Yandex Cloud

Infrastructure from Code: следующий этап развития IaC на примере Serverless

Виктор Кузенный,
руководитель разработки
Serverless-сервисов, Yandex Cloud

Виктор Кузенный

Руководитель разработки
serverless-сервисов,
Yandex Cloud

с 2023	Руководитель разработки Serverless в Yandex Cloud
2021–2023	Ведущий разработчик API Gateway в Yandex Cloud
2016–2021	Старший разработчик на Java в Кинопоиске
2008–2016	Аспирант, разработчик на Java в СПИИРАН
2003–2008	ВМиК МГУ

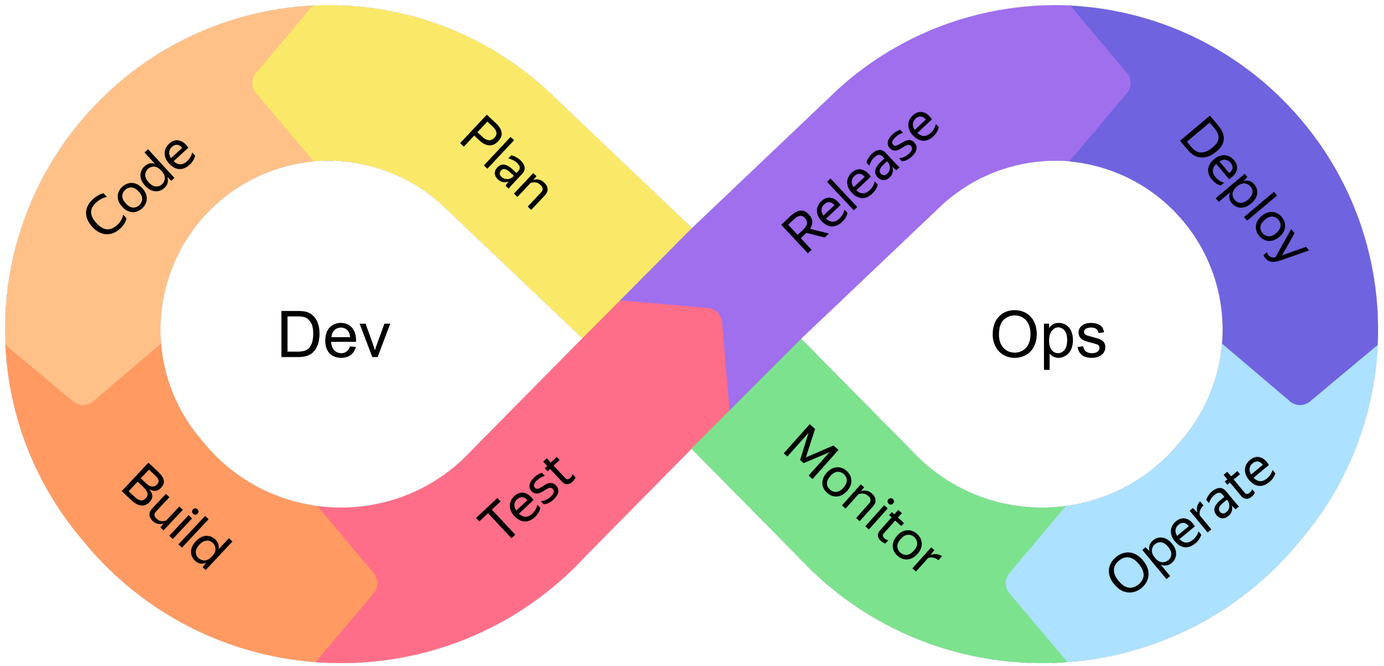
План доклада

1. Развитие Dev и Ops
2. Serverless
3. Развитие IaC: прошлое, настоящее и будущее
4. Обзор подходов, технологий и инструментов IfC
5. Демо на примере фреймворка Nitric и Yandex Cloud

1. Развитие Dev и Ops
2. Serverless
3. Развитие IaC: прошлое, настоящее и будущее
4. Обзор подходов, технологий и инструментов IfC
5. Демо на примере фреймворка Nitric и Yandex Cloud

Цель DevOps

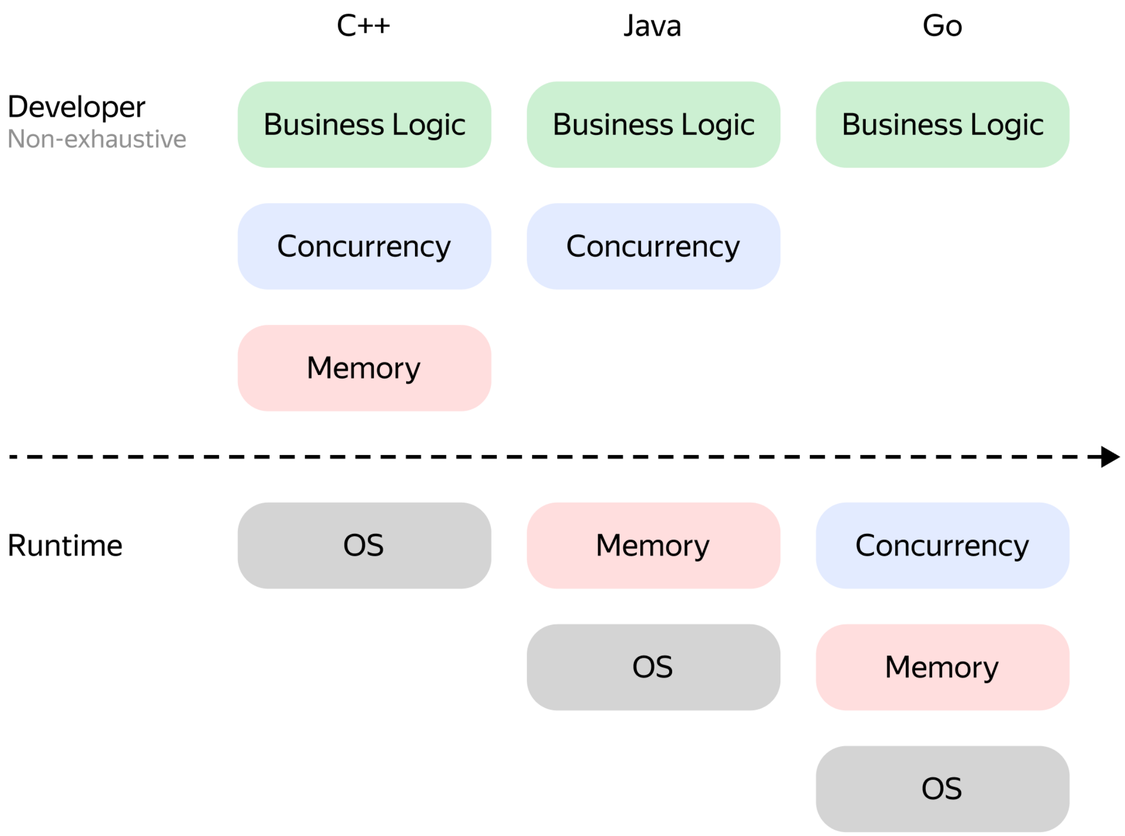
Just write business logic + Just deploy business logic



