KODIBOX

PRODUCTION-LIKE ENVIRONMENTS ON DEMAND

KODIBOX

Contents

About the product briefly

- Production-like, repeatable environments with a full set of necessary functionality for
  - software development,
  - testing,
  - preview.
- An easy way to deploy these environments.

KODIBOX is a cloud-based platform for improving the efficiency of software development in multicomponent applications.

Team members get to create, build, and test their code, service, or part of a service whenever and however they want, in a separate, isolated environment.

The Problem

In any company, the complexity of software development increases over time.

The faster you grow, the

- Your ecosystem becomes more fragmented and complex
- The rate at which new functionality is released to the market is falling
- The organization of work becomes more complex
- Control over the system is lost

Thus, team members have "parasitic" activities, "crutches" and "rakes" appear in the project/application. The work is no longer transparent. As a consequence, teams suffer and the user suffers.

Bottom line for business:

- OPEX grows
- Time to Market grows

Problematics

Most companies close this issue through:

- Formation of processes, instructions, regulations,
- Hiring additional management staff
- allocation of additional technical, financial and human resources
- etc.

From time to time it has some effect, but the problems recur...

Our Solution

In our opinion, the solution lies at the level of organization of team interaction.

One possible approach to solving these problems is to use a consistent configuration for each environment, from development to testing.

We provide:

- Production-like, repeatable software development environments with a full set of required functionality, coordinated between team development processes.
- An easy way for developers to deploy these environments for development and testing.
- Ability to run the same integration tests during development

The environment works exactly as long as necessary. When it is not needed, it "dies" without wasting extra resources and money.

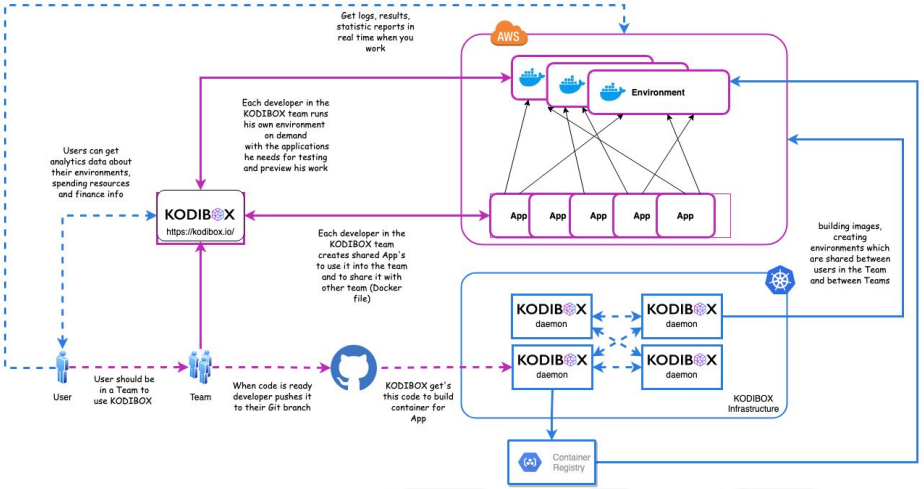
How does it work?

You describe your service's technology stack configuration in a declarative YAML file with all the necessary dependencies. The file is located in the repository, next to the Docker file and the source code of your service. The configuration files provide a consistent way of describing the required environment in **KODIBOX**.

The environment can be:

