)	Oldest CDE project
Red Hat OpenShift Dev Spaces	CDEs for OpenShift, based on Eclipse Che	B3	CheCluster.yml	devfile	VS Code, JetBrains IDEs	y (OpenShift CLI)	2	y	n	Per hour, part of OpenShift subscription	?[1]	?[1]	?[1]	2019	US	Commercial SaaS offering of Eclipse Che
Gitpod	No discernible USP	B2	Not possible	gitpod.yaml	VS Code, JetBrains IDEs, Jupyter	y	2	y	n[4]	Per hour	n / a	n / a	€ 39,60	2021	Germany	
Github Codespaces	Launch CDE directly from Github	B2	Not possible	devcontainer	VS Code, JetBrains IDEs, Jupyter	y (Github CLI)	2	y	n	Per hour	n / a	n / a	€ 43,74	2021	US	
Strong Network	Security, observability	B1	Not possible	GUI	VS Code, JetBrains IDEs	n	2, 3	?[5]	?[5]	Per user	?[5]	?[5]	€ 36,45	Probably 2021 or 2022	Switzerland	
Okteto	Production-like CDEs in Kubernetes	B3	Okteto manifest, helm, kubectl	Okteto manifest	Any (local source code)	y	1	y	y	Per user	€ 92,53	€ 32,12	€ 127,65	2022	US	
Coder	„Wrapper for Terraform"	C	Terraform	Terraform or devcontainer or dockerfile	VS Code, JetBrains IDEs, Jupyter, RStudio	y	2	n	y	Per user	€ 32,71[2]	€ 28,70	€ 61,41	2023 (V2)	US	Open Source with limited features, Enterprise licences for support & advanced features
CodeSandbox	Fast spin-up of CDEs, collaboration	B2	Not possible	devcontainer	VS Code, proprietary in-browser editor	n[3]	2	y	y (Enterprise only)	Per hour	n / a	n / a	€ 57,95	2023 (DevBoxes)	Netherlands	Low-code dev tools, focus on FE devs, performance, shared CDEs: 1 per branch
DevZero	Smart caching for faster builds, AI for writing recipes	B3	Not possible	DevZero recipe	Proprietary in-browser IDE, VS Code, Eclipse Che, JetBrains IDEs	y	2	y	y	Per user	€ 28,03	?	?	2023	US	Only works with Github repositories
Amazon CodeCatalyst	Simple to use with other AWS tools	B2	Not possible	devfile	AWS Cloud9, VS Code, JetBrains IDEs	y (AWS CLI)	2	y	n	Package includes 200 hours per month per user	€ 18,69	€ 16,80	€ 35,49	2023	US	
GCP Workstations	Gemini code assist integrations, simple to use with other Google Cloud tools	B2	Not possible	WorkstationConfig.json	RStudio, JetBrains IDEs, VS Code	y (Google Cloud CLI)	2, 3	y	n	Per hour	€ 41,52	€ 28,70	€ 70,22	2023	US	
Microsoft Dev Box	Windows Remote Desktop VMs	A	Not possible	GUI	Visual Studio, Windows compatible IDEs	y (Azure CLI)	3	y	n	Per Dev Box	n / a	n / a	€ 138,03	2023	US	
Cloudomation DevStack	Code stays local, supports complex deployment models	C	Cloudomation Engine	Devfile, devcontainer or custom	Any	y	1, 2	y	y	Packages with shared hours per team	€ 41,95	€ 28,70	€ 70,65	2024	Austria	
Gitlab Workspaces	Integrate tightly with Gitlab	B3	Not possible	devfile	Gitlab web IDE (VS Code fork)	n	2	y	y	Per user	€ 26,81	€ 32,12	€ 58,93	2024	US	
JetBrains CodeCanvas	Easy to use with other JetBrains products	B1	Not possible	GUI	JetBrains IDEs	n	2	n	y	Per user	€ 46,22	€ 28,70	€ 74,92	2024	Poland	Overlap with JetBrains Space
IDX by Google	Gemini coding assistant, Android and web emulator	B2, C	VM	Nix	VS Code	n	2	y	n	Free (for now)	€ 0	€ 0	€ 0	n / a	US	Public beta
DevPod by Loft Labs	Desktop app for local CDE deployment	?	Not in scope (laptop is infrastructure)	devcontainer	VS Code, JetBrains IDEs	y	1	n	y	Open source	€ 0	€ 28,70	€ 28,70	n / a	US	Alpha
Daytona	No clear USP	B2	?	devcontainer	Web-based IDEs, no specifics	y	2	n	y	Open source	tbd	n / a	tbd	n / a	US	Alpha
CPS1	Kubernetes-based CDEs that „just work"	B3	Not possible	devfile	VS Code in browser	n	2	n	y	Per user	tbd	n / a	tbd	n / a		Not released

Category description
Here you will find a brief summary of the categories „Source Code Security" and Customisability", as they are difficult to understand without context. Please refer to the chapter „Main differences between CDE products" to learn more about the individual categories.

Source Code Security

1. The entire source code is stored on each developer's laptop.
2. Source code is accessed via ssh.
3. Developers work via a remote desktop software.

Customizability: What can run on the CDE
(A) Highly opinionated: Only one specific type of software is supported. The CDE deployment is largely hidden from the user.
(B) Intermediate: These CDEs have a fixed deployment model that the user cannot customize, but allow flexible customization of the software that runs within the CDE. There are hree sub categories: containers vs. VMs

- B1: Single container
- B2: Multi-container (without Kubernetes, containers running directly in Docker)
- B3: Kubernetes

(C) Agnostic: (Almost) any type of software is supported. This includes Docker-only setups for multi-container applications, as well as cluster-based, single-container, single VM or multi-VM deployments, or serverless applications, as well as hybrid setups where e.g. some services are shared between CDEs, while others are unique to each CDE.