Развитие Dev

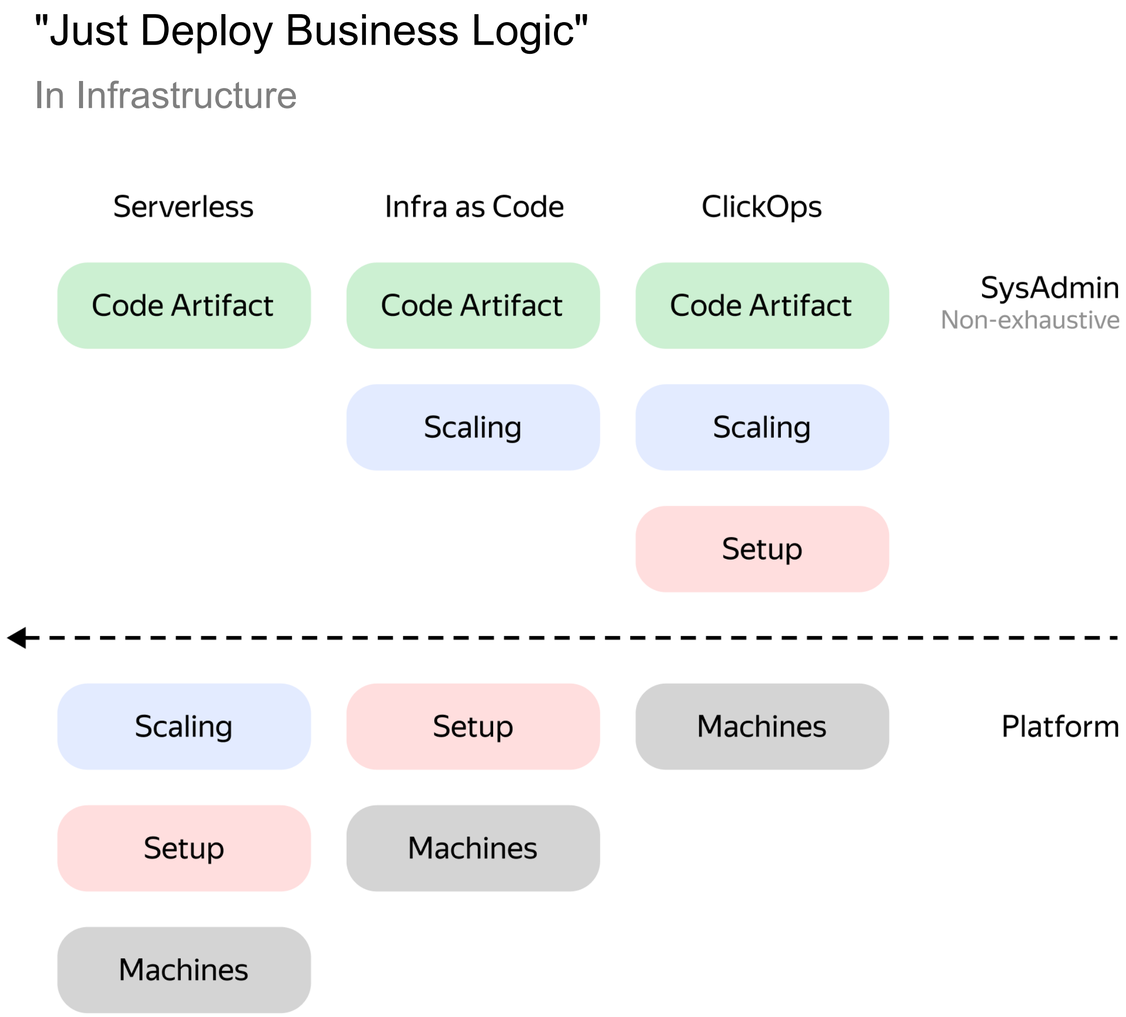
"Just Deploy Business Logic"

In Programming Languages

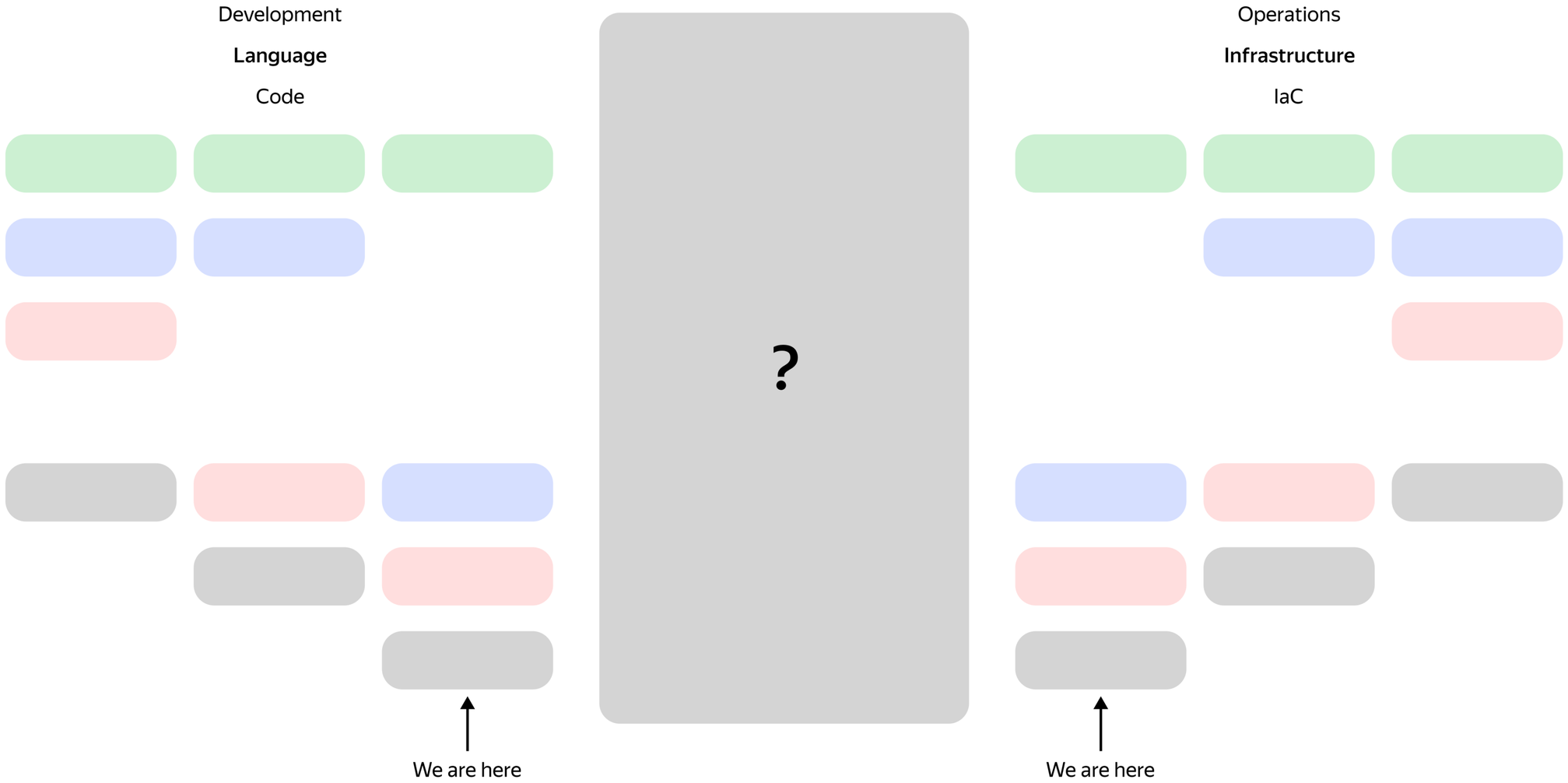


swyx.io/self-provisioning-runtime

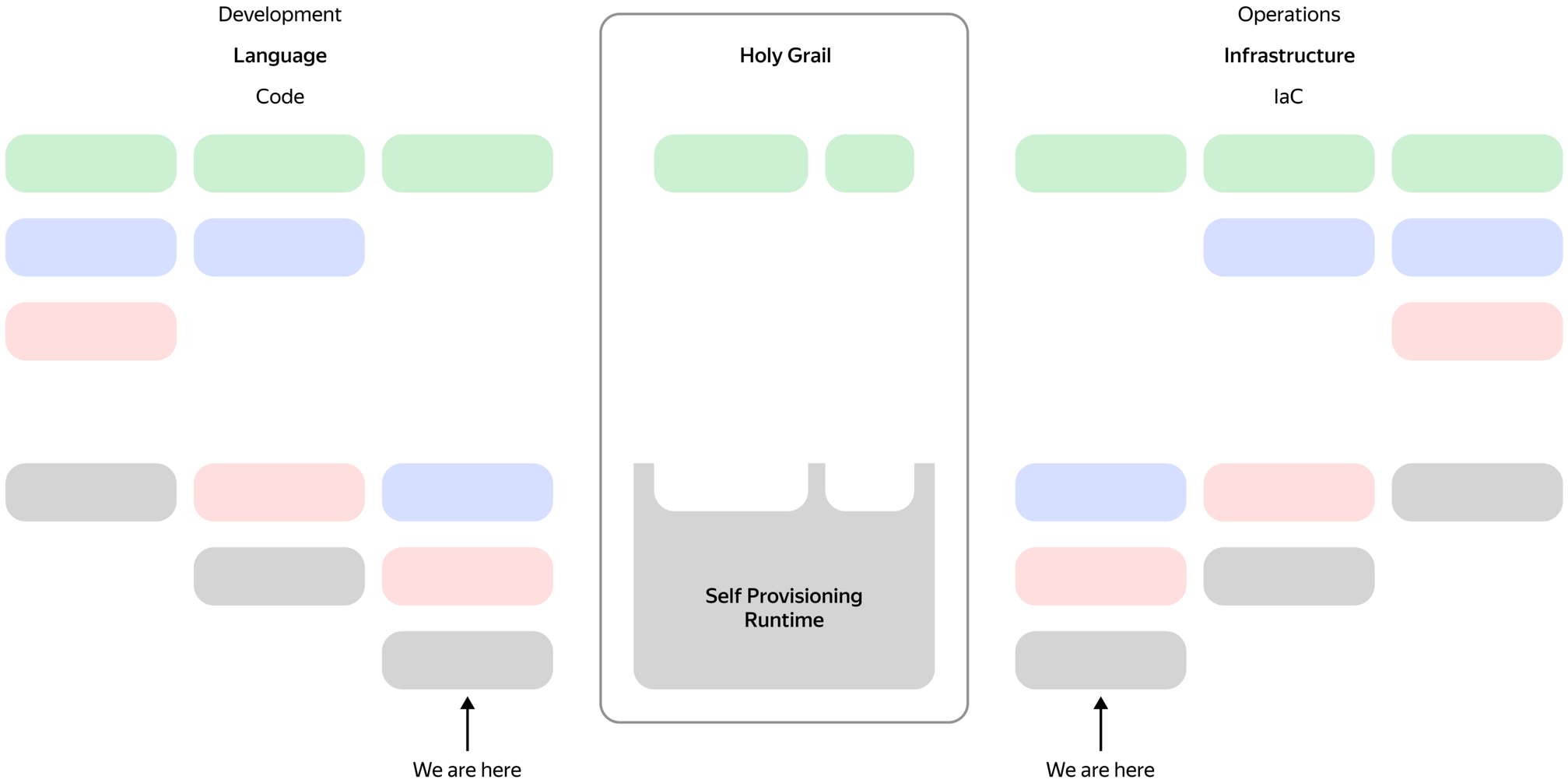
Развитие Ops



Проблема: разрыв между Dev и Ops



Infrastructure from Code?



