GODEL TECHNOLOGIES

Java microservices: from Netflix OSS to Kubernetes.

Обо мне

Александр Ноздрин-Плотницкий

Ведущий разработчик Godel Technologies

a.nozdryn-platnitski@godeltech.com

О чем поговорим

- Система V1
- Проблемы V1 решения
- Почему Kubernetes
- Миграция на V2 и грабли
- Выводы

Godel Technologies

Предоставляем сервисы по разработке программного обеспечения для среднего и крупного бизнеса в Великобритании.

Система V1

Базовые требования

Java 8 (август 2016)

Микросервисы

Amazon Web Services (AWS)

Spring Framework

Нефункциональные требования

Развертывание без простоя (Zero Downtime deployment)

Безопасность

Масштабируемость

Доступность API 99,99%

Восстановление после ошибок

Нефункциональные требования

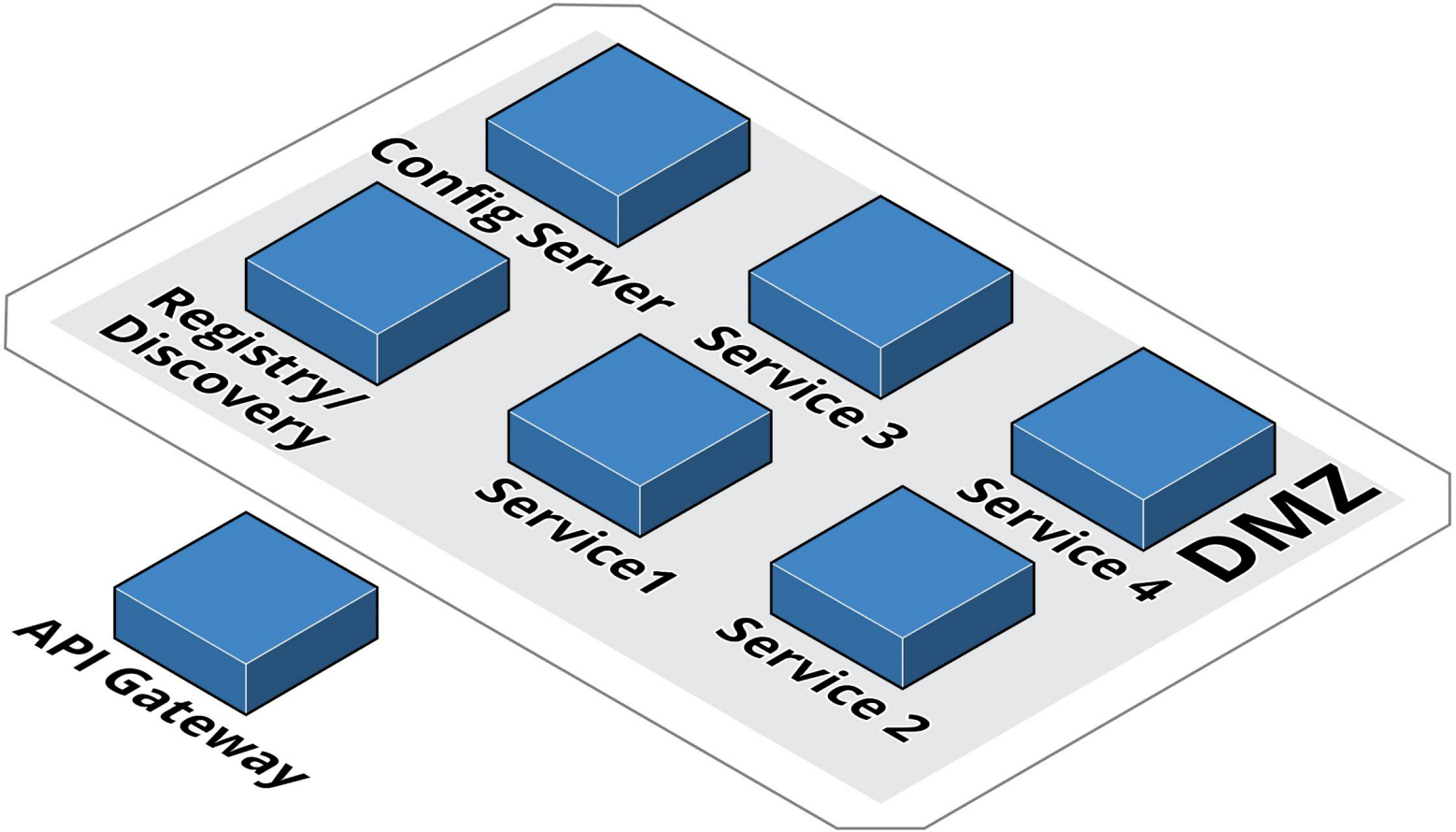
Централизованное логирование
Централизованный мониторинг
Оповещения о сбоях в системе
Система непрерывной интеграции и развертывания (CI/CD)
Автоматизация инфраструктуры