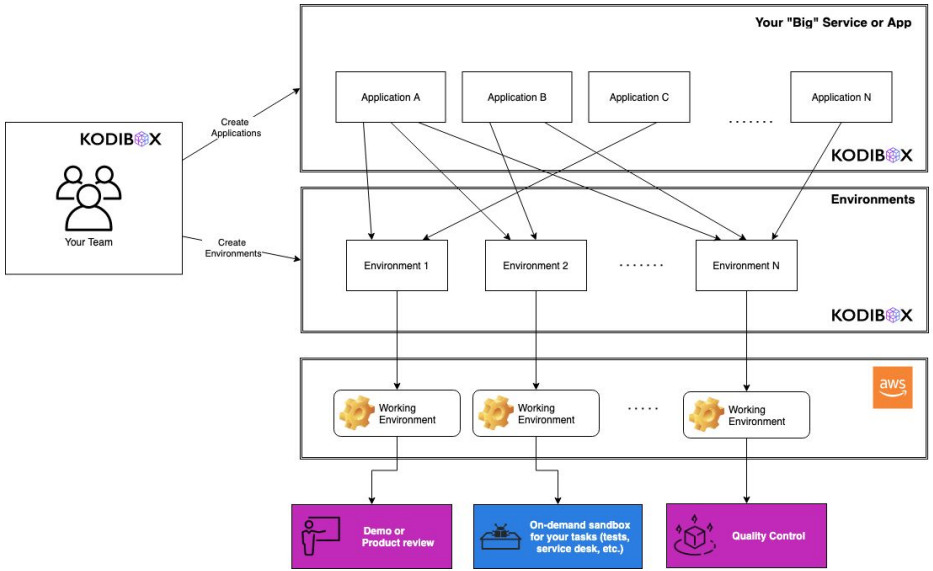
- Completely isolated;
- Being on the platform (i.e., using only cloud resources);
- Can be integrated with your local resources.

KODIBOX platform instead of the developer analyzes the information from the file and automatically does all the necessary work to prepare and deploy the environment infrastructure for the task or for a particular team.



Use Case #1 - Multicomponent Service

Your service may consist of many components/applications, each with its own development lifecycle. So you can test and validate your application regardless of whether your test environment is currently free.



The **KODIBOX** platform will deploy your environment in the cloud at your request for testing, demonstration or other task and take care of removing the environment itself, without distracting you with this "trivia".

KODIBOX will help you build your environment in the right configuration, with the right components. Especially when the components of your system become dozens or hundreds and the combinatorial complexity of reproducing the environment in a different mode increases manifold.

You don't have to waste valuable time of other developers or DevOps engineers. And it doesn't depend on what role you play on the team or what your technical background is.

Use Case #2 - feature development

You have a system being developed (project) in your company - one, two or more. It does not matter for whom and for what the system is being developed: for the company's own needs or to order.

Two teams are working on the project. Each team develops specific components of the system:

- Ficha 1 - Team 1
- Ficha 2 - Team 2

Feature 1 and Feature 2 are two components of the same project. At the end of the development these components must be integrated.

During the development of Fiches 1 and 2, teams and team members face the following questions:

- How do you deploy infrastructure?
- What is the development process?
- How and where to deliver the result of their work?
- Where is the knowledge about the project stored?
- How do you share your work with other team members?

Each project has the following "knowledge:

- Documentation;
- How-to;
- FAQ about problems and solutions.

At some point the teams need to integrate with each other and they have the following questions:

- What is the current state of the application and the relationship between the services?
- How do I connect Feech 1 to a Feech 2 assembly?
- What dependencies do Phicha 1 and Phicha 2 have?
- How to conduct joint software development?
- Where is the documentation of the fic?
- How do I monitor performance and debug?
- How do I test features?
- How not to break the environment on which to work?

And hundreds more questions.

With KODIBOX, team members no longer worry about anything but their work. Thanks to isolated builds, they don't depend on each other and can always start over again. The platform takes care of all the overhead itself: deployment, monitoring, logging, tracing, coordination, dependencies, cost accounting, etc.

Use Case #3 - Onboarding

You are hiring a new developer or bringing in a team to expand development.
What issues do you need to consider to ensure the effective and quick inclusion of new people?

The current state of the application and the relationship between other services. If a developer is going to write services that are part of an existing application, he needs to know how each service was created, tested and deployed. Perhaps this is described in a bunch of unofficial READMEs that no one officially owns, and therefore no one updates them. Or maybe it lives only in the heads of individual "unique" people on the other side of the country.

How to communicate the architecture of the application and the relationship between services to other team members. If a developer creates new services (or builds an application from scratch), how should they document it and bring the rest of the team up to speed so that everyone knows how to properly test and troubleshoot?

Dependencies with related services that the developer needs to know about and manage during development. It's also important to know who to contact if there are any questions or if someone accidentally breaks their service.

How does the developer share his work so that it can be reviewed by team members. Some features need to be tested and tested manually. How can a developer share a working version of a test application so that it works for everyone else just as it does for him?