1. Развитие Dev и Ops
2. Serverless
3. Развитие IaC: прошлое, настоящее и будущее
4. Обзор подходов, технологий и инструментов IfC
5. Демо на примере фреймворка Nitric и Yandex Cloud

Функция как сервис

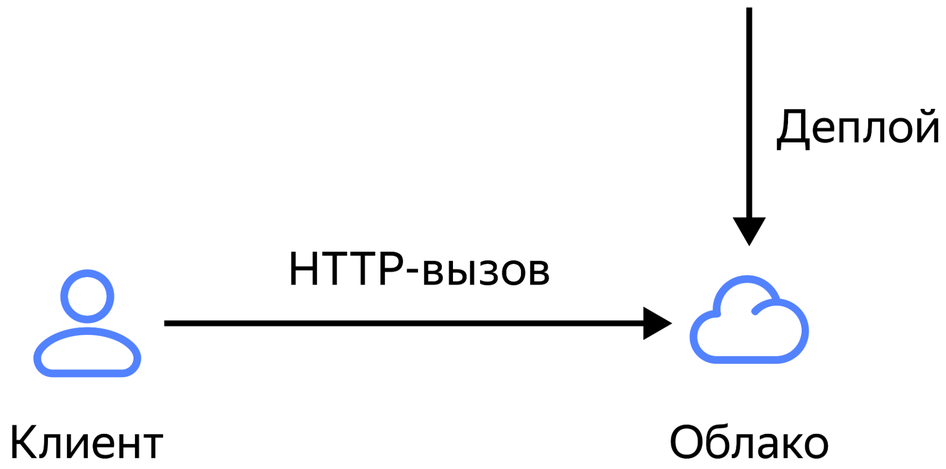
FaaS

- Авто-провижионинг
- Авто-масштабирование
- Авто-обновление
- Авто- …

Функция

index.js

```
1  module.exports.handler = async function (event, context) {
2      return {
3          statusCode: 200,
4          body: 'Hello World!',
5      };
6  };
7
```

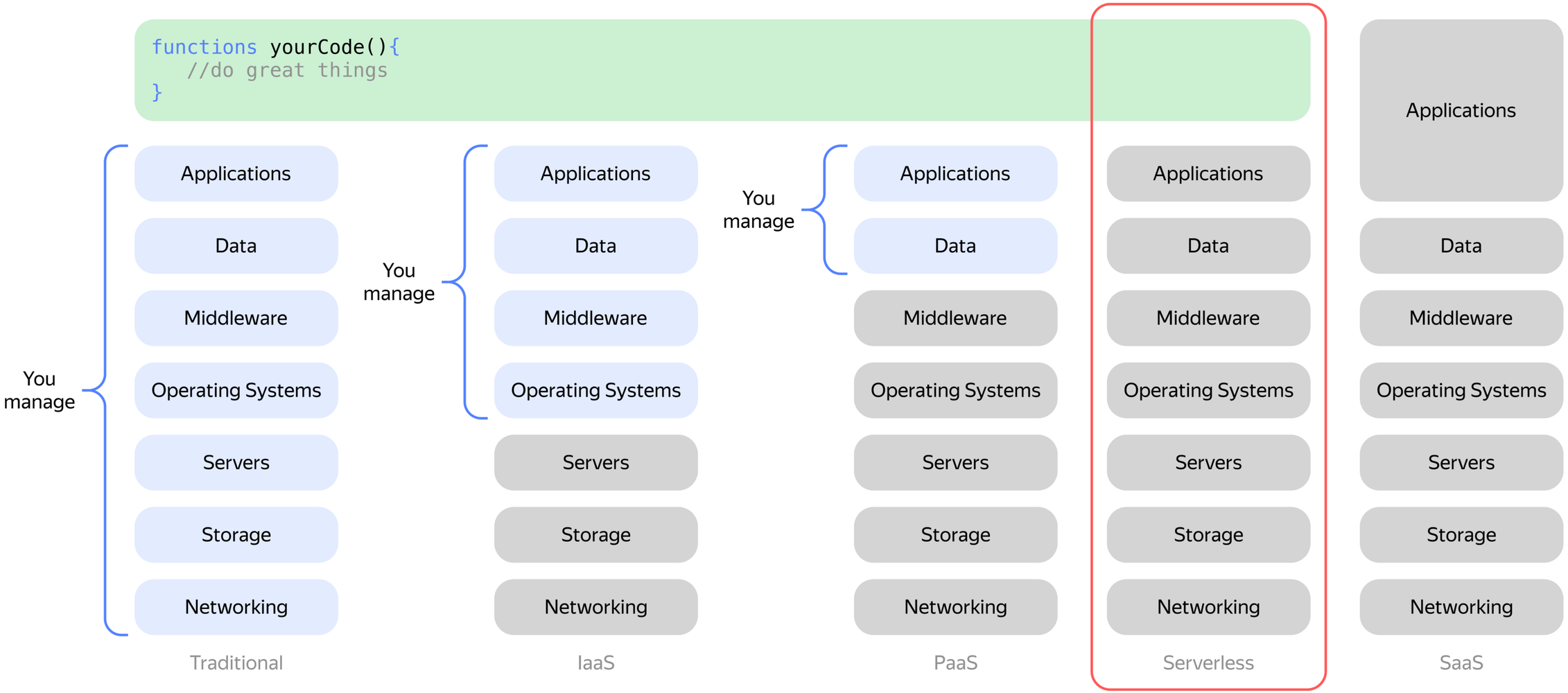


Теперь не только функции …

As a service

- Функция
- API
- Контейнер
- Бакет
- Топик или очередь
- Таблица или коллекция
- Триггер или шина
- Джоба или воркфлоу
- …