[1] OpenShift dev spaces are bundled with OpenShift, which is purchased as an entire cluster with per-hour pricing. It is impossible to untangle or estimate the price of one dev space within an OpenShift cluster.

[2] Coder has no pricing information on their website (anymore). The estimate here is based on pricing information from the Google Cloud marketplace, as well as historic pricing information from 2021, when Coder still had prices on their website.

[3] There is a CLI for their „Sandbox" IDE product, but not for their Devbox CDE product.

[4] Gitpod can be run „within your own cloud", i.e. customers can give Gitpod access to their AWS account and Gitpod will deploy their platform into the customer's AWS account and manage it there. Gitpod gave up their self-hosted option, probably because they were unable to give customers a goold experience in self-hosting their complex Kubernetes-based platform.

[5] On their website, Strong Network has no pricing information available and doesn't explicitly mention hosting options. In their feature comparison table, self-hosted in presented as a core feature. On G2, they list a pricing model for a SaaS option, but it is not clear whether this is up-to-date.

List of vendors

Below, I briefly describe each CDE product and provide links with further information. To make navigation easier, the list is sorted alphabetically.

Amazon CodeCatalyst Dev Environments

Amazon calls its CodeCatalys a “software development service” which intends to offer automation and cloud resources for a number of different DevOps tools and processes, among them development environments. As such, CDEs (called “Dev Environments”) are presented as just a part of the package, which includes CI/CD tools, monitoring and even a ticketing system.

As befits AWS, it is a complex product. Admitting that a lot of the feature richness (or complexity, however you want to call it) of CodeCatalyst is lost in this, I will try to describe it very briefly and simply:

Amazon Dev Environments are single CDE containers, defined using the devfile standard. They are intended to run an IDE backend and allow access to the source code stored in AWS git. A standard Dev Environment doesn’t contain the application that a dev works on, but instead is integrated with AWS CI/CD workflows to build and deploy the application in a production-like manner into AWS compute resources. As such, the developer is separated from the application.

As befits AWS, it is a complex product. It can do a lot, but it also requires a lot of expertise to get started and assumes that other AWS services are used for all aspects of the development workflow, such as AWS git, AWS CI/CD and of course AWS as infrastructure for everything.

My personal opinion: Working with such a CDE provides a very different way of working than when working locally, building and deploying the application on the developers workstation. It will feel the same as working with the code locally, pushing to a source repo, and waiting for the CI/CD pipeline to finish in order to inspect the application in a remote environment - a form of working that means slow iterations. I wrote a somewhat opinionated blog post on this: The problem with developing blindly.

It also provides very little added value compared to working locally. If the build is done in a CI/CD pipeline, and the application is deployed remotely anyway, then this is typically triggered by a commit to the source code repository, which I can do just as well locally.

Having a place to run the IDE backend is only marginally valuable - only for projects that are so big that even debugging is too compute-intensive to run locally. But that is usually not the problem. Version control and IDE are parts of the development environment that already work very well locally. The problem is the

application itself: local build, local deployments. CodeCatalyst does not aim to support this.

A side note: Amazon recently decided to discontinue a number of dev / devops focused products, among them their Cloud9 IDE, which it promoted as the IDE to be used with CodeCatalyst. It remains to be seen if this presages a bigger shift of AWS away from providing dev-tools, and what that would mean for the future of Amazon CodeCatalyst.

Amazon CodeCatalyst is best for:

- Companies that already use other AWS products for source control and CI/CD and who run their software on AWS infrastructure in production
- Companies with deep AWS expertise and dedicated teams that would provide Dev Environment templates as a service to developers
- Companies that are not looking to provide developers with a "local–like" development experience but who are fine with developers working blindly (which, in my opinion, is only justified if there really is no way to provide a development environment to developers that allows direct access to the application.)

Product website: https://codecatalyst.aws/explore/dev-environments
Docs: https://docs.aws.amazon.com/codecatalyst/latest/userguide/devenvironment.html

Cloudomation DevStack

Full disclosure: This is my company's product

Cloudomation is a software startup that started out with an automation platform (called Cloudomation Engine). They launched their DevStack CDE product in a closed beta in 2023 and reached general availability in early 2024. It is based on their automation engine, which is used to deploy the CDEs.

The default usage model of DevStack assumes that each CDE is a single VM on which the application under development is built and run in exactly the same way as it would on a developer's laptop. Source code is mirrored between the CDE and the developer's laptop. In this way, developers can continue to use their favorite IDE and other tools locally and do not have to change their workflow when starting to use a CDE. Interaction of the CDE is done primarily via a command-line interface that allows developers to:

- Create new CDEs, start, stop, delete CDEs
- Start, stop and configure port forwarding
- Start, stop and configure two-way file synchronisation between the CDE and the developer's laptop - this includes the source code, but can also sync any other files, e.g. f iles produced or used by the application that developers want to inspect with specific tools locally
- Open interactive terminal sessions on the CDE

- Tail logs on the CDE
- Configure the CDE, e.g. timeout to hibernation, directories for file sync, which DevStack instance to connect to etc.

All configuration done via the CLI can be templated and stored in a configuration file as well.

CDE management can also be done via a web interface.

The standard CDE setup deploys a VM for each CDE and allows the user to fully customise what should be deployed to this VM:

- A single development container specified in a devfile.yaml or devcontainer.json
- A multi-container application described in a docker-compose file
- Fully custom deployment of any type of software directly to the VM, with or without Docker

In addition, it is possible to create custom CDE templates for other, non-standard deployment models such as:

- Multi-VM applications
- Complex deployment models, e.g. mixing shared services and services that are deployed uniquely for each developer, with custom logic to define which services have to deployed and connected in which situation
- Serverless apps
- Applications running in Nomad or Kubernetes clusters
- …

Templating of CDEs is done in Python, which provides limitless flexibility to deploy CDEs of any type that the user requires. Templates for common deployment models (such as using devfile, devcontainer, single-VM) are available and can be used with minimal additional configuration effort.