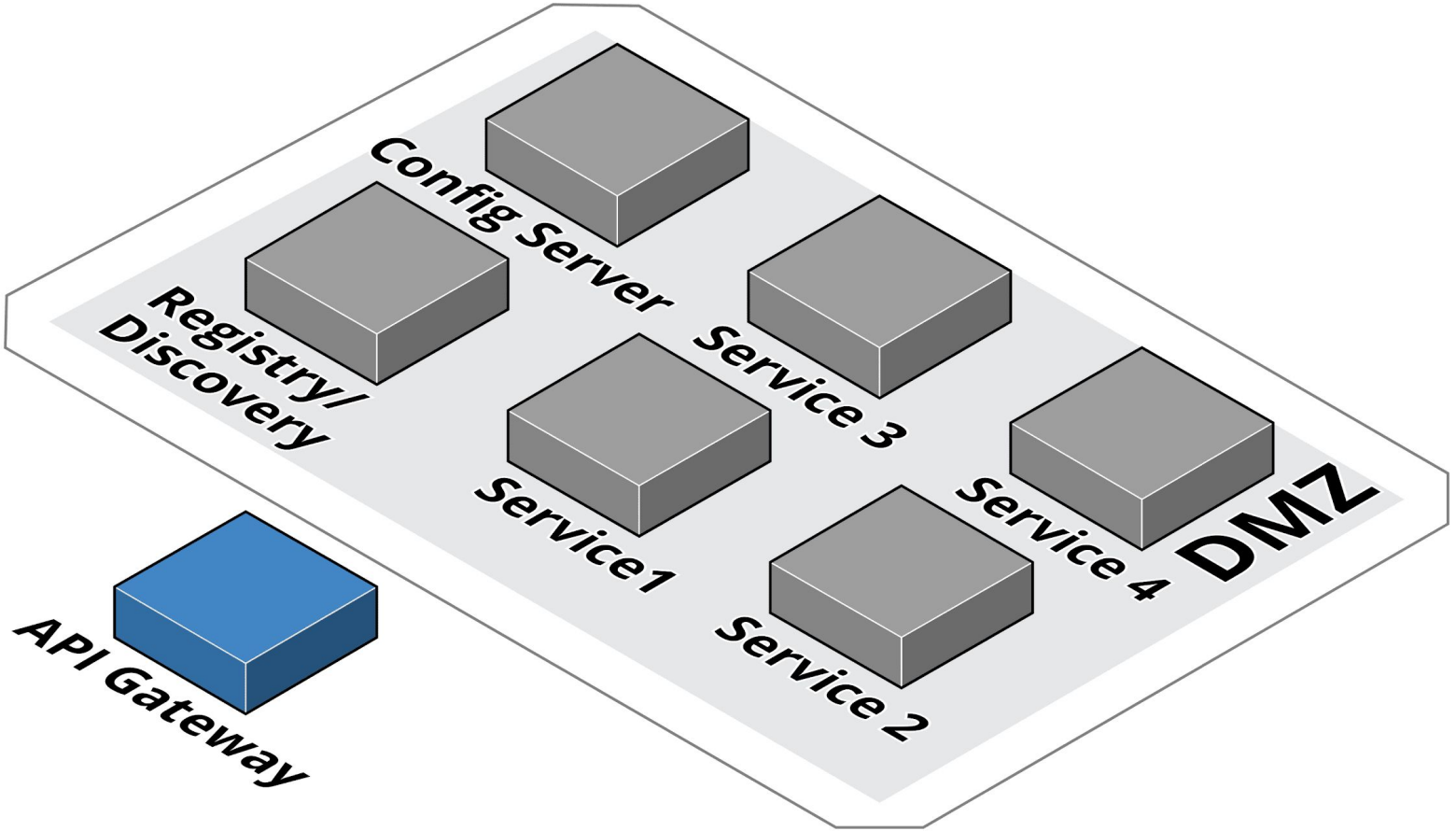
Команда

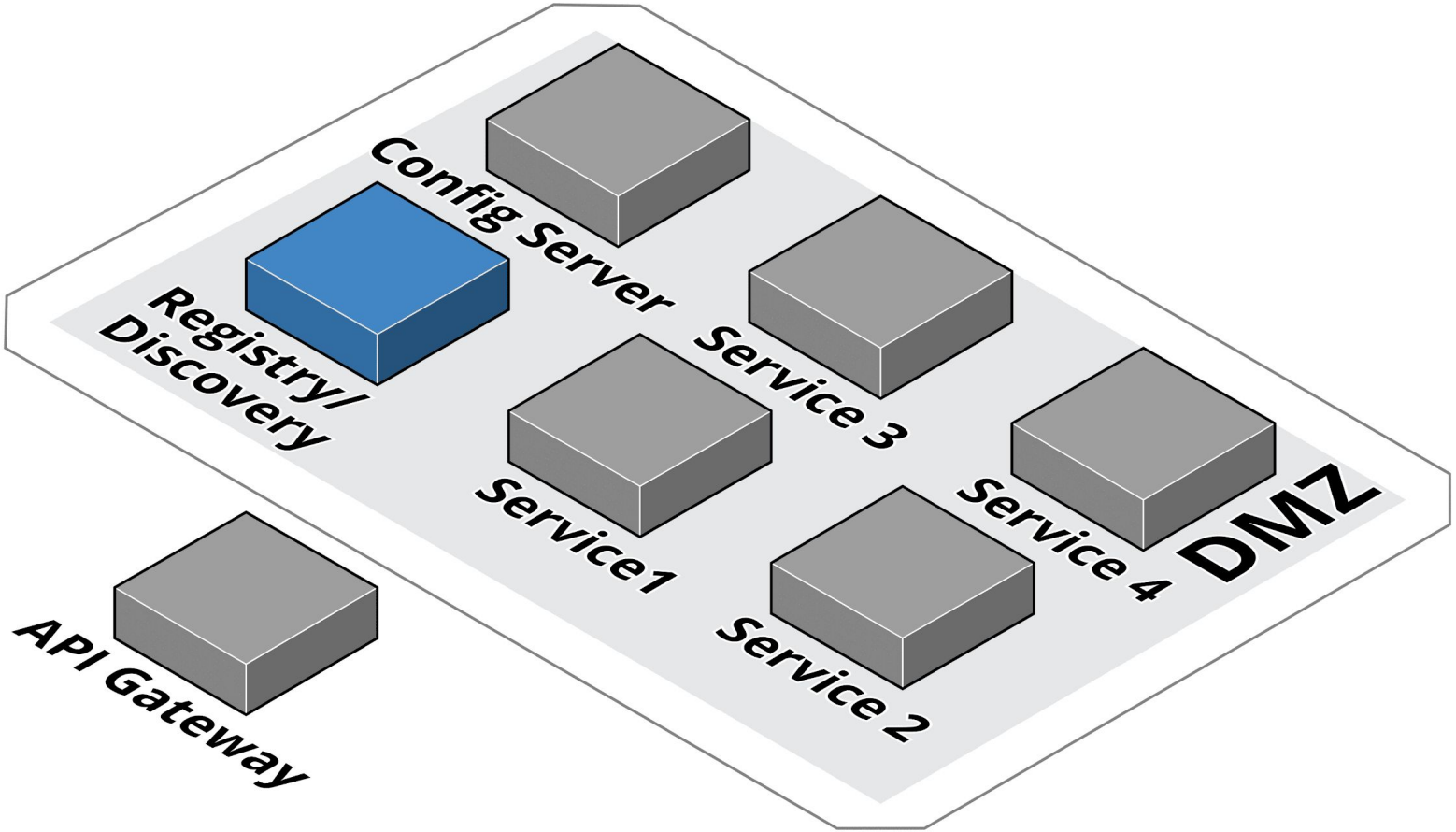
Начало: 8 Инженеров

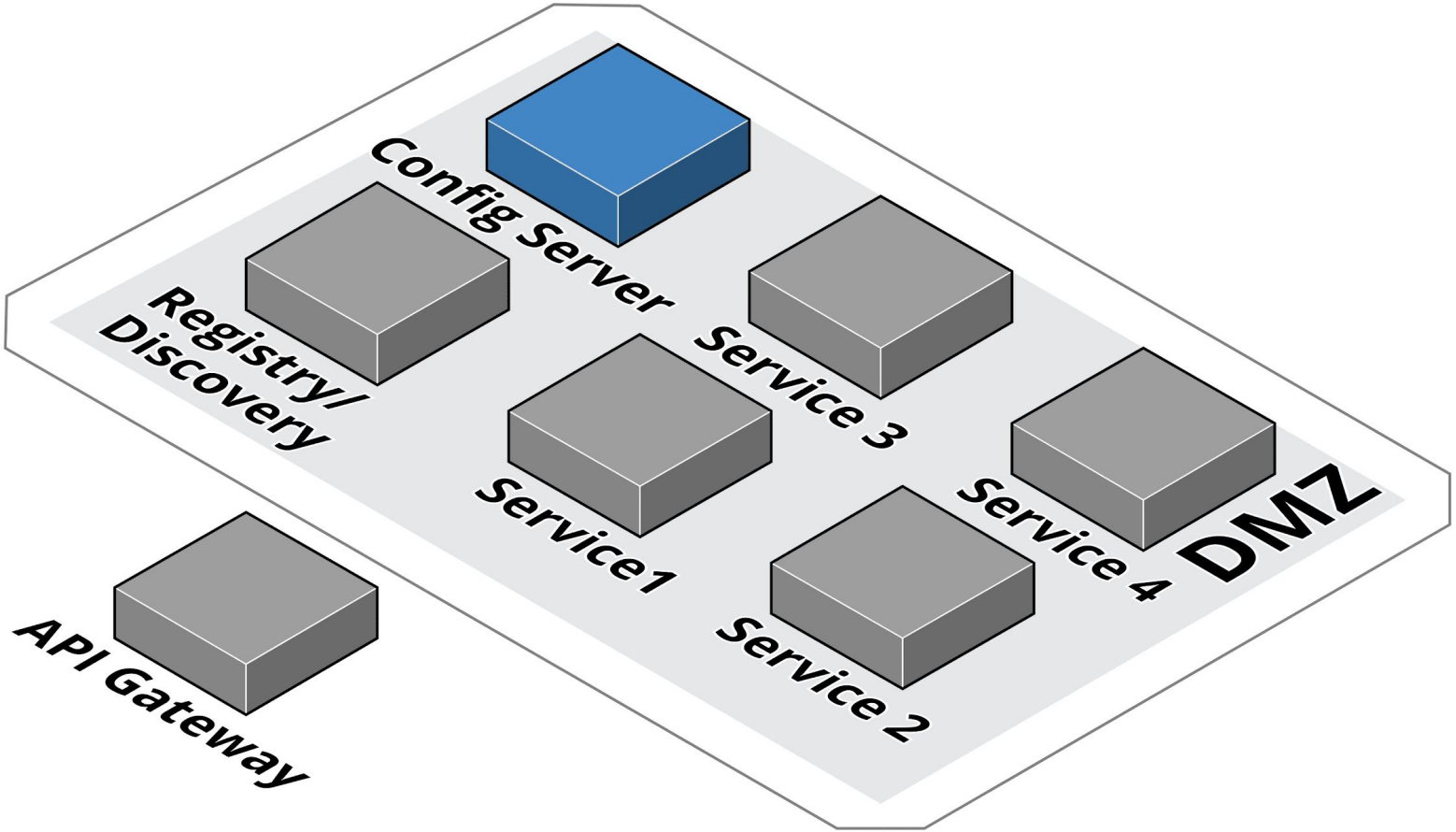
Пиковое: 20 Инженеров - 2 потока разработки