Облачные слои

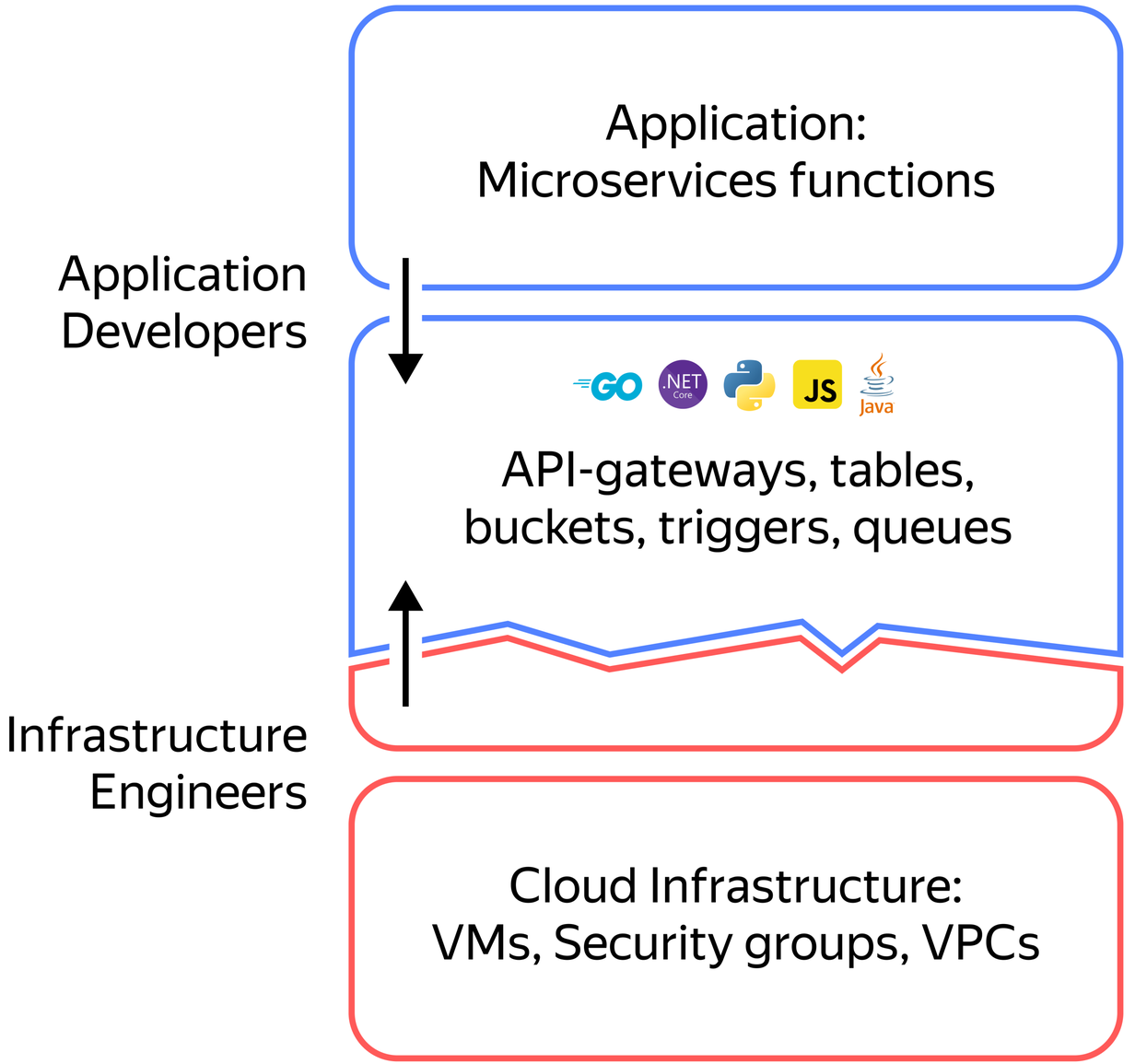


cloud-architecture.io/cloud/

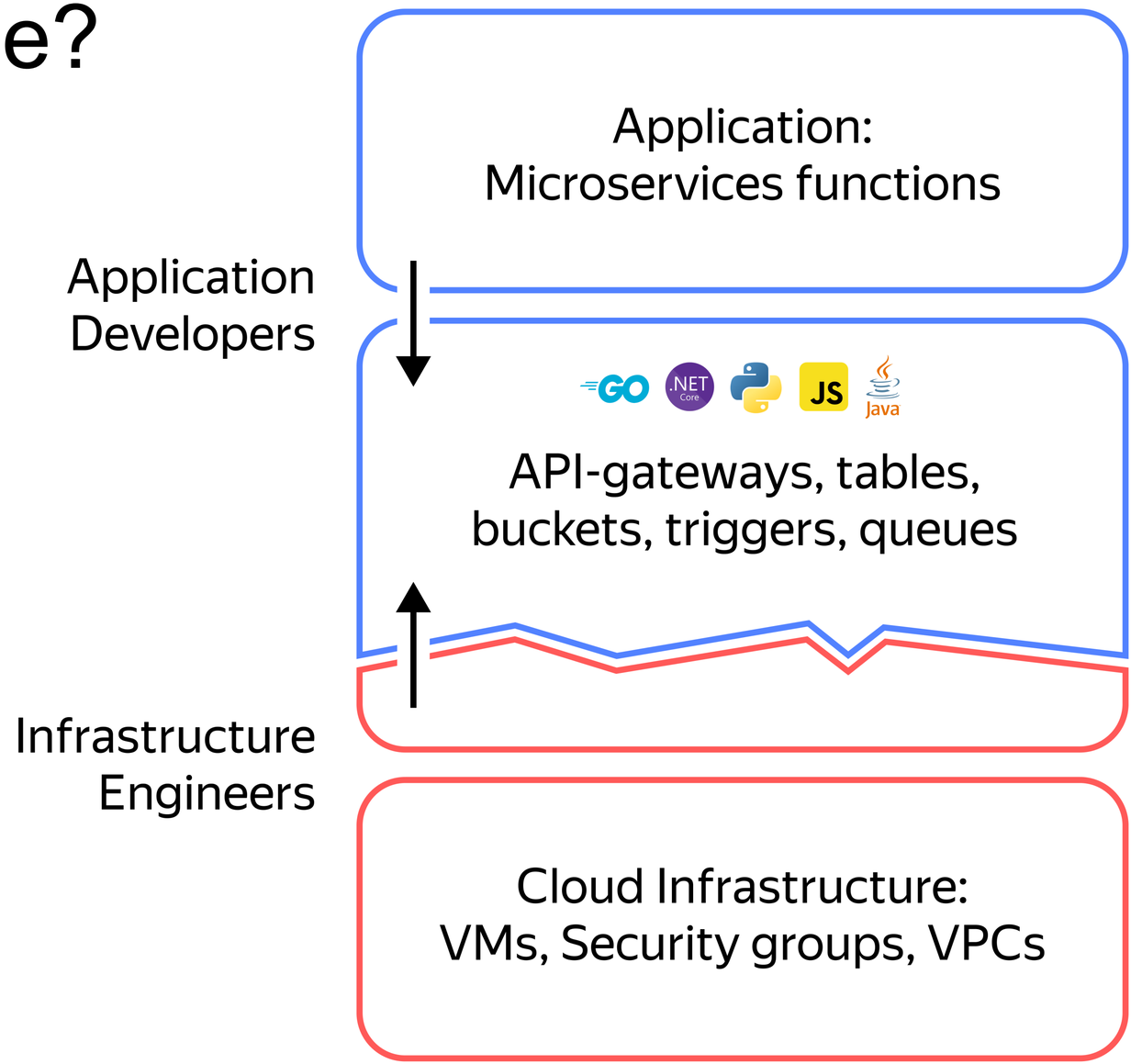
Для сравнения

Слой	IaaS	PaaS	Serverless
API	Network Load Balancer	Application Load Balancer	API Gateway
Business Logic	Virtual Machine (VM) Instance Group	K8S	Function Container
Data	Network Blob Storage (NBS) Network HDD/SSD	Database Network File System (NFS)	Table Bucket
Messaging	Network (VPC)	Message Broker (Kafka)	Queue Topic

Проблема: верхи не хотят — низы не могут!



Infrastructure from Code?



1. Развитие Dev и Ops
2. Serverless
3. Развитие IaC: прошлое, настоящее и будущее
4. Обзор подходов, технологий и инструментов IfC
5. Демо на примере фреймворка Nitric и Yandex Cloud

Императивный DSL

II до н.э.

- Ansible
- Chef

```yaml
# Create Lambda functions
- name: looped creation
  lambda:
    name: '{{ item.name }}'
    state: present
    zip_file: '{{ item.zip_file }}'
    runtime: 'python2.7'
    role: 'arn:aws:iam::987654321012:role/lambda_basic_execution'
    handler: 'hello_python.my_handler'
    vpc_subnet_ids:
    - subnet-123abcde
    - subnet-edcba321
    vpc_security_group_ids:
    - sg-123abcde
    - sg-edcba321
    environment_variables: '{{ item.env_vars }}'
    tags:
      key1: 'value1'

# To remove previously added tags pass an empty dict
- name: remove tags
  lambda:
    name: 'Lambda function'
    state: present
    zip_file: 'code.zip'
    runtime: 'python2.7'
    role: 'arn:aws:iam::987654321012:role/lambda_basic_execution'
    handler: 'hello_python.my_handler'
    tags: {}

# Basic Lambda function deletion
- name: Delete Lambda functions HelloWorld and ByeBye
  lambda:
    name: '{{ item }}'
    state: absent
  loop:
    - HelloWorld
    - ByeBye
```

Декларативный DSL

I до н.э.

- Terraform (HCL)
- AWS Cloud Formation (YAML/JSON)
- Google Cloud Deployment Manager (YAML/Python)
- Azure Resource Manager (JSON)

Проблема: многословно и трудоемко

Код

index.js

```
module.exports.handler = async function (event, context) {
    return {
        statusCode: 200,
        body: 'Hello World!',
    };
};
```

Инфра

Infrastructure from Code?

Код

index.js

```
1  module.exports.handler = async function (event, context) {
2      return {
3          statusCode: 200,
4          body: 'Hello World!',
5      };
6  };
7
```

Автоматическая генерация

Инфра

Serverless DSL

Н. э.

- Serverless Framework
- AWS Serverless Application Model

  SAM

```yaml
service: apigw-lambda-sls
frameworkVersion: '^3' # require serverless v3 or later

plugins:
  - serverless-plugin-typescript # enable TypeScript support

provider:
  name: aws

  # common configuration for all Lambda functions in this st
  runtime: nodejs20.x
  architecture: arm64 # use Graviton for running all Lambda

  # override the default stage (dev)
  stage: prod

  # optional, Lambda function's memory size in MB, default i
  memorySize: 256

  apiGateway:
    description: API Gateway for this stack

# Lambda function triggered with events from the default Eve
functions:
  logEvent:
    handler: src/handler.echoEvent
    events:
      # use shortcut path definition
      - http: GET /{proxy+}
      - http: GET /
```

Cloud Development Kit

Н. э.