With Cloudomation Engine as a Python-based general-purpose automation platform under the hood, DevStack CDEs can also be seamlessly integrated with existing tools and workflows seamlessly.

At the moment, DevStack is available only on-premise, with the option to choose self-hosting or managed on-premise where the vendor takes care of management of the CDE platform, but on the customer's infrastructure. A SaaS offering is planned for the future.

DevStack is best for:
- Organisations with complex and / or non-standard deployment models of their applications

- Organisations with well-working existing development workflows who want to keep developers working as they are, with CDEs that integrate well into existing toolchains and workflows
- Organisations looking for a botique vendor that provides personal support and services

Product website: https://cloudomation.com/devstack/
Documentation: https://docs.cloudomation.com/devstack/

Coder

In my opinion, Coder is one of the most interesting CDE vendors. It was founded in 2017, underwent several fundamental changes to its product, and released a fundamentally changed Coder v2 in August 2023.

Their product history shows how the market develops.

Coder started out developing VS Code for the browser as an open source project. They are the original authors of code-server which makes it possible to run VS Code in the browser. This was and still is hugely successful: It had 66,8k stars on Github at the point of writing this whitepaper.

From there, they listened to their users and realised that “just” an IDE is not enough: developers need to be able to run the software they develop as part of their work environment. They then started to build a container-based CDE platform.

Again listening to their users, they then realized that a container is not enough: developers of more complex software - which are the ones who most benefit from using a CDE because their development environments are the most difficult - cannot run their software comfortably within a container.

So they set out to provide VMs as CDEs. Constrained by their initial architecture choice of providing CDEs as containers within Kubernetes, their first solution consisted of providing container-based virtual machines (CVMs) which can run Docker within a container. This is painful to manage and comes with a significant performance penalty.

So they set out to make their CDEs even more flexible. Coder version 2 allows users to use a number of different Terraform templates to deploy workspaces as VMs, containers, or whatever else is possible to define with Terraform.

Now, Coder positions this flexibility regarding the infrastructure of CDEs as their USP. To me, this showcases nicely the story of a solution that is incrementally improved based on feedback from users.

Unfortunately, it also shows the typical evolution of any software solution: it started out very simple, and became more and more complex. While Coder now supports a wide range of use cases, it is also more complex to set up and use than many other CDE solutions.

Coder is best for:

- Companies that need alot of flexibility in the definition of their CDEs, e.g. because they have non-standard or complex deployment models
- Companies that already have experience with Terraform or who already use Terraform for production and other deployments
- Companies with dedicated platform engineering teams that can manage the Coder platform and provide automation templates for developers as a service

Docs (v2): https://coder.com/docs/v2/latest
Product website: https://coder.com/

Coder Pricing

The coder platform is open source and free to use with a limited set of features. For support and extended features, an enterprise offering is available but pricing details for this are not publicly available.

However in the past, Coder had per-user pricing on their website, which they removed some time after 2021. Back then, the cost per user per month was USD 35, billed annually, and self-hosted, i.e. excluding infrastructure cost.

At the time of writing this whitepaper (August 2024), Coder still had pricing information online on the Google Cloud marketplace, where a package for up to 10 users is available for USD 350 / month, again billed annually and excluding infrastructure cost.

I used this information as a basis for the cost estimate provided in the feature comparison table, but want to point out that it may not be up to date.

Codesandbox

Codesandbox has also recently changed the core architecture of their product and continues to make fundamental changes to its product.

At the time of writing this whitepaper (August 2024), Codesandbox supports two types of CDE products:

- Devboxes, which are based on microVMs and support running multi-container Docker-based applications
- Sandboxes, which run purely in the browser, with a severely limited feature set (e.g. only JavaScript projects and no Docker support). At the end of 2023, Codesandbox deprecated their “legacy sandboxes”, but continue to support Sandboxes in their new platform.

The feature comparison table as well as the rest of this section will focus only on the Devboxes product.

Codesandbox has a clear focus on frontend developers. Its documentation heavily features examples from frontend development. It provides a set of dev tools within their custom built in-browser IDE which are tailored to work on visual elements, supporting the work of designers and frontend developers.

Codesandbox is best for:

- Frontend teams or teams working on software with a strong focus on user experience and design
- Dev teams that have no established toolsets or working practices yet (e.g. who are starting a new project) or who are explicitly looking to change their existing workflows and tools

Docs: https://codesandbox.io/docs/
Product website: https://codesandbox.io/

CPS1

CPS1 (short for “cloud programming shell 1”) is an on-premise Kubernetes-based CDE platform developed by a Brazilian startup.

CDEs are configured using Devfile.yaml and are deployed as single containers to a Kubernetes cluster, alongside other application containers. Management of CDEs is done via a web interface.

Product website (in Portguese): https://www.cps1.tech/

Daytona

Daytona was founded in 2023 and decided to open-source their product, currently in early beta, in spring 2024. Despite the early state of their product and recent founding date, they are very present in the online discussion about CDEs.

Calling their product a “Development Environment Manager”, it follows a very similar approach to DevPod. Developers install Daytona locally on their laptops, and use it to deploy container-based CDEs either locally using Docker, or to remote infrastructure. Developers will appreciate their command-line-first approach, currently exposing their product exclusively via a CLI (command-line interface). CDEs are single containers which can be described with a devcontainer.json, or by specifying a container image directly.