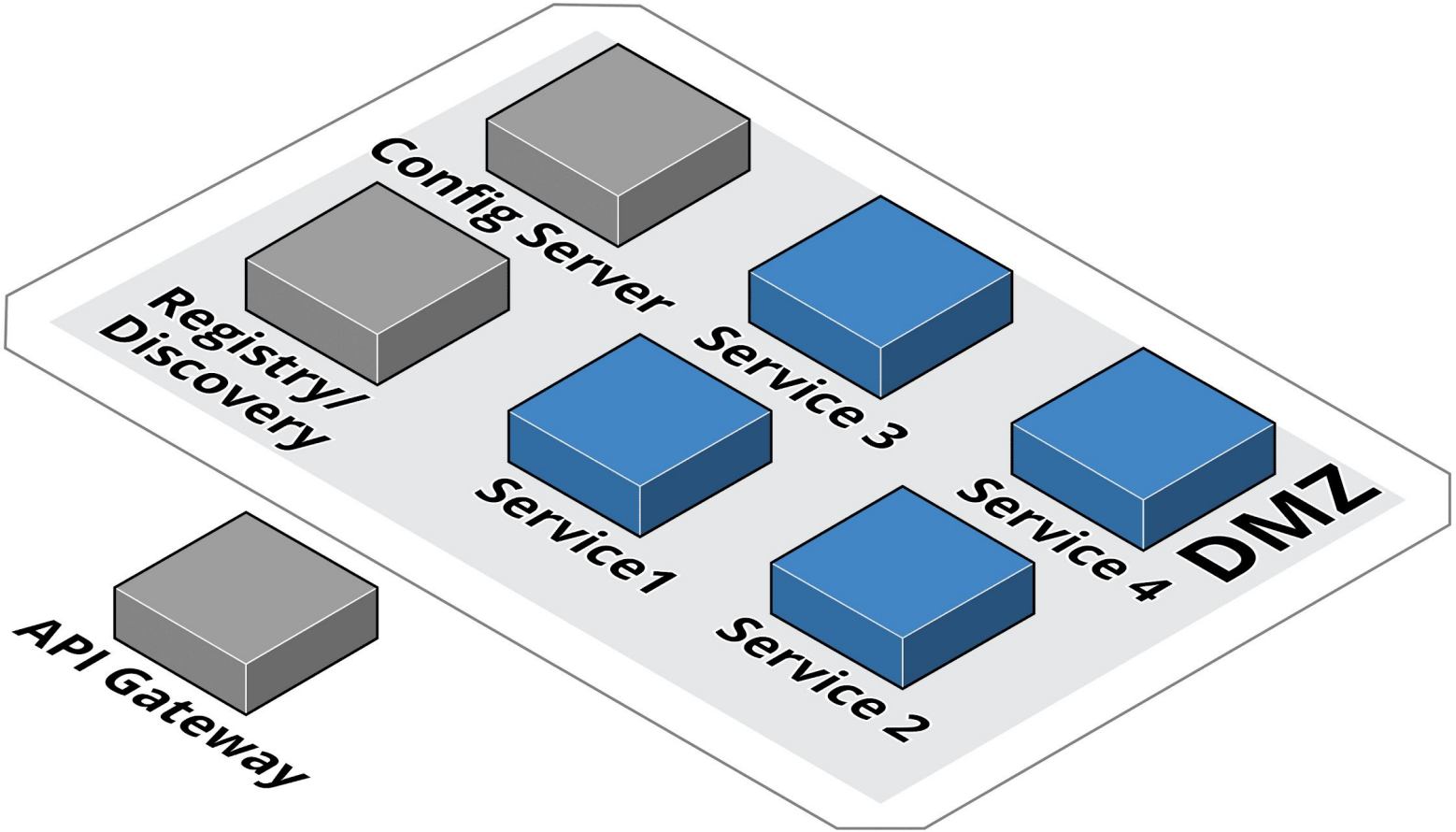
Архитектурное решение


Config Server
Registry/
Discovery
Service 3
Service1
Service 4
DMZ
API Gateway
Service 2


Config Server
Registry/
Discovery
Service 3
Service1
Service 4
DMZ
Service 2
API Gateway


Config Server
Registry/
Discovery
Service 3
Service1
Service 4
DMZ
Service 2
API Gateway


Config Server
Registry/
Discovery
Service 3
Service1
Service 4
DMZ
API Gateway
Service 2


Config Server
Registry/
Discovery
Service 3
Service1
Service 4
DMZ
Service 2
API Gateway