In KODIBOX, each component of your application is described in its own configuration file: how it is built, tested and deployed. No massive project-wide files. Developers only care about their configuration files.

You can create environments with different names for different purposes: testing, previews, demonstrations, experiments, etc.

Developers can run a set of integration tests or share the environment within the project or with their team to ensure the same test result for all team members.

Use Case #4 - Product Manager

You are a Product manager or a Project manager.
As you develop the application and interact with the development teams, you need to understand as best you can what is done, what quality it is, and what you need to change to make the product successful.
This information is also important when interacting with users or customers to meet expectations and deliver quality results.
To do this, you need to do a number of things periodically.

In KODIBOX, product and project managers can easily preview a working version of an application and get an idea of how it behaves. Conduct quality control by simply deploying a working environment to preview the application. These environments are particularly useful for collaborative work outside the development team. They are ideal for product managers or other non-development stakeholders.

Verification of the implemented functionality.

Your team is in the process of developing a new feature or a new application. You want to get an idea of how it behaves in reality quickly and without delay, so that you can make the necessary changes in time without causing delays in deadlines or wasting valuable time and money.

Product demonstration

You want to demonstrate the finished part of the work to the customer, and preferably no one from the development team interferes with it.

Acceptance testing

You want to make sure your application works properly before showing it to a customer. But you don't want to stop the team and freeze development for a few days.

Hypothesis testing

You want to do an experiment or, in development, you want to make a small change to the product and see how users react. But you don't want to take up the team's time to deploy additional capacity for your idea.

Testing of mobile and web applications.

You only need to test the frontend part of your application (e.g. the new button and notification output) and you don't want to distract valuable backend development and limit testbeds to deploy a specific backend version for this task.

Challenges and opportunities

Designing

- How does the service being developed fit into the current architecture?
- How does the service being developed fit into the current infrastructure?
- What standards should be upheld?
- What processes are in place to get started?
- Which teams will be involved in development?
- How will the teams interact in development?
- Will the product be passed from team to team?

Development

- How do you deploy infrastructure?
- How do I manage the configuration?
- How will the testing be organized?
- What resources do we have?
- How do you manage addictions?
- What is the development process (gitflow, etc.)?
- How do I check my change?
- How to deliver the result to the environment?
- Creating Pipelines in CI
- Build
- Security

Documentation

- Description of the service/project stack
- Dynamic interdependencies
- Wiki.
- Accounting for standards

Other applications

ServiceDesk

A sandbox for technical support services for quick and secure verification of problems and complaints from customers

Sales/Marketing

- Giving the potential customer access to a trial, isolated, version of the application/service being sold
- Deploy different versions for different audiences, different geo-locations
- A/B testing
- Designing your own service by the end user. The user chooses the desired functionality and starts using your application (if your software supports this architecture)

Education/Science

Creation of temporary environments for the tasks of the class, diploma, term paper, conducting experiments or other research.

Artificial Intelligence

Many studies show that on the horizon of 10 years there is a high probability that software will be written by neural networks as well as by humans. So the relevance of the integration problem will at least remain, but most likely become an even higher priority.

Results

For technicians

- "Distributed" approach to customization
  - makes it a suitable choice for complex microservices applications consisting of many different components.
- Each team can manage its configuration files independently of each other
  - and no one needs to deal with a single, complex, "centralized" configuration file
- Consideration of dependencies at any level of environment creation
- Ability to run the same integration tests during development.
- Simplify understanding of the application by generating "live documentation"
  - showing the relationship between all of its components, including dynamic components.
  - This is the kind of knowledge (contextual awareness) that is very difficult to convey otherwise.
- The knowledge of what needs to be done to raise the new service is stored in the system, not in the "heads" of a narrow group of people
- The developer gets immediate feedback and can solve the problem while he is in context and writing code.
- You can be sure that your tests will pass in the general pipeline.
- The platform allows you to make a complete description of your stack, including how it is created, deployed, and tested.
- Every developer knows the current architecture of the application and the relationship between the components - including the services they are not working on.
- Every developer can create environments that are identical to those used by their peers and as similar to production environments as possible. No more local environments that behave differently on each laptop, making it impossible to work cohesively.