This means that developers are again responsible for managing and configuring the deployment of their development environments themselves. They set up and configure Daytona on their laptops, meaning that they cannot use Daytona CDEs from other devices.

Daytona is best for:

- Individual developers curious to test and provide feedback to an early-stage CDE product

Product website: https://www.daytona.io/
Documentation: https://www.daytona.io/docs/

DevPod by Loft Labs

DevPod is an open source project with a different approach to CDEs than most other tools. It is a desktop app (available for Windows, Mac and Linux) which can be used to deploy CDE containers to almost any environment that can run Docker containers, including Kubernetes clusters in the cloud or the developer's own laptop.

I say almost any environment because DevPod comes with a certain set of so-called providers which allow deployment into specific environments. Providers are available for Docker, Kubernetes, SSH, AWS, Google Cloud, Azure and Digital Ocean. However, it is possible to write your own provider. These providers take a devcontainer.json config file and create a container (and if necessary a VM on which to deploy the container) based on the config in the selected environment.

An IDE backend can be deployed to the CDE container. VS Code Browser, VS Code with ssh backend and JetBrains IDEs with ssh backend are supported, meaning that deployment of their backends into the CDE container and configuration of connection to those backends can be done pretty easily.

Using standard Docker to deploy CDEs, the single-container CDEs are not intended to run multiple Docker containers or other system-level software within the container. The idea is again that the CDE is used primarily to run an IDE backend and any utilities required to enable debugging and linting, but not to actually deploy the full application that a developer works onunless it is a lightweight web app that can sensibly run within a single container.

The assumption is that there is a CI/CD pipeline that builds and deploys the application after the developer commits their code changes to the source code repository - once again leaving the developer stuck working with slow feedback cycles. (See: The problem with developing blindly)

I think this is a quite curious approach. It doesn't remove dependence on developers laptops. It also doesn't allow to provide CDEs centrally as a service to developers (though the devcontainer.json file can be provided centrally). It doesn't allow central management or visibility of CDEs. It relies on individual developers setting up and correctly configuring the DevPod app on their laptops, figuring out how to connect to a remote environment, correctly using a provider, and then deploying CDE containers.

DevPod is best for:

- Work on small, lightweight projects such as webapps
- Individual developers who work on different projects and want to ensure separation of projects from each other
- Small development teams working with different operating systems on their laptops, struggling to deploy IDE backends to these different laptops
- Developers who want to deploy a single-container CDE both locally on their own laptops or remotely to a cloud environment, e.g. because debugging locally is slow due to resource constraints but needs to be possible because the developer doesn't always have access to an internet connection

Product website: https://devpod.sh/
Docs: https://devpod.sh/docs/what-is-devpod

DevZero

Looking at their documentation, DevZero looks like an ambitious but still immature product to me.

Their idea is sound: The core of their product is a "CDE management plane", i.e. a component that allows to store CDE templates and use them to create and manage CDEs. This CDE templating, deployment automation and management functionality is the core of what CDEs in general provide, on top of standard compute resources and integrations with existing tools.

The management plane is connected to one or more Kubernetes cluster, in which the CDEs run. CDEs are called "workspaces" and consists of a cluster, a DevBox (not to be confused with Microsoft Dev Boxes of Devbox by Jetpack), which looks equivalent to the CDE container in other setups, and any other resources that are defined as part of the workspace in their custom yaml configuration format called "recipe" (not to be confused with Ansible recipes).

Put simply: You get a CDE container that runs alongside other resources (i.e. containers or pods) in a Kubernetes cluster.

However the way they go about it is a bit strange. The management plane is always provided as SaaS. Customers can choose to deploy CDEs in their own infrastructure, but then they have to provide root access to their infrastructure (e.g. a cloud account) to the management plane in order to allow it to create resources within their infrastructure. There is currently no option to also host the management plane on-premise.

DevZero also has a few known issues which they will probably solve at some point, but mean some inconvenience for current users:

- Templating of CDEs is done via a custom yaml configuration format they call “recipes” (not to be confused with Ansible recipes). These recipes can only be defined in a web UI, where they are also stored and versioned. This means that recipes are not kept in yoursource code repository together with your source code, but are contained within the DevZero web UI and can only be accessed there. As a custom configuration format, it also requires some learning to figure out how such recipes are defined - though they now promote an AI assistant that supports the creation of recipes (haven’t tried it).
- CDE containers are hardcoded to use an Ubuntu:22.04 base image. The documentation states that custom base images are available only to enterprise customers, suggesting that base images are somehow managed manually by the DevZero team.
- CDEs cannot be stopped / hibernated manually, and it is not possible to customise hibernation logic. Once a workspace is “inactive” (no inbound connections) for more than 30 minutes, it will automatically be hibernated. Workspaces will automatically restart when there is an incoming connection. But: Only one hard-coded directory (/home/devzero) is persisted for hibernated workspaces. Any content outside of that directory is deleted on hibernation!

DevZero is best for:

- Hard to say! I would say for companies that develop Kubernetes-based products, but for those, Okteto looks like a more mature commercial offering (though DevZero might be cheaper), and Eclipse Che is already the go-to choice for open-source tinkerers working with Kubernetes.
- Possibly for companies that are looking for a small vendor that may provide personalised support and services, though that is hard to judge from their website.

Product website: https://www.devzero.io/
Docs: https://www.devzero.io/docs/

Eclipse Che

Eclipse Che is the oldest CDE Open Source project. It started as a web IDE and developed additional CDE features from there.

It is Kubernetes-based and configuration of the development container is done via Devfile. Eclipse Che CDEs are Kubernetes pods. Several containers can be deployed to these pods. This means that multi-container applications can run within an Eclipse Che CDE, with one development container running alongside the application containers.

It supports VS Code, JetBrains and Eclipse Theia browser IDEs.