Spring Boot + Netflix OSS

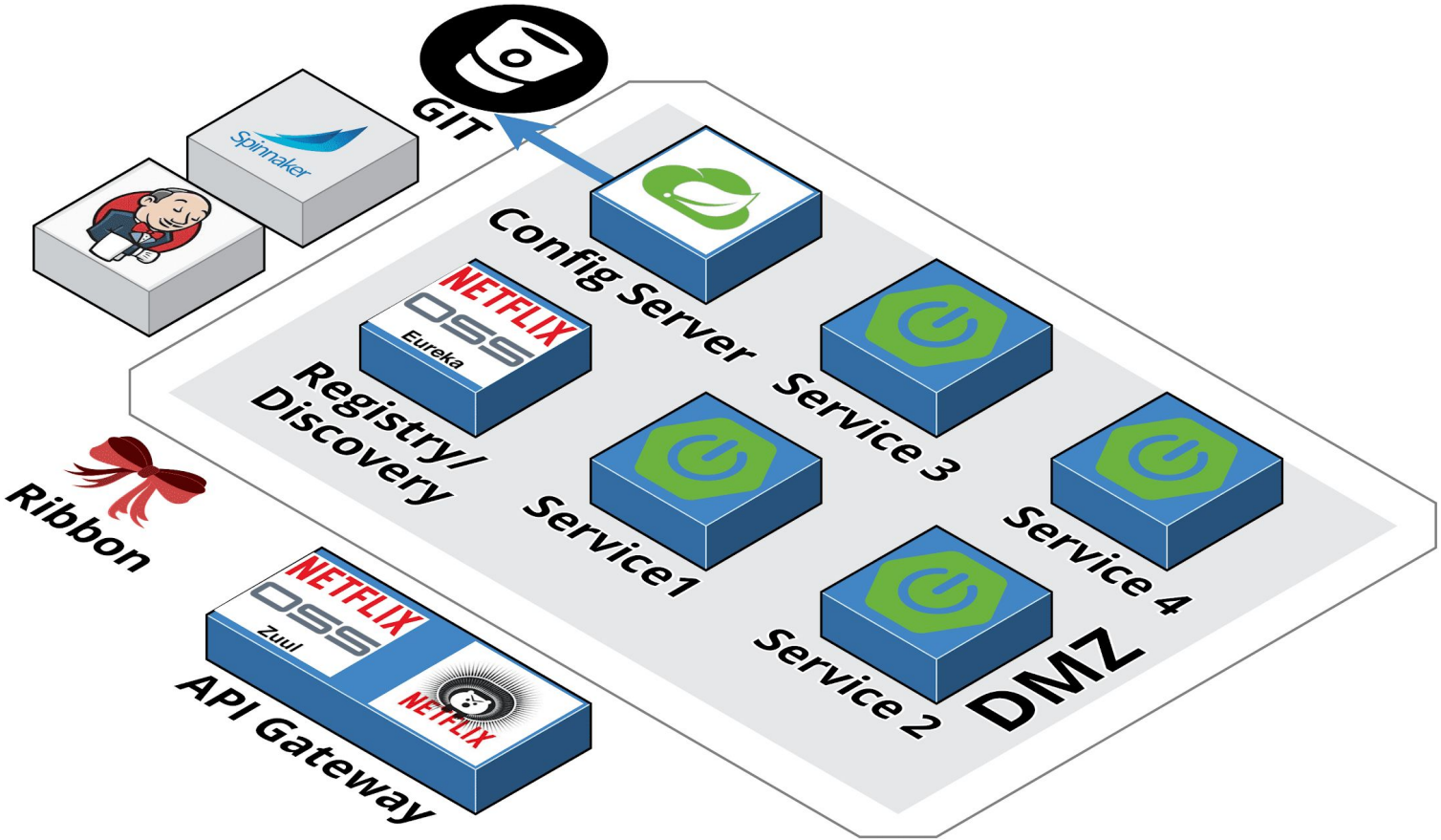
Java

Решения для микросервисной архитектуры

Тесная интеграция с Spring Framework

Тесная интеграция с AWS

Система непрерывной доставки кода

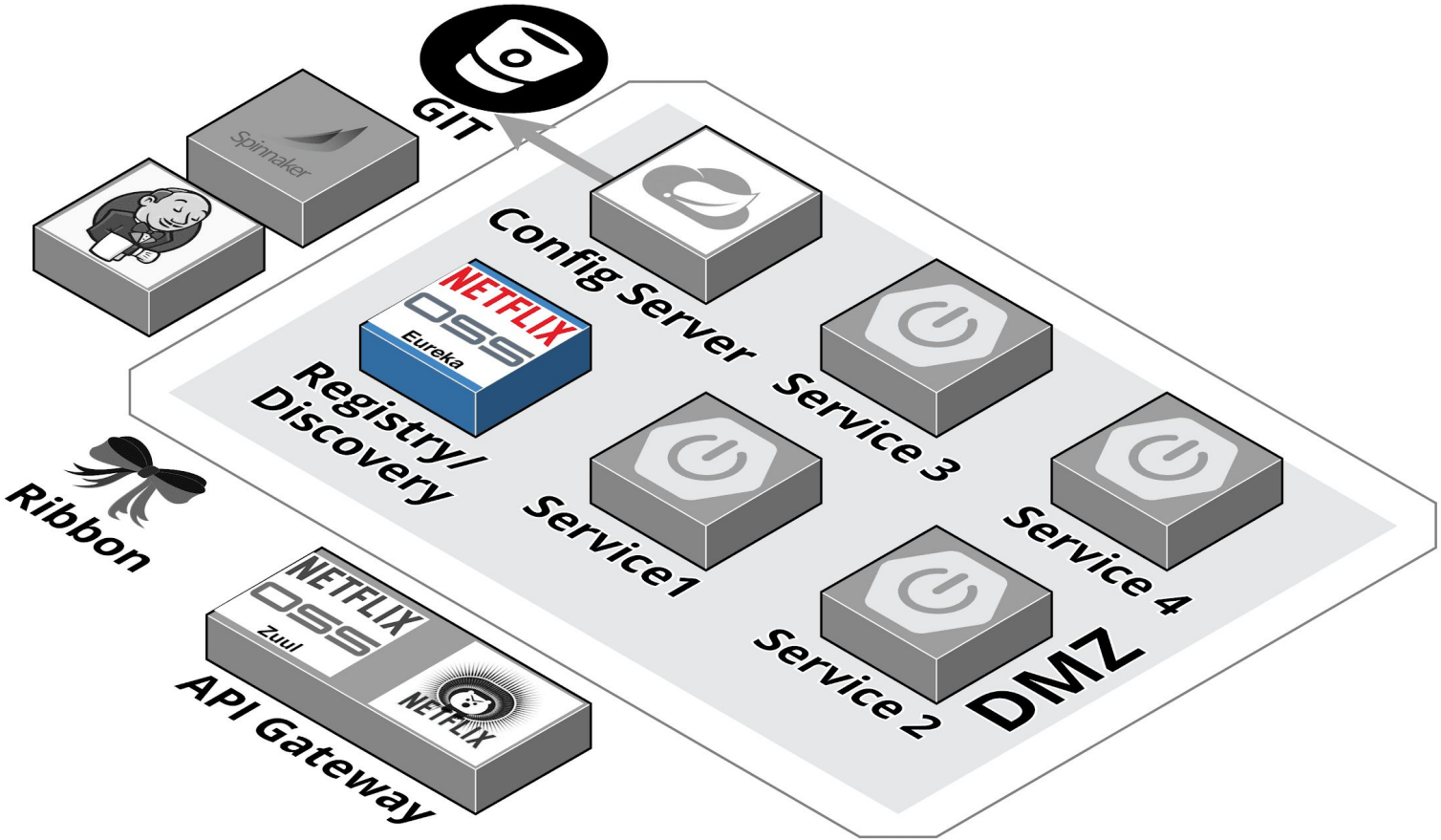

GIT
Spinnaker
Config Server
NETFLIX
OSS
Eureka
Registry/
Discovery
Service 3
Ribbon
Service1
Service 4
NETFLIX
OSS
Zuul
NETFLIX
Service 2
DMZ
API Gateway

Service Registry

Netflix Eureka - собирает и раздает адреса сервисов

Feign - декларативный REST клиент

Spring Boot Actuator - предоставляет информацию жив сервис или мертв


GIT
Spinnaker
Config Server
NETFLIX
OSS
Eureka
Registry/
Discovery
Service 3
Ribbon
Service1
Service 4
NETFLIX
OSS
Zuul
NETFLIX
Service 2
DMZ
API Gateway

Service Registry: Сервер

```
@EnableEurekaServer
@SpringBootApplication
public class ConfigServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Service Registry: Сервер

```
@EnableEurekaServer
@SpringBootApplication
public class ConfigServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Service Registry: Клиент

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients(...)
public class RateServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Service Registry: Клиент

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients(...)
public class RateServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Service Registry: Клиент

```
@FeignClient("allocation-service")
public interface AllocationClient {
    @RequestMapping(
      method = RequestMethod.GET,
      value = "/allocations")
    List<Allocation> getAllocations();
}
```

Service Registry: Клиент

```
@FeignClient("allocation-service")
public interface AllocationClient {
    @RequestMapping(
      method = RequestMethod.GET,
      value = "/allocations")
    List<Allocation> getAllocations();
}
```

Service Registry: грабли

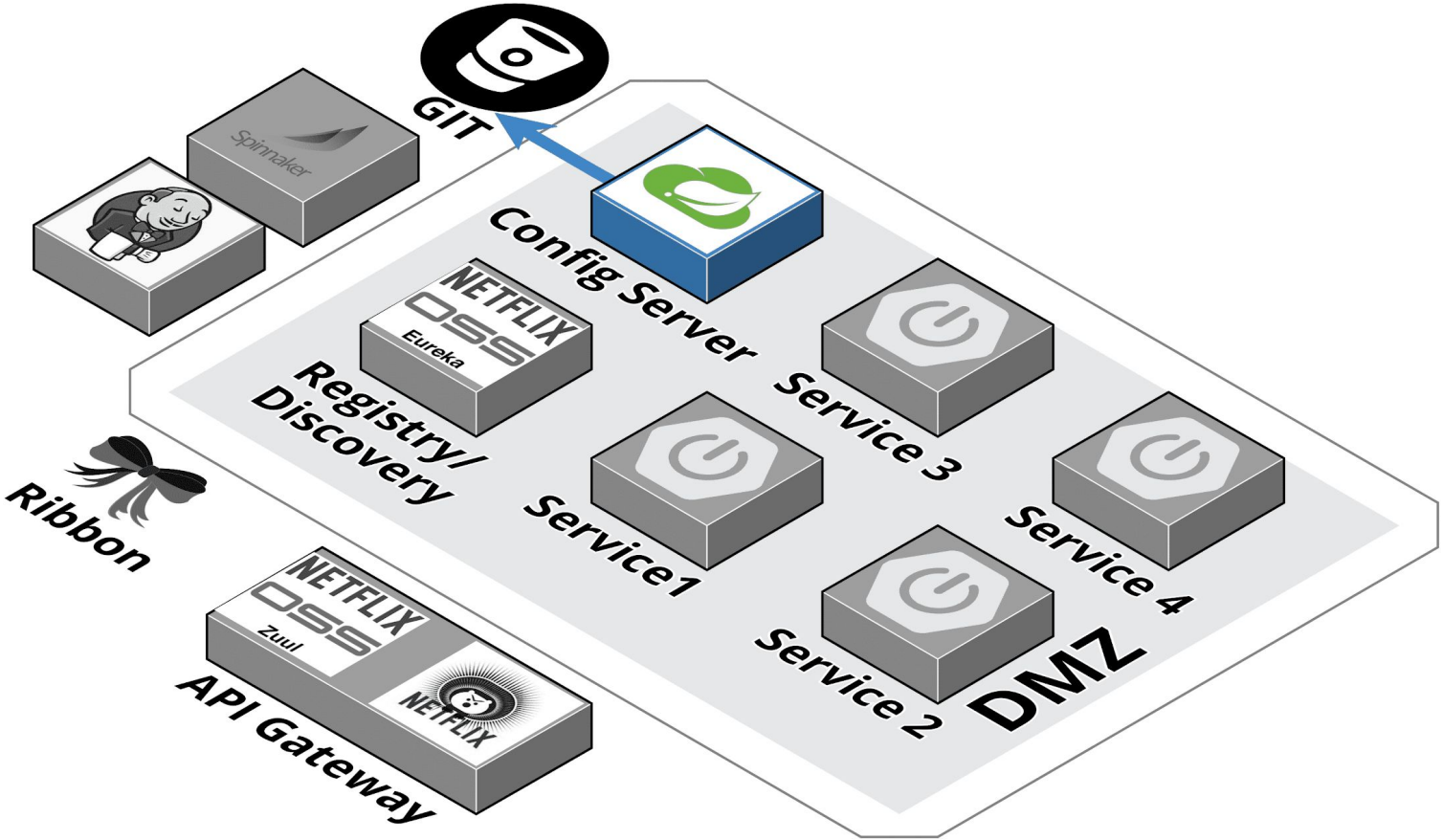
❌ Eureka Server ответственна за всю коммуникацию между сервисам - может стать точкой отказа