- AWS CDK
- Terraform CDK
- Pulumi

```
import * as aws from '@pulumi/aws'
import * as apigateway from '@pulumi/aws-apigateway'

// Create a Lambda Function
const helloHandler = new aws.lambda.CallbackFunction('hello-handler', {
  callback: async (ev, ctx) => {
    return {
      statusCode: 200,
      body: 'Hello, API Gateway!',
    }
  },
})

// Define an endpoint that invokes a lambda to handle requests
const api = new apigateway.RestAPI('api', {
  routes: [
    {
      path: '/',
      method: 'GET',
      eventHandler: helloHandler,
    },
  ],
})

export const url = api.url
```

Serverless CDK

Н. э.

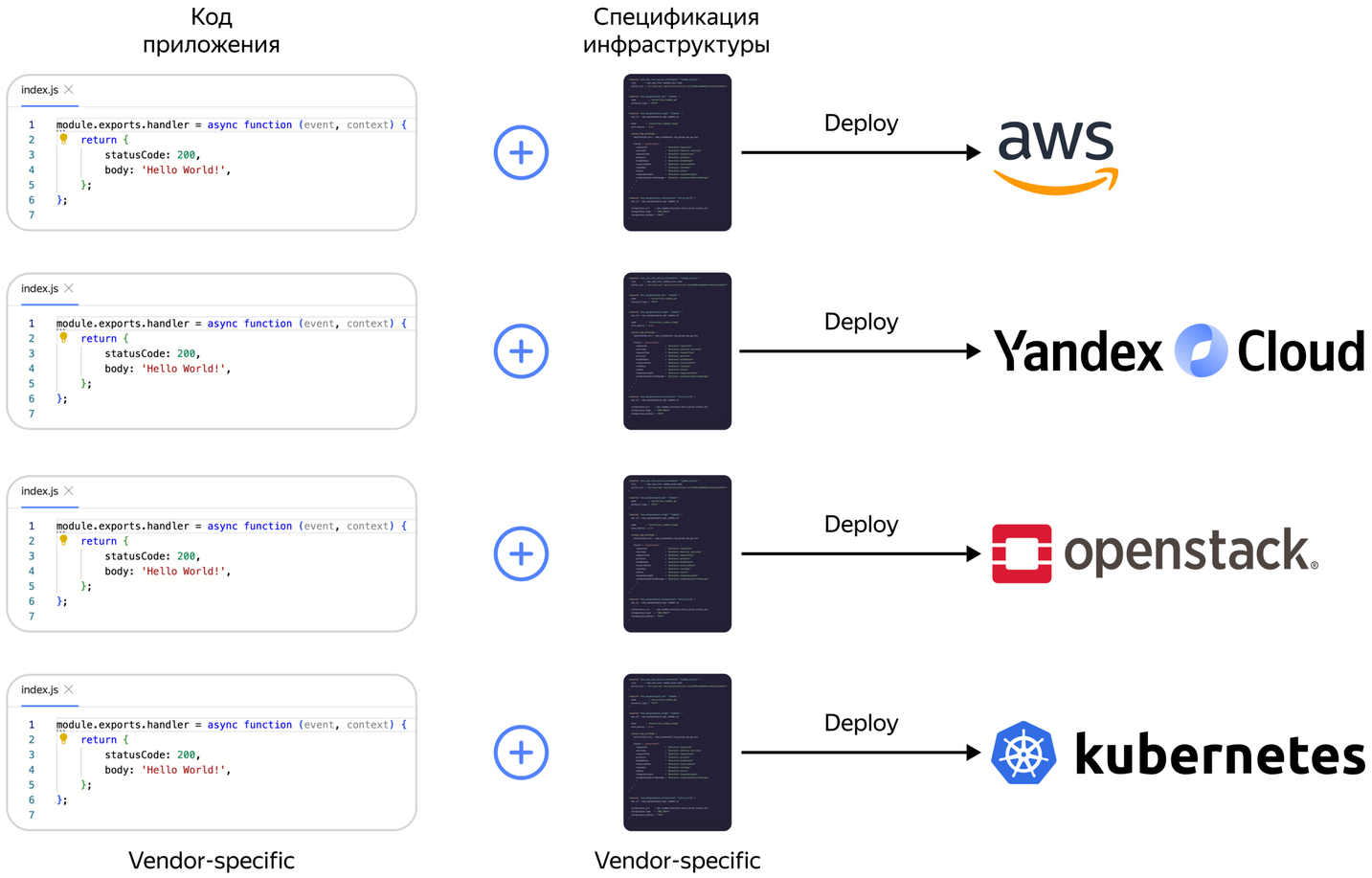
- AWS Amplify
- Serverless Stack

  SST

sst.config.ts

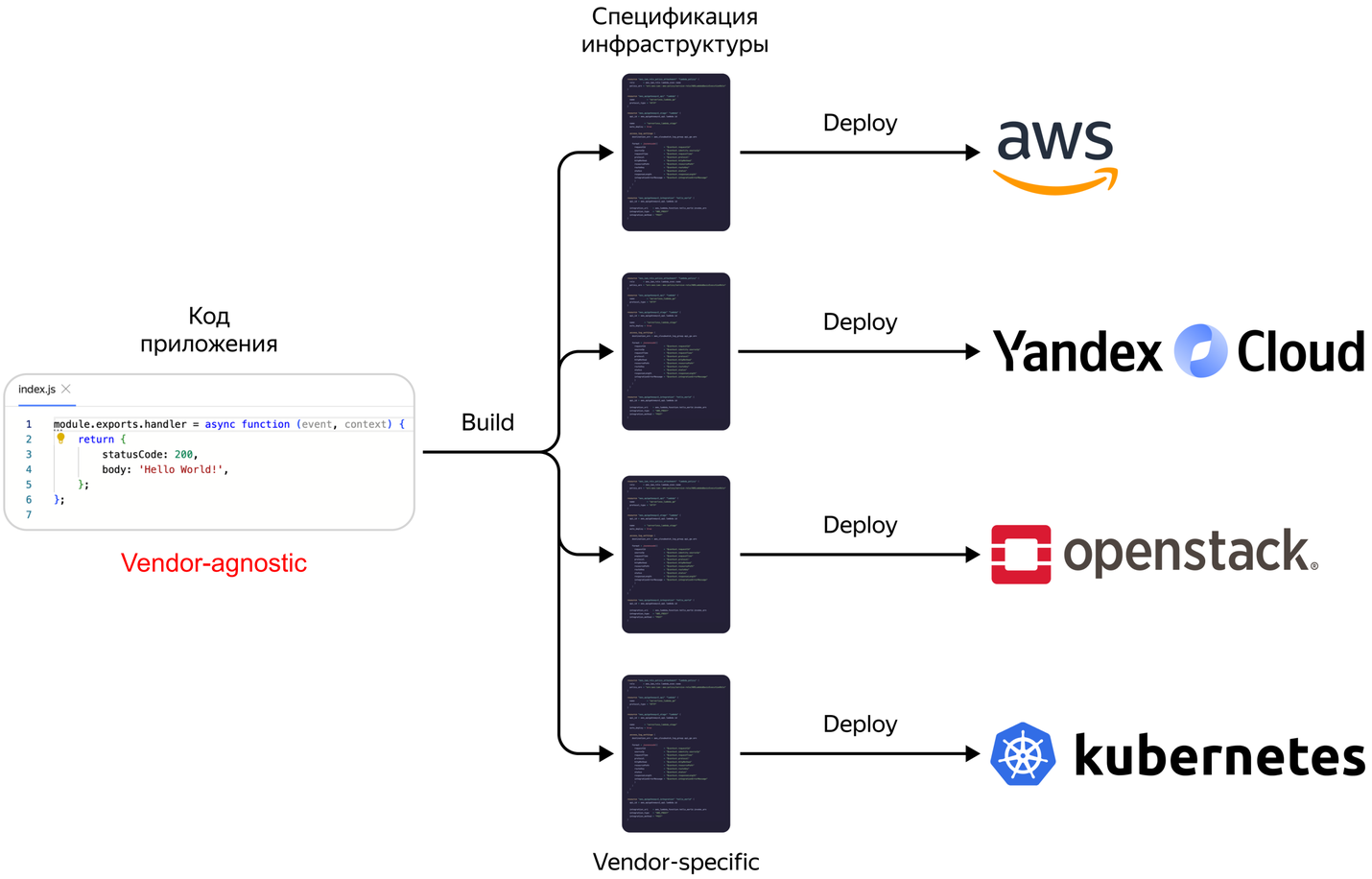
```ts
const db = new planetscale.Database("Db")

const email = new sst.aws.Email("Email", {
  sender: "mail@example.com"
})

const api = new sst.aws.Service("Api", {
  memory: "4 GB",
  image: "./rails",
  link: [db, email]
})

const web = new sst.aws.Nextjs("Web", {
  link: [api],
  path: "./nextjs",
  domain: "example.com",
  dns: sst.cloudflare.dns()
})
```

Проблема: vendor- или platform-lock



Infrastructure from Code?