While a great project, talking to engineers who have worked with it I have heard that it can be difficult to set up and get to work properly, and requires quite a bit of effort in continuous maintenance as well as expertise in Kubernetes.

Eclipse Che is best for:

- Dev teams working on products that run in Kubernetes in production
- Organisations with dedicated platform engineering (or similar) team(s) with Kubernetes expertise who have the resources to manage Che as a service for their dev teams
- Organisations with the inclination and resources to support Che internally, without relying on support from a vendor

Docs: https://eclipse.dev/che/docs/
More info: https://github.com/eclipse/che

GCPWorkstations

Google Workstations are pretty standard container-based CDEs. Workstations run in a Google Cloud compute cluster. Each workstation is a separate VM in which a single Workstation container is deployed.

Workstations are built from a container image. If and where this container image is built is up to the user, though there is documentation to support users on how to set this up. Nevertheless, building the container from a configuration file is not part of the Google workstation CDE offering.

Additional customisation, such as the machine type to be used, is done via the Google Cloud console web interface or via the Google Cloud API or CLI. The configuration is stored in a proprietary, but fairly standard and simple configuration format called WorkstationConfig.json.

Curiously, users will quickly encounter a “nested virtualisation” option, which allows users to deploy VMs within their workstation container.

Google mentions in its documentation that this comes with severe performance penalties of 10% or more. No wonder: You'd have a VM that runs a Docker container which emulates running of another VM inside the container. That's a few layers too many to provide sensible performance or experience.

As customary for Google products, the documentation is very good and there are a large number of examples and templates available.

Like all CDE products from cloud infrastructure providers, Google workstations are only available in Google cloud.

Google workstations are best for:

- Organisations that already use Google Cloud to run their application in production
- Organisations that already use other Google Cloud products in their development processes, such as Google Cloud Build and the Google container registry
- Organisations that have people with experience and knowledge of Google Cloud, who provide workstation configurations as a service to developers

More info: https://cloud.google.com/workstations?hl=en.
Docs: https://cloud.google.com/workstations/docs/

Github Codespaces

Github Codespaces has the huge advantage of being tightly integrated and offered directly via github, therefore addressing a large user base which can use it for up to 60 hours a month for free. It is only available as SaaS and cannot be self-hosted. It can also only be used with Github and not with other source code repositories.

Github Codespaces is best for:

- Individual developers or small teams who host their projects in Github
- Developers working on open source projects that are hosted on Github

Docs: https://docs.github.com/en/codespaces/overview
Product website: https://github.com/features/codespaces

Gitlab Workspaces

Gitlab workspaces are the CDE product of Gitlab. It was launched in Beta in June 2023 and reached general availability in January 2024.

They are pretty standard container-based CDEs which are configured using a devfile.yml which is stored alongside source code in a Gitlab repository. CDEs are managed directly via the Gitlab web interface and can be spun up for any Gitlab project.

The workspaces CDEs are bundled with the Gitlab platform and can be used as SaaS or self-hosted, but only in combination with the rest of the Gitlab platform. CDEs can only be used with the Gitlab IDE, which is a VS Code fork.

The CDEs run as pods in a Kubernetes cluster. This means, hosting Gitlab on-premises means running a Kubernetes cluster as well.

As a very new feature, Gitlab workspaces still have a fairly limited feature set and quite a few known issues. Built as a feature of the Gitlab platform, it is also not intended to be used as a standalone CDE product.

Gitlab Workspaces are best for:

- Organisations that already use Gitlab as a central tool in their development workflow
- Organisations with lightweight containerised applications whose deployment can be expressed in a single docker-compose file

Docs: https://docs.gitlab.com/ee/user/workspace/

Gitpod

Gitpod is one of the most mature and popular CDE products currently on the market. It established the technological blueprint that most other CDE products follow: CDEs are containers that are configured to mimic VMs, meaning that any type of workload can run within them, including system-level software like Docker or Kubernetes. To a developer, working in such a container feels similar to working in a VM or on a laptop. For the CDE vendor, this architecture has the advantage of allowing better isolation of CDEs from the vendor's infrastructure.

In addition, it allows Gitpod to host their CDEs in Kubernetes, allowing for flexible scaling and efficient use of compute resources, which is necessary to be able to offer it as a SaaS product with a reasonable price point.

However, using Kubernetes as a basis for their product means significant operational overhead for running a Gitpod instance, which is why Gitpod discontinued their self-hosted option in December 2022 and let go of the engineering teams that were working on this.

One of their engineers started a project to continue supporting self-hosting Gitpod, but gave up in April 2023 when Gitpod stopped publishing containers in publicly available registries.

While ostensibly open source, removing access to vital resources to enable deployment of a self-hosted Gitpod instance, Gitpod cannot be used free of charge.

I think there are two reasons why Gitpod discontinued supporting a self-hosted option. First, they could not monetise it. Second, they were not able to provide a good experience for it (which is the reason they publicly stated).

Gitpod tried to turn this into an advantage by switching to a “self-hosted, vendor managed” model. In this model, a customer gives Gitpod full access to an AWS account within their VPN, and Gitpod deploys and manages a dedicated Gitpod instance for the customer in this AWS environment.

While this might be a sensible offer for enterprise customers with hundreds of developers, for anyone else this is a very expensive option that they will struggle to utilise to an extent where they achieve a sensible ROI. The cost and operational overhead of running Gitpod becomes cost effective only for very large numbers of developers.

For anyone else, Gitpods SaaS offering is a good entry point for individual developers and small teams, though it can also become quite expensive when developers need more than a 2-core machine.