Service Registry: грабли

Eureka сервер должен быть развернут в режиме высокой доступности (High Availability)

Централизованная конфигурация

Spring Cloud Config Server и конфигурация в GIT


GIT
Spinnaker
Config Server
NETFLIX OSS
Eureka
Registry/
Discovery
Service 3
Ribbon
Service1
Service 4
NETFLIX OSS
Zuul
NETFLIX
Service 2
DMZ
API Gateway

Config Server

```
@EnableConfigServer
@EnableEurekaServer
@SpringBootApplication
public class ConfigServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Config Server

```
@EnableConfigServer
@EnableEurekaServer
@SpringBootApplication
public class ConfigServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Config Server

application.yaml

spring.cloud.config.server.git.uri: https://company.com/private/repo

Git backend

rates-dev.yaml

rates.config.acris: GHTY

Config Server

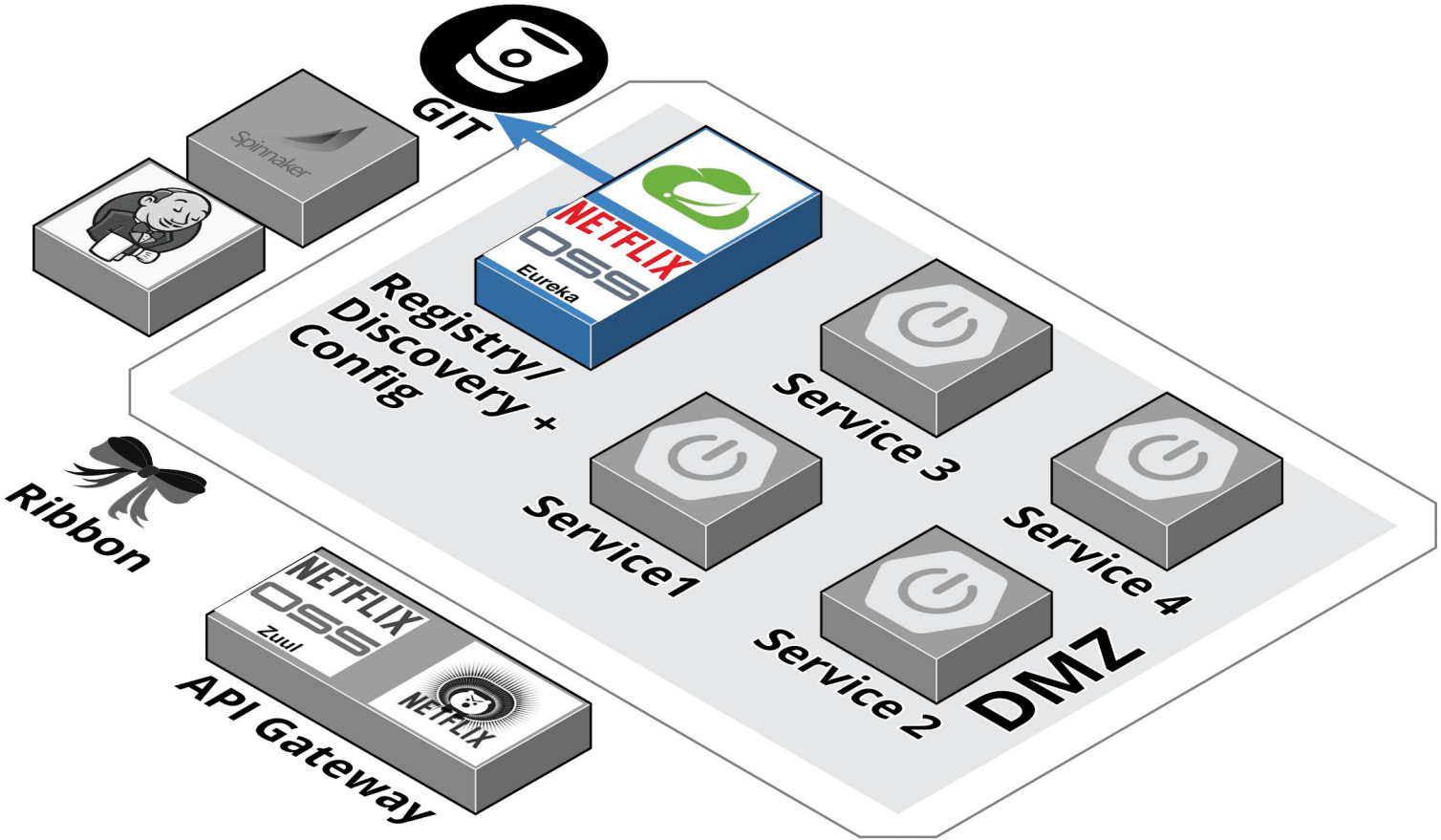
application.yaml

```
spring.cloud.config.server.git.uri: https://company.com/private/repo
```

Git backend

rates-dev.yaml

```
rates.config.acris: GHTY
```


GIT
Spinnaker
NETFLIX
OSS
Eureka
Registry/
Discovery +
Config
Service 3
Service1
Service 4
Service 2
DMZ
Ribbon
NETFLIX
OSS
Zuul
NETFLIX
API Gateway

Config Server: грабли

❌ Config Server зависит от системы контроля версии GIT - точка отказа

Config Server: грабли

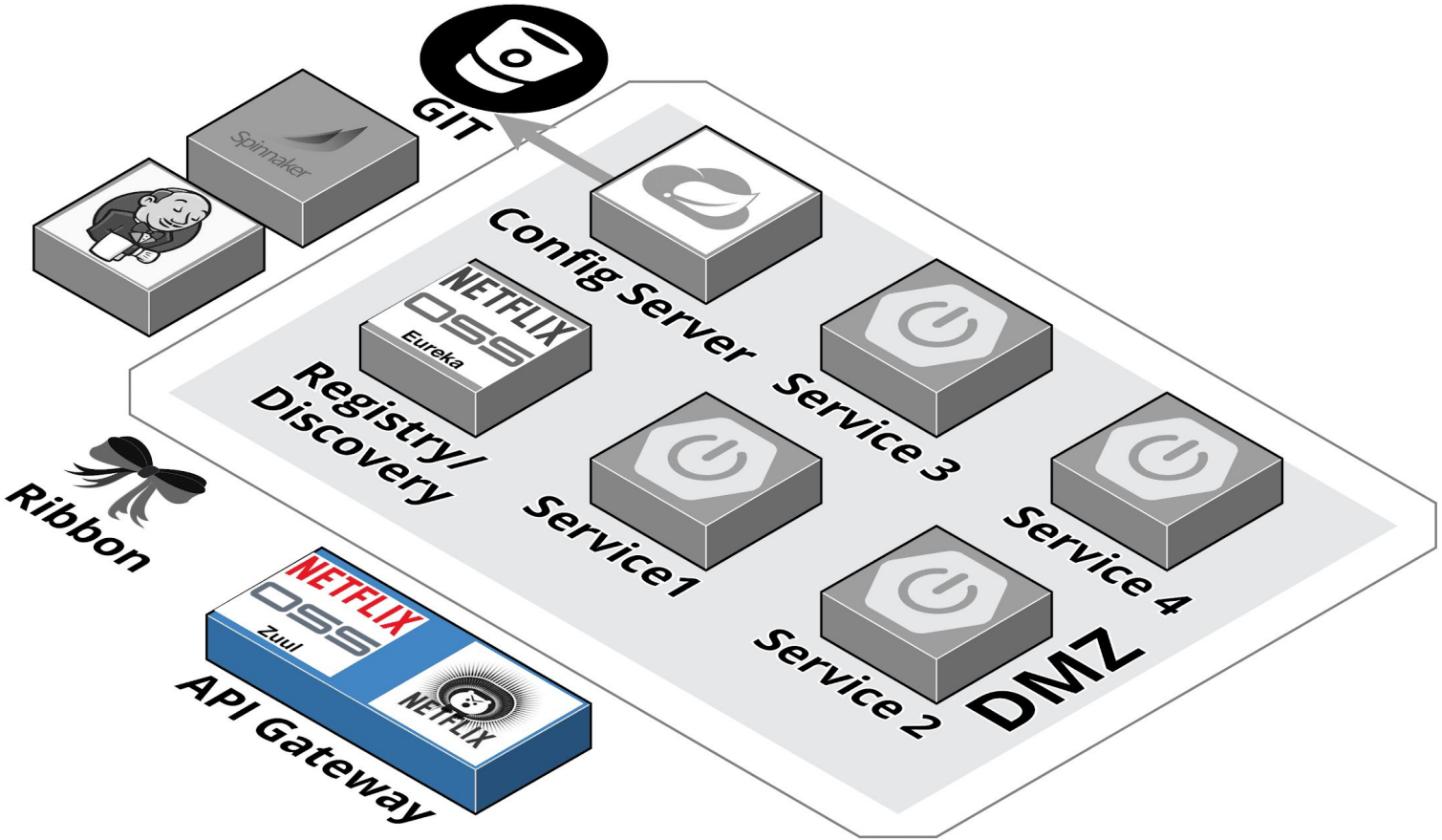
Config Server должен быть настроен на кэширование данных из Git