Write Once Deploy Anywhere



Принципы IfC

Low-Ops

Автоматизировать создание инфраструктуры, опираясь только на информацию из кода приложения

Developer-friendly

Предоставить возможность описывать высокоуровневые и относительно гранулярные абстракции без boilerplate, понятным и привычным для программиста способом на ЯП

Vendor-free

предоставить возможность описывать инфраструктуру в независящем от платформы (облака, вендора) виде

1. Развитие Dev и Ops
2. Serverless
3. Развитие IaC: прошлое, настоящее и будущее
4. Обзор подходов, технологий и инструментов IfC
5. Демо на примере фреймворка Nitric и Yandex Cloud

Обзор: developer experience

1. Аннотирование кода
2. Software Development Kit

   SDK
3. Язык программирования

   ЯП

Удобство

Простота

Гибкость

Экосистема

Компетенции

1. Аннотирование кода

Ключевое

Код размечается аннотациями

Или комментариями в специальном формате

Анализатор автоматически:

- Находит аннотации
- По ним определяет тип ресурса
- Создает ресурс или генерирует его описание на языке спецификации IaC

```
/* @klotho::expose {
 *   target = "public"
 *   id = "app"
 * }
 */
http.ListenAndServe(":3000", r)
```

1. Аннотирование кода

Особенности

Ориентирован на конкретный ЯП — у каждого ЯП будет свой способ аннотирования кода и своя реализация

Ограниченная поддержка в IDE — использование нестандартных аннотаций и комментариев

Зависимость от роадмапа языка и инструментария для анализа кода в этом языке

1. Аннотирование кода

Пример

- Klo.dev
- Open Source 1100 ⭐
- Языки: C#, Go, JS, Python
- Платформы: AWS

Klotho **klotho**

```go
func main() {
    r := chi.NewRouter()
    r.Use(middleware.Logger)

    r.Get("/hello", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte("Hello from Klotho!"))
    })

    r.Get("/hello/{name}", func(w http.ResponseWriter, r *http.Request) {
        w.Write([]byte(fmt.Sprintf("Hello %s!", chi.URLParam(r, "name"))))
    })

    fmt.Println("Listening on :3000")

    /* @klotho::expose {
     *   target = "public"
     *   id = "app"
     * }
     */
    http.ListenAndServe(":3000", r)
}
```

2. SDK

Ключевое

- **Простая интеграция с традиционным IaC**

  Проект кода приложения (Dev) и проект спецификации инфраструктуры (Ops) могут существовать независимо и дополнять друг друга с помощью определенного заранее протокола

- **Простая кастомизация и расширение**

  В случае специфичных для конкретного проекта или вендора / платформы / провайдера нужд

- **Бесплатная поддержка в IDE и инструментах разработки**

  Автодополнение, безопасность типов, подсветка ошибок и т. д.

hello.ts

```ts
import { api } from '@nitric/sdk'

const helloApi = api('main')

helloApi.get('/hello/:name', async (ctx) => {
  const { name } = ctx.req.params

  ctx.res.body = `Hello ${name}`
})
```

2. SDK

Особенности

Ориентирован
на конкретный ЯП:
для каждого ЯП
требуется своя
реализация SDK

Могут быть ограничения:
на структуру проекта,
зависимости, процесс сборки

2. SDK

Пример

- Nitric.io
- Open Source 1100 ⭐
- Языки: JS/TS, Python, Go, C#, Java, Dart
- Платформы: AWS, Azure, GCP

nitric

hello.ts

```ts
import { api } from '@nitric/sdk'

const helloApi = api('main')

helloApi.get('/hello/:name', async (ctx) => {
  const { name } = ctx.req.params

  ctx.res.body = `Hello ${name}`
})
```

3. ЯП

Ключевое

- **Единый и унифицированный подход**

  Объединяет IfC и IaC, предоставляя комплексное решение для управления как прикладным, так и инфраструктурным кодом

- **Повышенная производительность**

  Позволяет командам разработки работать с абстрактными и детализированными облачными API-интерфейсами, используя лучшие встроенные практики

- **Упрощенная структура и удобство**

  Языковые возможности напрямую позволяют управлять облачной инфраструктурой, устраняя необходимость в отдельных структурах управления в существующих языках, которые могут добавлять бойлерплейт

```
bring cloud;

// define an API
let api = new cloud.Api();
// define storage
let store = new cloud.Bucket();

// create routes for the API
api.get("/employees", inflight (req) => {
  let result = MutJson [];
  let var i = 0;
  for employee in store.list() {
    result.setAt(i, employee);
    i = i + 1;
  }
  return cloud.ApiResponse {
    status: 200,
    body: Json.stringify(result)
  };
});
```

3. ЯП

Особенности

Потеря скиллов:

Существующие знания и опыт не могут быть перенесены, от команды требуется изучение нового языка и методологии

Высокая стоимость перехода:

Командам может потребоваться переписать приложения и они не смогут использовать существующие библиотеки

Ограниченная экосистема:

Поскольку язык является новым, на начальном этапе будут отсутствовать инструменты, библиотеки, расширения и поддержка сообщества, что усложнит поиск и устранение неисправностей

3. ЯП

Пример

- Winglang.io
- Open Source 5000 ⭐
- Языки: Wing
- Платформы: AWS, Azure, GCP

winglang

```
bring cloud;

// define an API
let api = new cloud.Api();
// define storage
let store = new cloud.Bucket();

// create routes for the API
api.get("/employees", inflight (req) => {
  let result = MutJson [];
  let var i = 0;
  for employee in store.list() {
    result.setAt(i, employee);
    i = i + 1;
  }
  return cloud.ApiResponse {
    status: 200,
    body: Json.stringify(result)
  };
});
```

Обзор: deployment experience

1. Специализированные облака
2. Платформы, интегрированные с гиперскейлерами
3. Независимые фреймворки

Удобство

Простота

Гибкость

Независимость

1. Специализированное облако

Ключевое

- **Решение «под ключ»**

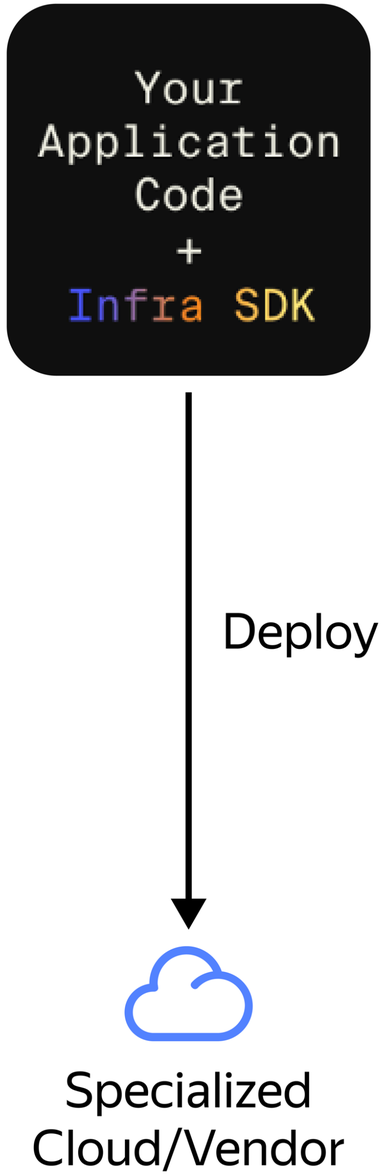
  Развертывание упрощается до нескольких шагов или команд, при этом большая часть инфраструктуры управляется платформой автоматически

- **Полная абстракция**

  Платформа управляет инфраструктурой как частью жизненного цикла приложения без вмешательства разработчиков

- **Отсутствие взаимодействия с IaC**

  Разработчикам не приходится напрямую взаимодействовать с инфраструктурой, поскольку платформа абстрагирует инфраструктуру как часть конвейера развертывания



1. Специализированное облако

Особенности

Отсутствие контроля

Разработчики имеют минимальную возможность кастомизировать или модифицировать настройки инфраструктуры

Vendor-lock

Решение становится зависимым от конкретной платформы, что создает проблемы при необходимости миграции и ограничивает возможности ценообразования