Gitpod is best for:

- Devs working on simple to medium-complexity software with low resource requirements
- Individual developers and small teams looking for a SaaS option
- Very large development teams looking for a vendor-managed solution that runs in their AWS environment

Product website: https://www.gitpod.io/
Docs: https://www.gitpod.io/docs/introduction

IDX by Google

Initially an experiment - when a single-page website announced it in mid-2023. Now, in mid-2024, it has graduated to “project IDX” and is in public beta. It is a CDE product, but one that focuses on mobile app and web development with a special focus on Flutter projects. As such, it features android emulators and web previews as core features. It also heavily promotes its integration with Google’s Gemini code assistant.

Developers access IDX purely in the browser, with a VS Code fork available as IDE. CDEs are virtual machines which, obviously, run in Google Cloud.

It is the first CDE product I’ve seen that is based on the nix package manager (https://nixos.org/). CDEs are templated via a nix configuration file.

With a focus on web and mobile apps and built-in emulators, IDX is built to run lightweight web and mobile apps directly in the CDE. However it is clearly not built for large-scale application development of business or server software.

IDX itself is currently free to use, though users are charged for the compute resources they use on Google Cloud as well as any other Google services they use through IDX. I assume that a pricing model will be communicated once the project leaves beta, but I personally think it is likely that there will be a free tier also in the future.

IDX is best for:

- Individual developers looking to experiment with AI-supported web or mobile development
- Individual developers or small teams starting a new web or mobile project

Product website: https://idx.dev/
Documentation: https://developers.google.com/idx/guides

JetBrains CodeCanvas

JetBrains is primarily known for its IDEs, which are integrated with a lot of CDE products.

Its product strategy is not quite clear to me, since it has launched several closely related products with partially overlapping feature sets that compete with products of their partners. First, JetBrains launched their Space product, an “intelligent code collaboration platform”, which had some limited CDE features. Now, its CodeCanvas product, launched in April 2024, is branded as a full CDE product. In the meantime, JetBrains is pursuing partnerships with several CDE-vendors who have integrated their products tightly with the JetBrains IDEs. As such, I’m personally unsure if and how their product roadmap will develop and if a CDE product has a fixed place in it.

CodeCanvas is described as a CDE management platform, through which single-container CDEs can be created, started, stopped and deleted. CDEs run in a Kubernetes cluster and are intended to run a JetBrains IDE backend. While it might be possible, it is not an intended use case of CodeCanvas to run the full application within the CDE.

Developers connect to the CDE with JetBrains Gateway, a desktop application that allows users to locally connect to JetBrains IDE backends.

CDEs are configured via a custom template that is created in a web UI. It contains reference to a dockerfile which describes the CDE container (single container).

The (JetBrains) IDE is also fixed at the template level. Notably, the template also contains resource specs and the git repository and branch, meaning that developers really only have to push a button to get started and don't have to specify any additional configuration (unless they want to deviate from the template).

JetBrains offers CodeCanvas as a self-hosted product that customers can run in their own cloud accounts. It requires a Kubernetes cluster and is currently compatible with Amazon EKS, Azure AKS and Google GKE.

JetBrains CodeCanvas is best for:

- Organisations that already use JetBrains IDEs and other JetBrains products in their development workflows
- Organisations who do not want to deploy their applications within the CDE (which, in my opinion, removes the main benefit of a CDE. Finding a place to run an IDE backend is not the challenge that CDEs need to solve.)

Product website: https://www.jetbrains.com/ide-services/codecanvas/
Documentation: https://www.jetbrains.com/help/codecanvas/introduction.html

Microsoft Dev Box

True to form, Microsoft is the only vendor brazen enough to brand a remote desktop product as a work environment for software developers. Considering Microsoft owns Github, it makes sense it doesn't want to directly compete with the Github Codespaces CDE product (though it still does that a little bit through its reference implementation of VS Code Remote with devcontainers).

DevBoxes are Windows machines that users connect to via the Microsoft Remote Desktop app or a browser. They are hosted in Azure and only available in the cloud.

Microsoft addresses the issue of latency which makes remote desktop working unattractive for most people by automatically deploying Dev Boxes to the closest available Azure region. I can't say if that leads to good experience, but for Microsoft-focused developers of desktop applications, it might be a good alternative since container-based CDEs are simply not suitable for development of Windows Desktop applications or fat clients.

Since Dev Boxes are Windows machines, any Windows software can run on them. Configuration of the machine specs as well as management of the Dev Boxes is done via a web GUI. Customisation of the Dev Box can be done by specifying a custom Windows image to be deployed to the Dev Boxes. This means the Dev Box configuration cannot be done as code but only through manually prepared images.

Microsoft doesn't list specific IDEs which are available in their Dev Boxes, but I assume it will be any IDE that can be installed in a Windows environment. There is a specific section in the documentation explaining precaching of resources with Visual Studio (not VS Code) to improve performance after starting a Dev Box.

Microsoft Dev Boxes are best for:

- Organisations that develop Windows Desktop applications

More info: https://azure.microsoft.com/en-us/products/dev-box
Docs: https://learn.microsoft.com/en-us/azure/dev-box/

Okteto

Okteto provides CDEs for multi-container applications that run in Kubernetes. The Okteto platform deploys a dedicated development container - the CDE - next to one or several containers for the application that is being developed.

Deployment of the application is based on an Okteto manifest, which is based on helm charts, kubernetes manifests, or docker-compose files which the user has to supply. Once this is set up, Okteto redeploys the application with each commit.

Because Okteto provides deployment automation for applications into Kubernetes, it doesn't primarily position itself as a CDE vendor. But their deployment automation is not intended for production deployment and doesn't replace a full CI/CD pipeline (it serves the "inner loop" needs of development teams). Their main claim is to improve developer experience for working on Kubernetes-based products.

Okteto as a company was founded in 2018, but version 1.0.0 of Okteto was only released in 2022.