Config Server: грабли

❌ Хранение секретов требует настройки шифрования или использования сторонних сервисов

Api Gateway

Netflix Zuul 1 - проксирует трафик на сервисы по имени из Eureka Server


GIT
Spinnaker
Config Server
NETFLIX
OSS
Eureka
Registry/
Discovery
Service 3
Ribbon
Service1
Service 4
NETFLIX
OSS
Zuul
NETFLIX
Service 2
DMZ
API Gateway

Api Gateway: Сервер

```
@EnableZuulProxy
@SpringBootApplication
public class GatewayApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Api Gateway: Сервер

```
@EnableZuulProxy
@SpringBootApplication
public class GatewayApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Api Gateway: конфиг

```
zuul:
  routes:
    allocations:
      path: /allocations/**
      serviceId: allocation-service
    rates:
      path: /rates/**
      serviceId: rates-service
```

Api Gateway: конфиг

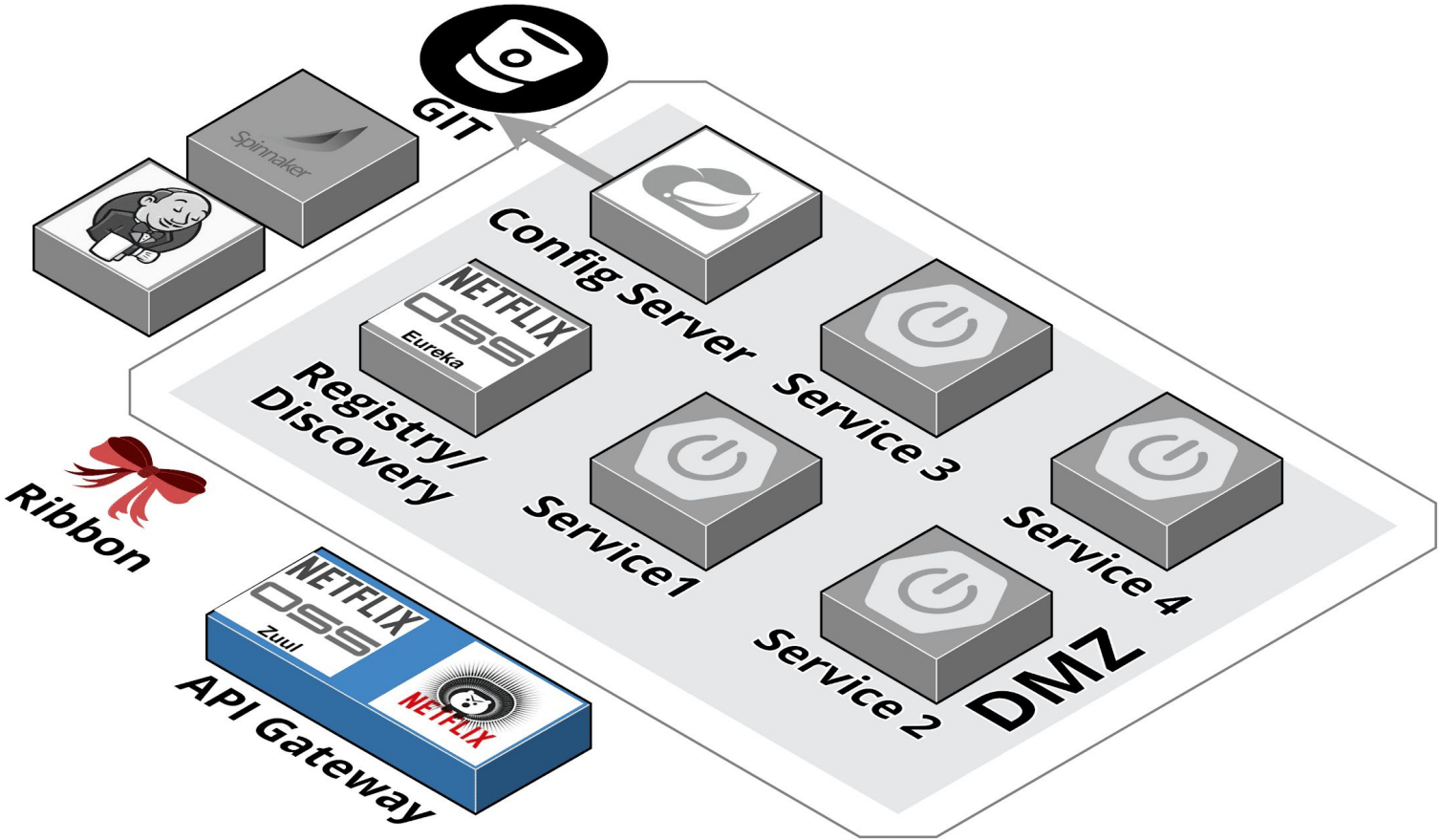
```
zuul:
  routes:
    allocations:
      path: /allocations/**
      serviceId: allocation-service
    rates:
      path: /rates/**
      serviceId: rates-service
```

Отказоустойчивость

Паттерн “Прерыватель” (Circuit breaker) – Hystrix

Ribbon – балансировать на стороне клиента

Spring Retry – повторять неудачные запросы


GIT
Spinnaker
Config Server
NETFLIX
OSS
Eureka
Registry/
Discovery
Service 3
Ribbon
Service1
Service 4
NETFLIX
OSS
Zuul
NETFLIX
Service 2
DMZ
API Gateway

Hystrix Конфигурация

```
public class RateClientAdapter {

  @HystrixCommand(fallbackMethod = "getFromCache")
  public List<RateDto> getAcrisses(Long supplierId) {...}

  public List<RateDto> getFromCache(Long supplierId) {...}
}
```

Hystrix Конфигурация

```
public class RateClientAdapter {

  @HystrixCommand(fallbackMethod = "getFromCache")
  public List<RateDto> getAcrisses(Long supplierId) {...}

  public List<RateDto> getFromCache(Long supplierId) {...}
}
```

Ribbon конфигурация

```
@RibbonClients
@SpringBootApplication
@EnableRetry
@EnableDiscoveryClient
public class GatewayApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Ribbon конфигурация

```
@RibbonClients
@SpringBootApplication
@EnableRetry
@EnableDiscoveryClient
public class GatewayApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Ribbon конфигурация

```
@Configuration
public class RibbonConfig {
...
    @Bean
    @Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
    public IRule ribbonRule() {
        return new RetryRule();
    }
}
```

Ribbon конфигурация

```
@Configuration
public class RibbonConfig {
...
    @Bean
    @Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
    public IRule ribbonRule() {
        return new RetryRule();
    }
}
```

Spring Retry: конфигурация

```
@RibbonClients
@SpringBootApplication
@EnableRetry
@EnableDiscoveryClient
public class GatewayApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Spring Retry: конфигурация