Ограниченная гибкость и расширяемость

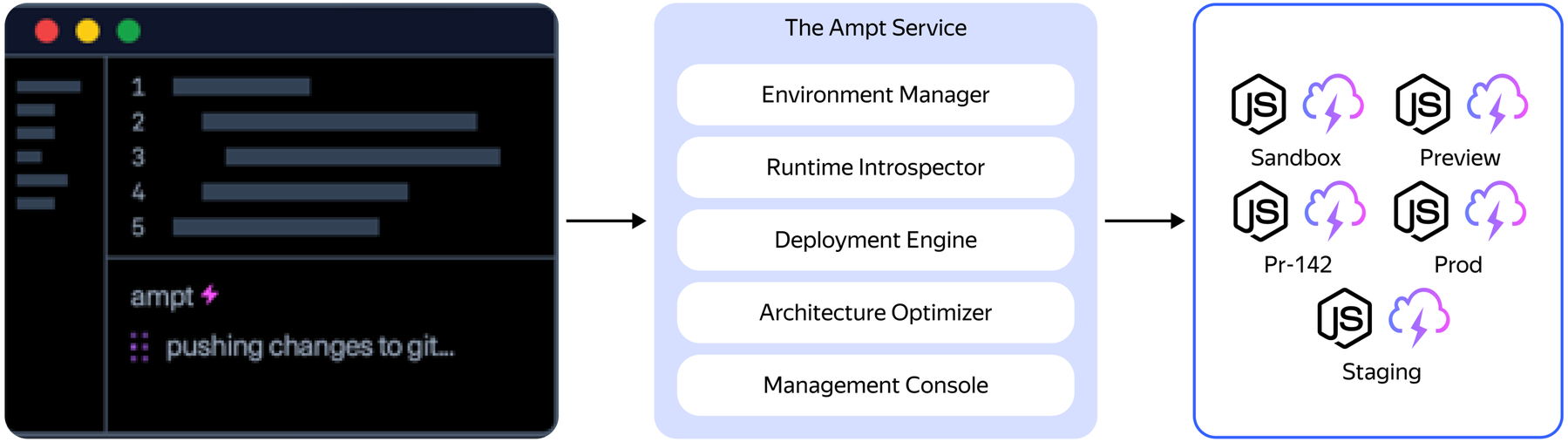
Поскольку платформа контролирует большую часть инфраструктуры, команды Dev и Ops ограничены функциями и конфигурациями, поддерживаемыми платформой, которые могут не покрывать все необходимые сценарии

1. Специализированное облако

ampt

Пример

- Getampt.com
- Proprietary
- SDK
- Языки: JavaScript
- Платформы: AWS



Self-Deploying

Ampt connects to your source control and automatically builds and deploys code changes

Self-Provisioning

Ampt uses runtime introspection to analyze apps, then provisions and configures cloud resources

Self-Optimizing

Ampt monitors your apps and automatically optimizes memory, resources, and compute

2. Интегрированная платформа

Ключевое

- ## Автоматизация развертывания

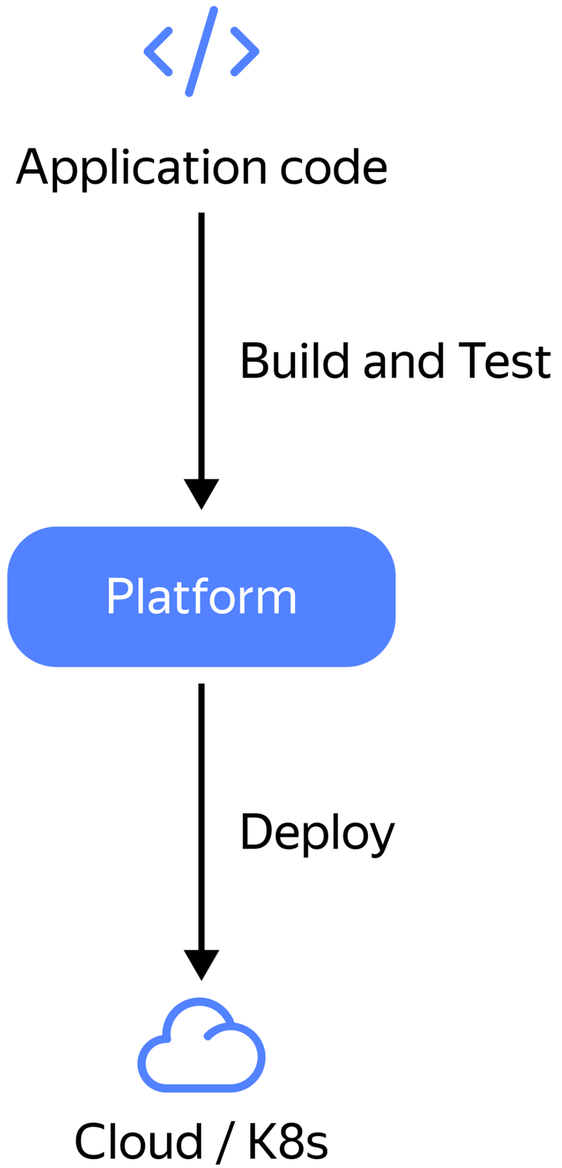
  Развертывание происходит автоматически,
  при этом некоторые части обрабатываются платформой,
  а другие настраиваются пользователем

- ## Настраиваемый IaC

  Разработчики, как правило, по-прежнему контролируют
  подготовку инфраструктуры и могут внедрять
  настраиваемый IaC в процесс развертывания

- ## Относительная гибкость

  Предоставляет баланс между контролем и удобством,
  обеспечивая большую гибкость по сравнению
  с полностью управляемыми платформами



2. Интегрированная платформа

Особенности

Сложность

Инструменты, поддерживаемые платформой, могут усложнить работу, поскольку пользователи должны знать, когда следует полагаться на автоматизацию, а когда вручную настраивать определенные части развертывания

Platform-lock

Хотя приложения могут разворачиваться на различные окружения и облака, платформа, выполняющая автоматизацию, будет фиксированной. Часто для использования этих платформ требуется подписка.

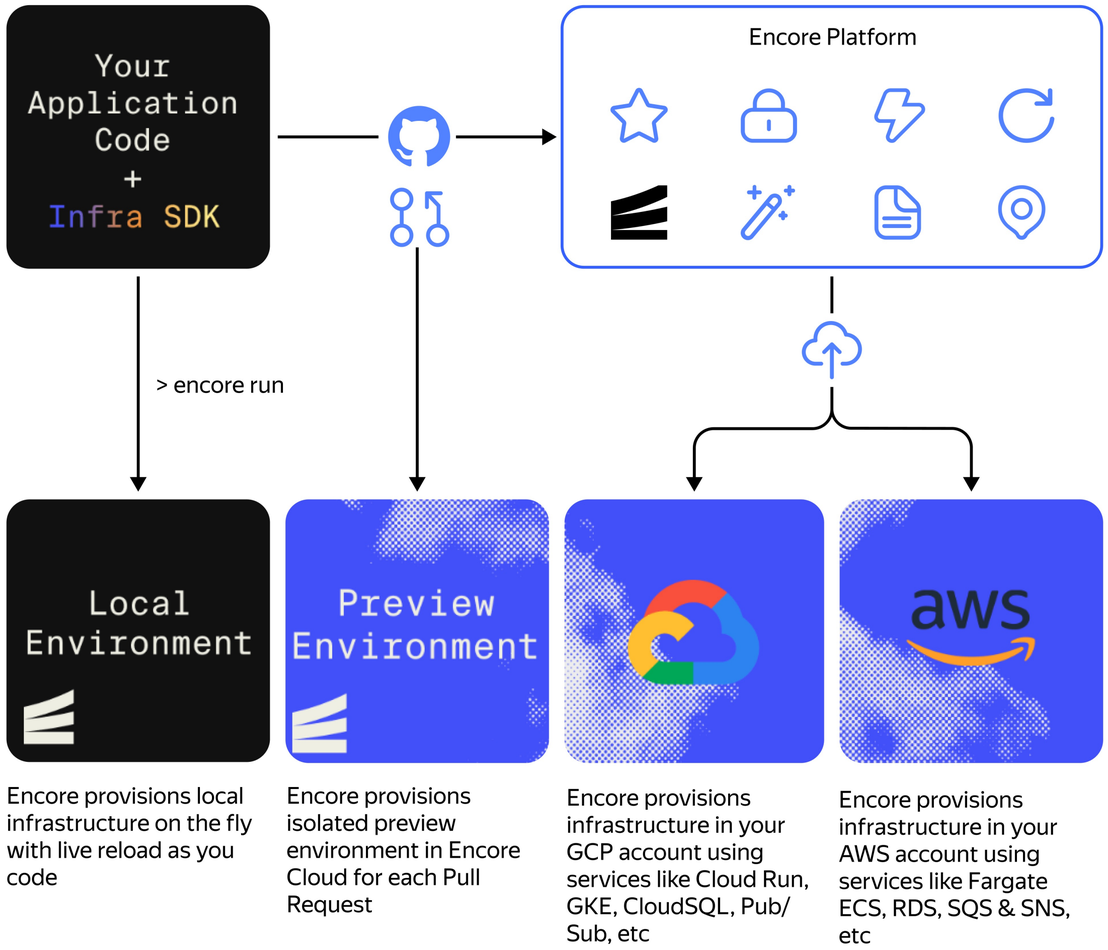
Накладные расходы на техническое обслуживание

Ответственность за определенные компоненты инфраструктуры по-прежнему лежит на командах разработки или эксплуатации, что требует дополнительных усилий по мониторингу и управлению