Okteto is best for:

- Dev teams working on software that runs in Kubernetes in production
- Organisations that are looking for a commercial vendor to provide support
- Organisations looking for a SaaS offer or a supported self-hosted option

Product website: https://www.okteto.com/
Docs: https://www.okteto.com/docs/

Red Hat OpenShift Dev Spaces

Previously called Red Hat CodeReady, Red Hat OpenShift Dev Spaces is a commercial offering for Eclipse Che.

Red Hat offers it as a standard part of its OpenShift subscription, as a way to deploy development containers alongside application containers into OpenShift.

Red Hat OpenShift Dev Spaces is best for:

- Organisations that use OpenShift and run their production environments in OpenShift

When researching OpenShift, you will quickly come across Codenvy, which is part of Eclipse Che (this Eclipse newsletter from 2015 explains it best).

Product website: https://access.redhat.com/products/red-hat-openshift-dev-spaces or https://developers.redhat.com/products/openshift-dev-spaces/overview
Docs: https://docs.redhat.com/en/documentation/red_hat_openshift_dev_spaces/3.14/

Strong Network

Strong Network is a security-focused CDE product. It is only available as self-hosted and has a lot of features related to air-gapping the CDEs so that no data can leave the CDE undetected.

Strong Network also has a lot of “observability” features as well which allow administrators to snoop in great detail on what the developers are doing on the CDEs.

The CDEs feature a fine-grained permissions concept (role-based), which looks fairly complex to manage.

It is also a classical container CDE, which can work with any browser-based or ssh-capable IDE, notably VS Code and JetBrains IDEs. The containerised CDEs run in a Kubernetes cluster. Customisation of the workspace is done via a web portal where limited configuration options for workspaces are available.

My impression of Strong Network is that it focuses a lot less on developer experience and a lot more on the goals of management than other CDE vendors.

The product documentation as well as their website in general give the impression that Strong Network probably has close relationships with their customers, support them individually a lot, and don’t focus on providing information to the public, since their product is not intended to be used in self-service anyway. They do not publish release notes and I was not able to find out when their product was initially released. The company was founded in 2020, so I assume that they left beta sometime in 2021 or 2022.

Strong network is best for:

- Organisations with a strong security focus that want to track their source code closely

Product website: https://strong.network/
Docs: https://docs.strong.network/getting-started/

Discontinued CDE products

Hocus

Hocus had a short but interesting life. It was founded in early 2023 and wound down again in November 2023. Focused on providing memory-efficient CDEs that allow to run kernel-level processes on a CDE with clean separation of individual CDEs from each other, they got down to the very nitty gritty details of virtualisation technologies and their limitations (see https://hocus.dev/blog/).

I don't know if they were not able to solve it to their satisfaction, were intimidated by the competitive landscape or simply ran out of money - whatever the reason, they stopped development in November 2023.

Product website: https://hocus.dev/
Docs: https://hocus.dev/docs/intro

Koding

Founded in 2011, Koding was one of the earliest CDE projects out there. The company was wound down in 2018, the project was open-sourced, but the repository was archived (but is still available as read-only) in 2022.

Koding developed its own configuration format to define development environments as code. Based on these stack scripts, VM-based CDEs are deployed.

More info: https://www.koding.com/
Git repo: https://github.com/koding/koding
Docs: https://hocus.dev/docs/intro

Nimbus

Nimbus was a VM-based CDE platform with a fairly manual approach. VM snapshots were used as CDE templates, meaning that CDEs were not defined in code but as manually prepared snapshots.

Even though it is not clearly stated on their website, Nimbus was discontinued in mid-2023.

Product website: https://www.usenimbus.com/
Docs: https://docs.usenimbus.com/

Honorary mentions

In this section, I describe several CDE-adjacent tools. I think it is important to include them because

- some of them are sometimes included in lists of CDE tools though they differ in fundamental aspects from my definition of a CDE tool. Here, I explain what they do and why I do not classify them as CDE tools, and
- because some of them may be good solutions for some development teams who might not need a “full” CDE.

Stackblitz

Stackblitz is currently focused on Node.js projects. It prides itself on being extremely fast, with CDEs being available within seconds. The core innovation of StackBlitz is its usage of WebContainers, which move all the compute to the browser. Having a Node.js runtime in the browser means that you can run and debug Node.js backend code also in the browser.

This makes it not really a CDE, since it does not run in the cloud but rather on your machine, but within your browser. The benefit of this is that no internet connection is required to use this browser-based (C)DE. Therefore, I see it more as a different type of IDE rather than a CDE.

Docs: https://developer.stackblitz.com/
More info: https://stackblitz.com/

JetBrains Space

In the previous version of this whitepaper, I listed JetBrains Space as the CDE product of JetBrains. In the meantime, their focus has shifted, with Space growing into a “platform for the entire development pipeline”. Part of the Space offering is a compute environment for the IDE backends of the JetBrains IDEs - but beyond that, JetBrains space provides no CDE-specific functionality.

Instead, JetBrains has started to collaborate with CDE vendors to provide tight integration between their products. To this end, JetBrains is developing Gateway, a desktop application that can be used to connect to any CDE with the help of CDE-specific plugins, which can be provided by different CDE vendors.

Plugins are currently available for GitHUb Codespaces, Gitpod, Google Cloud Workstations, and Coder. Connections can also be set up using only ssh, which allows developers to connect to any remote environment that runs a JetBrains IDE backend.

To make things more confusing, JetBrains has also introduced yet another product called CodeCanvas in April 2024. It is described as a self-hosted remote development environment orchestrator.

Product website: https://www.jetbrains.com/space/
Docs: https://www.jetbrains.com/help/space/getting-started.html

Devbox by Jetpack

Not to be confused with Microsoft Devbox.