```
@RibbonClients
@SpringBootApplication
@EnableRetry
@EnableDiscoveryClient
public class GatewayApiApplication {
    public static void main(String[] args) {
        SpringApplication.run(...);
    }
}
```

Отказоустойчивость: грабли

Баги в Netflix OSS

Время исправления для некоторых > 1 года

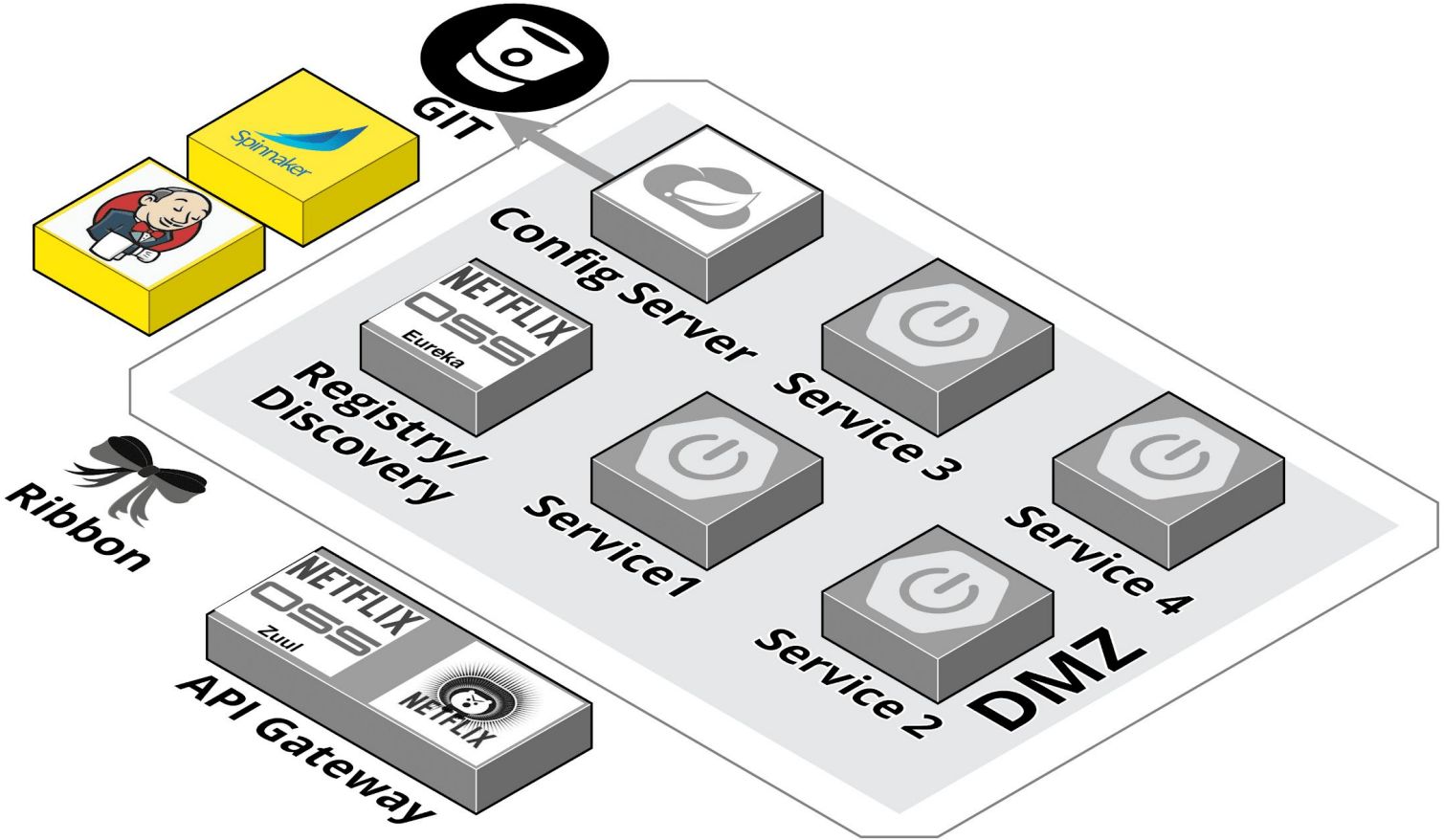
CI/CD: инструменты

Jenkins для компиляции, тестирования, сборки RPM пакета, размещения в RPM репозитории

Netflix Spinnaker для развертывания сервисов в AWS

CI/CD: концепт

Zero Downtime Deployment (Blue-Green)


GIT
Spinnaker
Config Server
NETFLIX OSS
Eureka
Registry/
Discovery
Service 3
Ribbon
Service1
Service 4
NETFLIX OSS
Zuul
NETFLIX
Service 2
DMZ
API Gateway

CI/CD: грабли

Надо знать 2 инструмента

Ручной запуск между Jenkins и Spinnaker

Blue-Green развертывание дорого с точки зрения ресурсов AWS

Ресурсы для системы V1

20 модулей:

- 15 бизнес-приложений по 2 реплики = 30 EC2 машин
- 1 Eureka+Config Server по 3 реплики = 3 EC2 машин
- 1 API Gateway 2 реплики = 2 EC2 машин
- 3 сервиса-оркестратора по 2 реплики = 6 EC2 машин

Ресурсы для системы V1

CI/CD:

- Jenkins + Agents = 3 EC2 машин
- Nexus OSS = 1 EC2 машина
- Spinnaker = 1 EC2 машина

Проблемы V1 решения

Spring Netflix в режиме поддержки

Уходит в режим поддержки с 12-12-2018

Только критические ошибки и небольшие исправления

Поддержка только в течение года от релиза Greenwich

Смена фокуса разработки

Концентрация на решении проблем взаимодействия сервисов между собой и обеспечения бесперебойной работы **вместо** реализации бизнес-задач

Сложность локальной разработки

Чтобы развернуть приложение для локальной разработки необходимо:

- ресурсы: процессор и память
- синхронизация старта сервисов
- конфигурация для локального запуска

Разный принцип развертывания при локальной разработке и на окружении

Размер пакета

Весь код по работе в микросервисной архитектуре находится в сервисе

Размер пакета с бизнес-логикой ~ 100-130 MB

Большое время старта сервиса ~ 120 секунд

Конфликты зависимостей

Утилизация ресурсов

Каждый сервис разворачивался как процесс на отдельной виртуальной машине.

Утилизация ~ 30-35%

Использование AWS ELB для распределения нагрузки на каждый сервис

Высокая стоимость инфраструктуры в AWS ~ 5000 USD

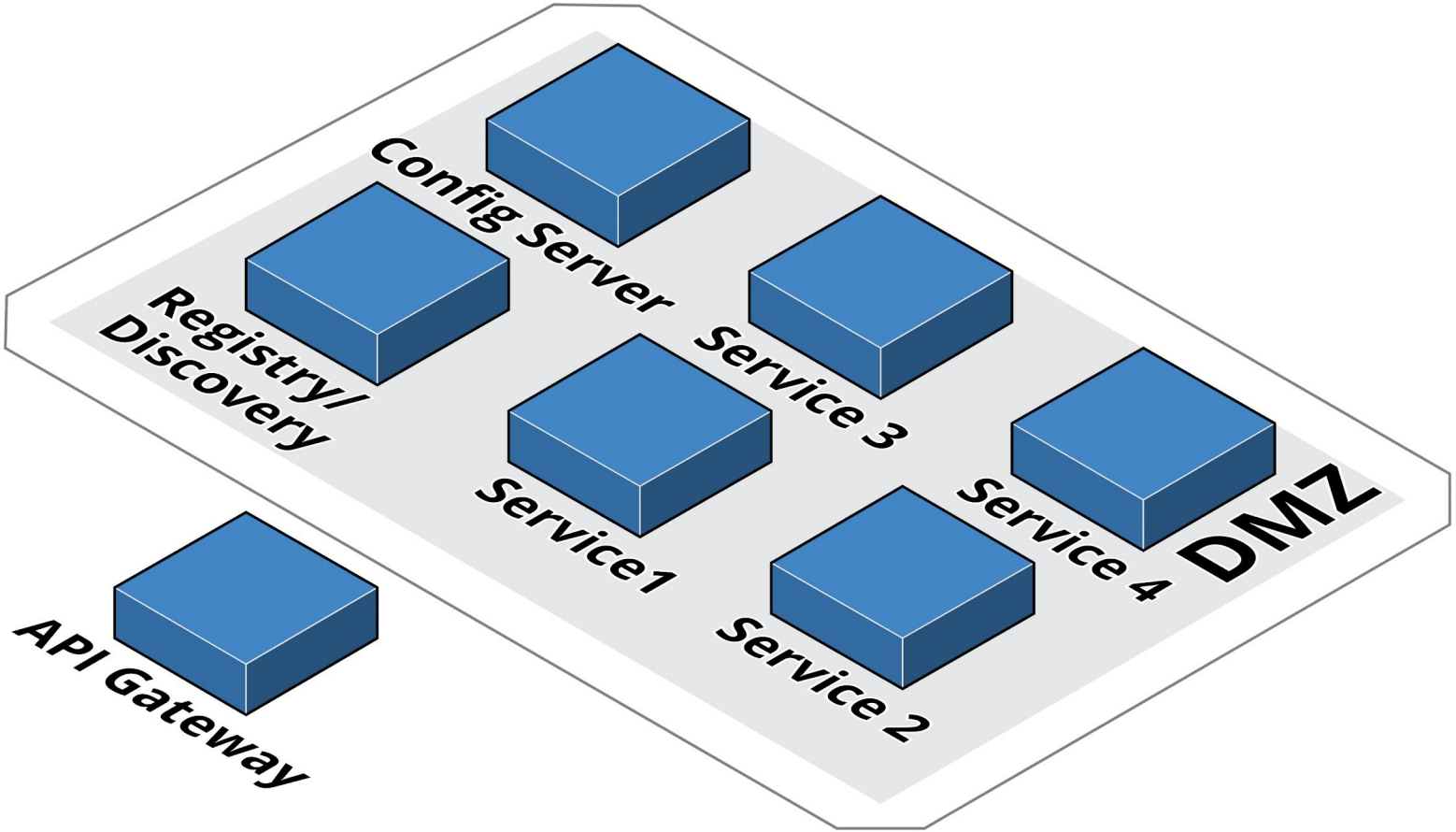
Почему Kubernetes (k8s)

Контейнеризация

Docker - единая среда для разработки и окружения

Управление контейнерами - Kubernetes

Пакетный менеджер - Helm 2 содержит > 400 пакетов приложений для K8s в стабильном статусе


Config Server
Registry/
Discovery
Service 3
Service1
Service 4
DMZ
Service 2
API Gateway

Service Registry/Discovery

Kubernetes Service API - объединяет инстансы приложения
CoreDNS - внутренний DNS сервер
Spring Boot Actuator - предоставляет информацию жив сервис или мертв

API Gateway

Kubernetes Ingress API - проксирует трафик внутрь кластера

Отказоустойчивость

Прямой замены Hystrix + Ribbon + Spring Retry нет

Использовать концепт “сервисное сито” (c)