For **Non-Technical** Professionals

- **Increase Capacity/Velocity for software development teams by 20-30%**
  - Reduced costs for onboading specialists
  - Reducing the cognitive load on developers
    - less context switching
    - Developers can focus on what really matters to the business
    - Engineers no longer need to know how deployment happens
  - Reduced loss of developer time for infrastructure tasks (**up to 15 hours per week**)
  - Reducing the inefficiency of multicomponent application development caused by their large size.
- **Understanding and managing the costs of development infrastructure**
  - The environment works exactly as long as it needs to. When it is not needed, it "dies" without wasting extra resources and money.
- Reducing the complexity of the configuration of the supported and developed application
- **Reduced automation costs**
  - support and configuration of CI/CD processes
  - less "bad" work with deployment
- Improving the manageability of the software under development
- **Quality control**
  - with a simple deployment, you get a preview environment for the application
  - These preview environments are especially useful for collaborative work outside the development team.
  - Product and project managers can easily watch a working version of the application and get an idea of how it behaves.
  - Ideal for product managers or other non-development stakeholders who need to do acceptance testing before release.

Results

For **business**

- Reducing transaction costs in software development
- OPEX reduction (up to 50%)
- Reduced Time to market (up to 50%)

- Your business becomes more efficient
- Your business becomes more productive

- You attract more customers
- You serve your current customers better

Profit

KODIBOX

Roadmap

Now	Later	Future
Our top priority. We're probably working on it right now or starting pretty soon.	Our next priority. We'll work on this soon if everything goes as planned	Not a priority. We're considering working on this but it's too early to know when.
Goal - MVP	Goal - Enterprise version	Goal - Cloud Version
Onboarding (Done)	**2FA, API, Mobile Version**	**Roles Management**
Manage applications, environments, commands, audit logs, SSL (Done)	**Environment Presets**	**Sharing artifacts between teams**
Integration with AWS (Done)	**On-premise deploy**	**SSO integration of SAML 2.0**
Integration with Yandex.Cloud (ToDo)	**Wizards (Onboarding, first setup, etc.)**	**Adding other payment methods**
Integration with Google SSO (Done)	**Secrets Management**	**Online Chat**
Auto-delete environments, Life Extension (In Progress)	**Reports and analytics**	**KODIBOX - Open Source**
Integration with custom Docker Registry and VCS (In Progress)	**Integration with Google Cloud Engine, Digital Ocean**	**Changing the tariff by the user**
Integration with messengers - Slack, Telegram (In Progress)	**Copying commands/environments/applications**	**Adding languages to work with the system**
Product Blog (In Progress)	**Integration with Facebook SSO**	**Notification Manager**
	Team Owner Change	**Deleting an account by a user**

You need KODIBOX when...

- You are developing a multi-component service or application
- Are you transitioning or are you already working on microservices.
- You have autonomous teams, you have distributed teams
- You have one team growing strongly
- You are growing and becoming more and more fragmented
- Your ecosystem is growing and becoming more complex
- You have long feedback loops in development
- You have a long integration test
- You have a problem with deployment of releases and component binding
- "Everything in development is getting slow."
- **You need as simple an understanding as possible of how things work**

- Expensive and difficult to do experiments
- You need to reduce the TTM
- You need to test different combinations of versions of End2End services
- You would like to reuse services in other systems, without recompiling (through variables)
- You need microservices versioning control

You have:

- >10 developers
- >5 services
- >2x duplicate configurations
- >10 assemblies per day
- You need to test it all, every time.
- >1 release per day

Who are we?

We're a startup and we believe that developers want to give back an uncomplicated, fast-paced development process in which they can focus on their work.

So we want to make development infrastructure *a standard service:*

- When you know ahead of time what you are going to get
- what quality it will be,
- How much time and money it will take you to test and get results
- and it doesn't depend on who is providing you with the service.

Contact

Konstantin Perminov
konstantin.perminov@kodibox.io
Co-founder, Chief Technical Officer
@devkp

Dmitry Votintsev
dmitriy.votintsev@kodibox.io
Co-founder, Chief Executive Officer
@dvotintsev

Sk Resident

Since September 2020,
Skolkovo residents.