Devbox is based on the Nix package manager and provides isolated Nix packages for development environments.

By creating CDEs as packages, the level of isolation is different from a container or a VM: an isolated package contains all dependencies the package needs to run, but shares the operating system with the rest of the environment. By isolating the packages, Nix makes it possible to remove dependency conflicts that arise from different software in the same environment needing different versions of some other software.

Configuration of the Devbox environments is done by customising existing Devbox templates through a CLI. Configuration is stored in a devbox.json which is intended to be committed to the source repo, where other developers can then deploy the same Devbox with the same config. Adding custom scripts e.g. to deploy your own software is also possible.

CDEs as packages are an interesting idea: it makes it very simple to distribute the packages. Developers simply install a package (based on a config file they get from their source repo) with a package manager like they do for many other things as well.

However, Devbox addresses only the issue of dependency conflicts - which the Nix package manager really does for them. What Jetpack does is to provide templates for dev environments that can be used with the Nix package manager. It doesn't provide tooling for remote development, and therefore doesn't qualify as a CDE.

It's not a cloud or remote development environment. Developers install CDE packages locally. They could be installed anywhere, but then the issue of managing and connecting to these remote CDEs is the problem of the user. This means that developers are stuck with the performance they get locally.

Product website: https://www.jetpack.io/devbox
Docs: https://www.jetpack.io/devbox/docs/

Where CDEs are valuable - and where they are not

Complexity of local development environments is high

E.g. for teams that work on complex, multi-component, heavy duty software with complex and / or non-standard deployment models. Many CDE products focus on lightweight applications with simple deployment models, such as webapps. In my opinion, such CDE products have only marginal benefits because local deployment of such apps is already simple and there is no big pain that such CDEs can address. I wrote more about this in our blog.

Complexity is effectively moved away from developers

and working automation for CDE deployment is provided as a service to them. There are two ways in which some CDE products fail to support this:

#1
CDE products that are installed on local laptops and don't support centralised CDE management and templating. While such products claim to empower developers by putting them in charge of CDE management, in the end they once again leave individual developers to deal with a lot of the complexity that is involved in configuring and deploying CDEs.

#2
CDE products that support only partial templating. Such CDE products require developers to conduct manual setup steps after the CDE has been created. This leaves room for manual error, once again saddles developers with having to know how to correctly set up their environments, reduces standardisation and - worst of all - incentivises reusing the same CDE over a long period of time, leading to the inevitable rise of pet CDEs that are not standardised and hard to maintain. Unfortunately, the limited nature of the existing CDE configuration standards (devfile and devcontainer) means that CDE products relying primarily or only on those standards for CDE templating often fall into this category. I write in our blog about the limits of devfile and devcontainer as standards for CDE configuration and explain why so many CDE vendors choose to use their own configuration format despite the existence of these standards.

Developers gain from using CDEs

E.g. if CDEs allow them to do things they are not able to do locally, such as running a build and deploying the application in their CDE, where they have full visibility and access. There are again two types of CDE products that provide only limited benefits to developers:

CDEs that don't aim to support building and running the application as part of the CDE at all. These are CDEs that only contain an IDE backend and the source code. These are the parts of the local development environment which typically work well anyway. There is not a lot of pain associated with running an IDE. On the other hand, being unable to locally build and deploy the application that a developer works on means forcing them to work blindly. I wrote about the problem with developing blindly in our blog as well.

Products that primarily deploy CDEs on developers laptops, where developers are stuck with limited compute resources.

Under these circumstances, CDEs will be most valuable to developers. Organisations will also benefit most in terms of time saved for developers, as well as quality improvements achieved through better standardisation of development environments.

This does not consider organisational motivations to use CDEs, such as improving source code security or making it easier to collaborate with external developers.

How to choose a CDE product

In my opinion, the most important factors to consider are two:

1. What are your current challenges and does the CDE product help you fix it?
2. What do your production deployments look like?

Answering the first question is often made difficult by the lack of comparable information available for different CDE products. In my experience, the first step is to get a list of potential products - like the one here in this whitepaper- and then to start crossing off those that definitely do not fulfill my requirements. The remainder is hopefully a short list that can be researched or tested in more detail.

The second question is important because a CDE product can be a great opportunity to get development and production closer together. For example, if your software runs in Kubernetes in production but developers don't want to run Kubernetes clusters locally and therefore use only Docker to run their application when developing, moving to a Kubernetes-based CDE can be a big step towards aligning development and production deployments without increasing the mental load on your developers. On the other hand, if your production deployments run on VMs, then you should probably choose a VM-based CDE.

Summary

Cloud Development Environments aim to solve the following problems faced by developers:

- A lot of time is invested in maintaining development environments
- A lot of knowledge is required to set up and maintain development environments
- A lot of computing power is required to build, test and run the software that developers work on

CDEs propose to solve this by providing development environments which are:

- Simple to set up and use - reducing the need for both time and knowledge required for set up and maintenance of development environments
- Run on powerful servers - reducing the load on local workstations.

I find it important to repeat these central points because:

- Not all developers face these problems. Many developers work with mature techstacks for which great tooling is available to run them locally without a lot of maintenance or deep knowledge required. A lot of software doesn't require a lot of computing power. For such developers, the main value propositions of most CDEs don't make sense.
- Not all CDE products (and CDE-adjacent products) solve all three of these problems.

Therefore, when looking for a CDE product, you should be aware which problems you are actually trying to solve, and if the products you're looking at actually solve them.

Start with Cloud Development Environments from Cloudomation.

Contact us

A brand of Starflows OG
Darnautgasse 6/6, 1120 Vienna, Austria

+43 699 10 69 5915
www.cloudomation.com
info@cloudomation.com

Supported by the Vienna Business Agency.
A fund of the City of Vienna.