2. Интегрированная платформа

Пример

- Encore.dev
- Open Source 7000 ⭐
- SDK
- Языки: TypeScript, Go
- Платформы: AWS, GCP



encore

3. Независимый фреймворк

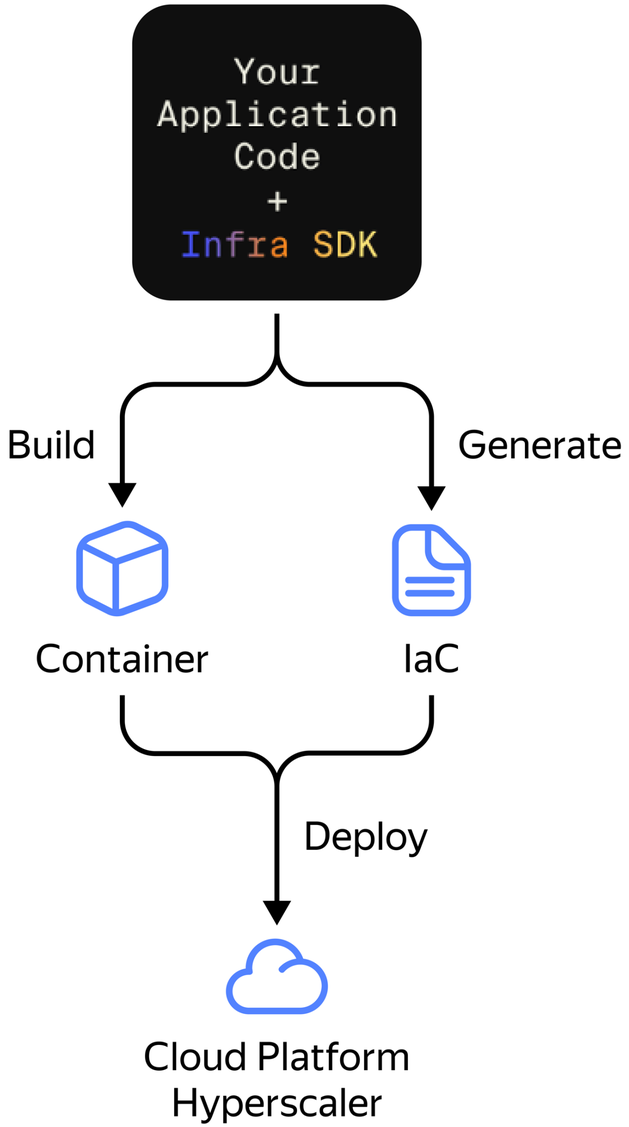
Ключевое

- **Расширяемость и гибкость**

  Легко расширяются с помощью IaC для удовлетворения уникальных потребностей конкретной инфраструктуры

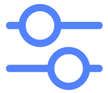
- **Интегрируемость**

  Могут легко интегрироваться непосредственно в существующие CI/CD-сценарии, что делает их отличным выбором для команд с налаженными рабочими процессами автоматизации



3. Независимый фреймворк

Особенности



Дополнительные усилия по настройке

Могут потребовать дополнительной настройки и конфигурации, что может занять некоторое время, особенно для команд, впервые работающих с IaC

Управление во время выполнения

Ускоряют разработку и развертывание, но могут мало помочь в управлении облачными ресурсами после их развертывания

Менее удобный и понятный

Возможность настройки каждого аспекта инфраструктуры является мощной и гибкой, но для эффективного управления может потребоваться опыт работы с IaC и облачными платформами

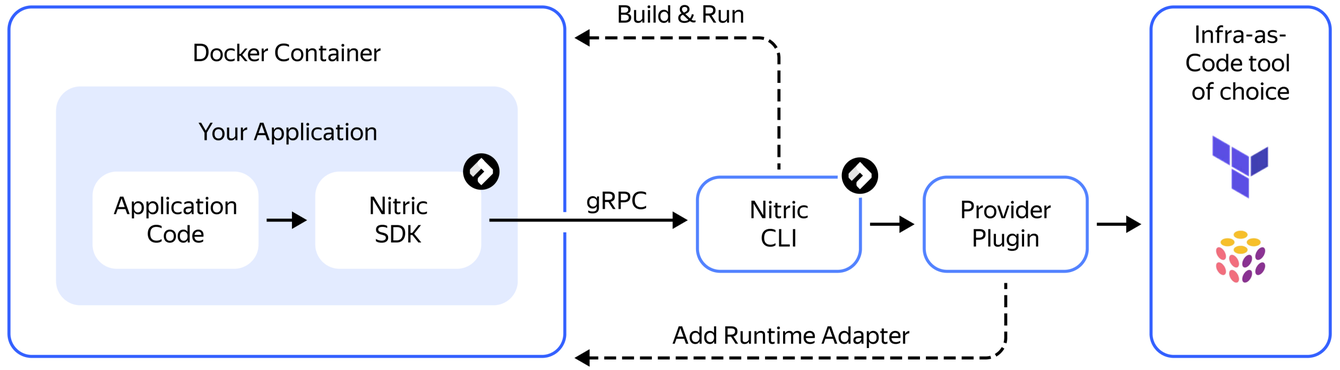
3. Независимый фреймворк

nitric

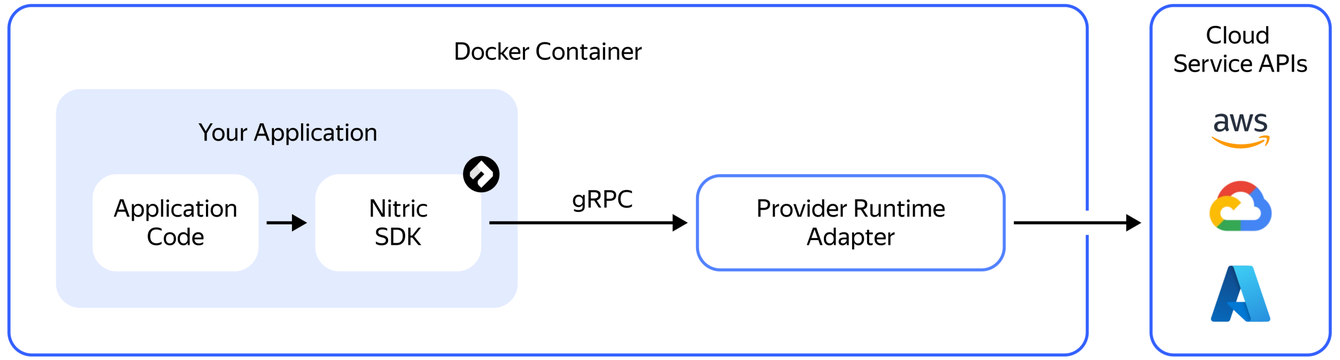
Пример

- Nitric.io
- Open Source 1100 ⭐
- Языки: JS/TS, Python, Go, C#, Java, Dart
- Платформы: AWS, Azure, GCP

Nitric Deployment Architecture



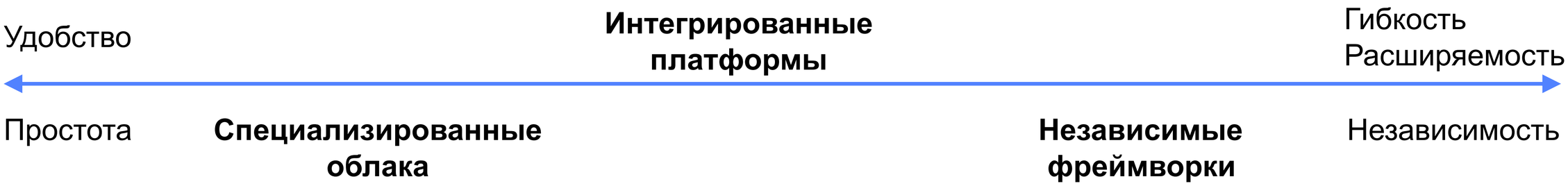
Nitric Runtime Architecture



1. Развитие Dev и Ops
2. Serverless
3. Развитие IaC: прошлое, настоящее и будущее
4. Обзор подходов, технологий и инструментов IfC
5. Демо на примере фреймворка Nitric и Yandex Cloud

Демо

Выводы



Deployment\Development	Анализ кода	SDK	ЯП
Специализированные облака	—	Ampt, Modal Labs, Shuttle	—
Интегрированные платформы	Klotho	Encore	—
Независимые фреймворки	—	Nitric	Wing

Выводы

Удобство **ЯП** **Аннотации** **SDK** Гибкость

Простота

Экосистема
Компетенции
Комьюнити

Deployment\Development	Анализ кода	SDK	ЯП
Специализированные облака	—	Ampt, Modal Labs, Shuttle	—
Интегрированные платформы	Klotho	Encore	—
Независимые фреймворки	—	Nitric	Wing

Спасибо
за внимание!

Виктор Кузенный
Руководитель разработки
Serverless-сервисов,
Yandex Cloud

yandex.cloud/ru/
services#serverless