JPoint О конференции Программа Тренинги Спонсоры Нормы поведения Контакты EN Купить билет

Cloud native, service-meshed Java Enterprise with Istio

День 2 / 11:15 / Зал 3 / EN / / kubernetes,cloudnative,servicemesh

Комментарий Программного комитета:

Кто-то использует Consul, кто-то — Eureka, кто-то — что-то другое. А что, если сделать межсервисное общение прозрачным и всю ответственность вынести на уровень инфраструктуры? Облегчит ли это разработку и поможет ли в мониторинге? Себастьян расскажет про cloud native-подход в исполнении Istio и Kubernetes.

In enterprise software, we see more and more of the cloud native technologies, especially container orchestration and service meshes, emerging and slowly taking over the market. Developers are facing the challenge which technology to choose to implement microservices for a cloud native setting. Java Enterprise has been used for software solutions for a long time and its APIs are well-established in the ecosystem. However, is it possible to develop cloud native, service-meshed Java Enterprise applications that fulfill concerns such as scalability, resiliency, and telemetry — in an effective, manageable way?

Все доклады

DaschnerS

Sebastian Daschner

IBM

Sebastian Daschner is a Lead Java Developer Advocate for IBM, author, and trainer and is enthusiastic about programming and Java (EE). He is the author of the book "Architecting Modern Java EE Applications". Sebastian is participating in the JCP, helping to form the future standards of Java EE, serving in the JAX-RS, JSON-P and Config Expert Groups and collaborating on various open source projects. For his contributions in the Java community and ecosystem, he was recognized as a Java Champion, Oracle Developer Champion and JavaOne Rockstar.

Besides Java, Sebastian is also a heavy user of Linux and container technologies like Docker. He evangelizes computer science practices on https://blog.sebastian-daschner.com, his newsletter, and on Twitter via @DaschnerS. When not working with Java, he also loves to travel the world —

Централизованная конфигурация

Kubernetes ConfigMap - хранение конфигурации

Kubernetes Secret - хранение секретов

CI/CD: инструменты

Jenkins 2

Helm 2 - пакетный менеджер

Google Jib/Kaniko - создание Docker образов без Docker daemon

CI/CD: концепт

Pipeline as a Code (Jenkinsfile)

Zero Downtime Deployment (Rolling update)

Основные отличия

Запуск и управление контейнерами, а не виртуальными машинами

Независимость от вендора облачных решений

Разделение логики взаимодействия сервисов и бизнес процессов в системе

Миграция на V2 и грабли

Вне доклада

Развертывание и настройка кластера для тест и прод окружения

Централизованный логгинг/мониторинг/алертинг

PoC сервиса на Java

Java 10
Spring Boot 2
REST CRUD
Spring Data JPA
Liquibase
OpenAPI

Jenkinsfile
Helm Chart
Google JIB

PoC сервиса на Java

Команда 7 Инженеров

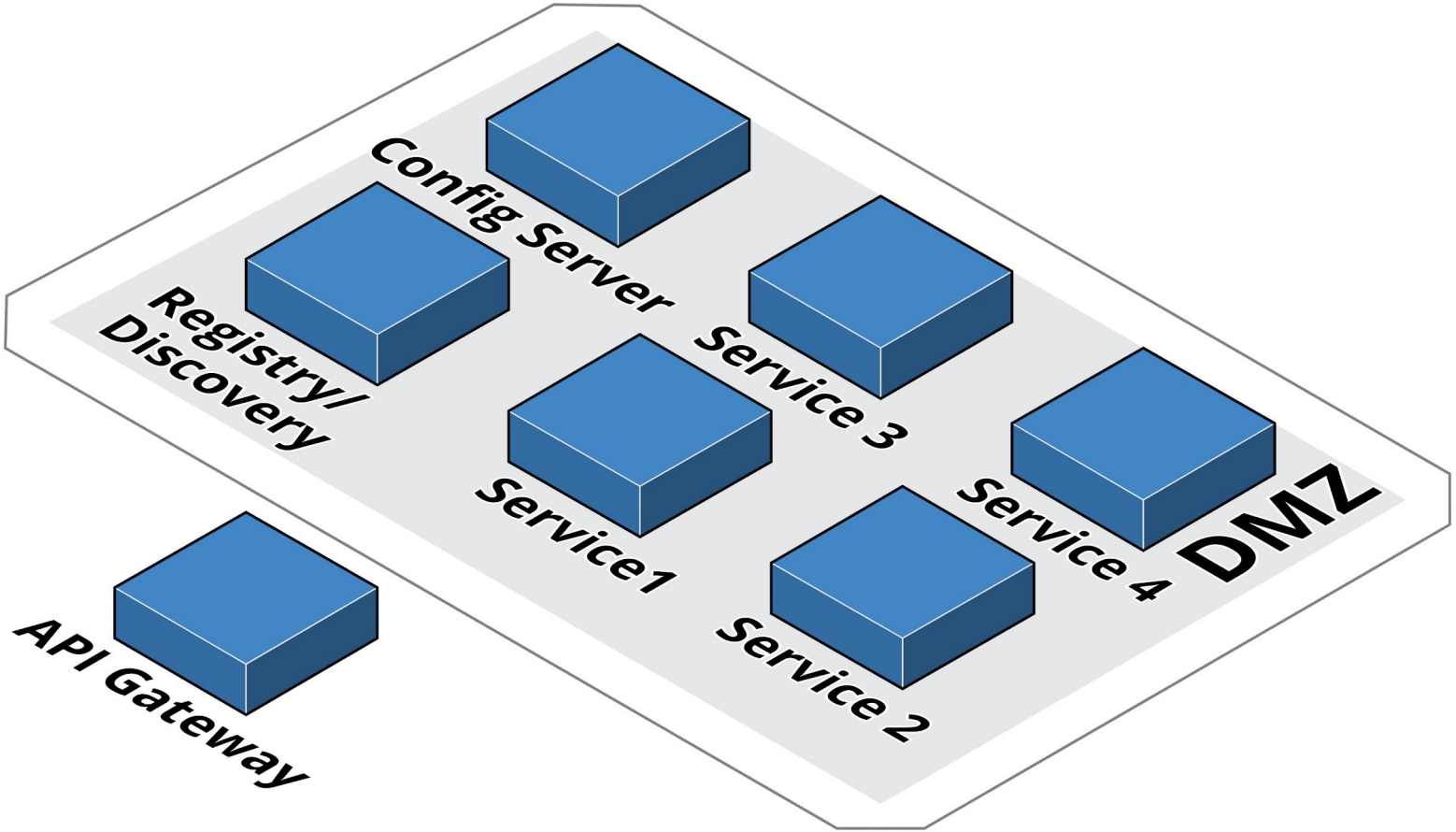
Реализован за 4 недели

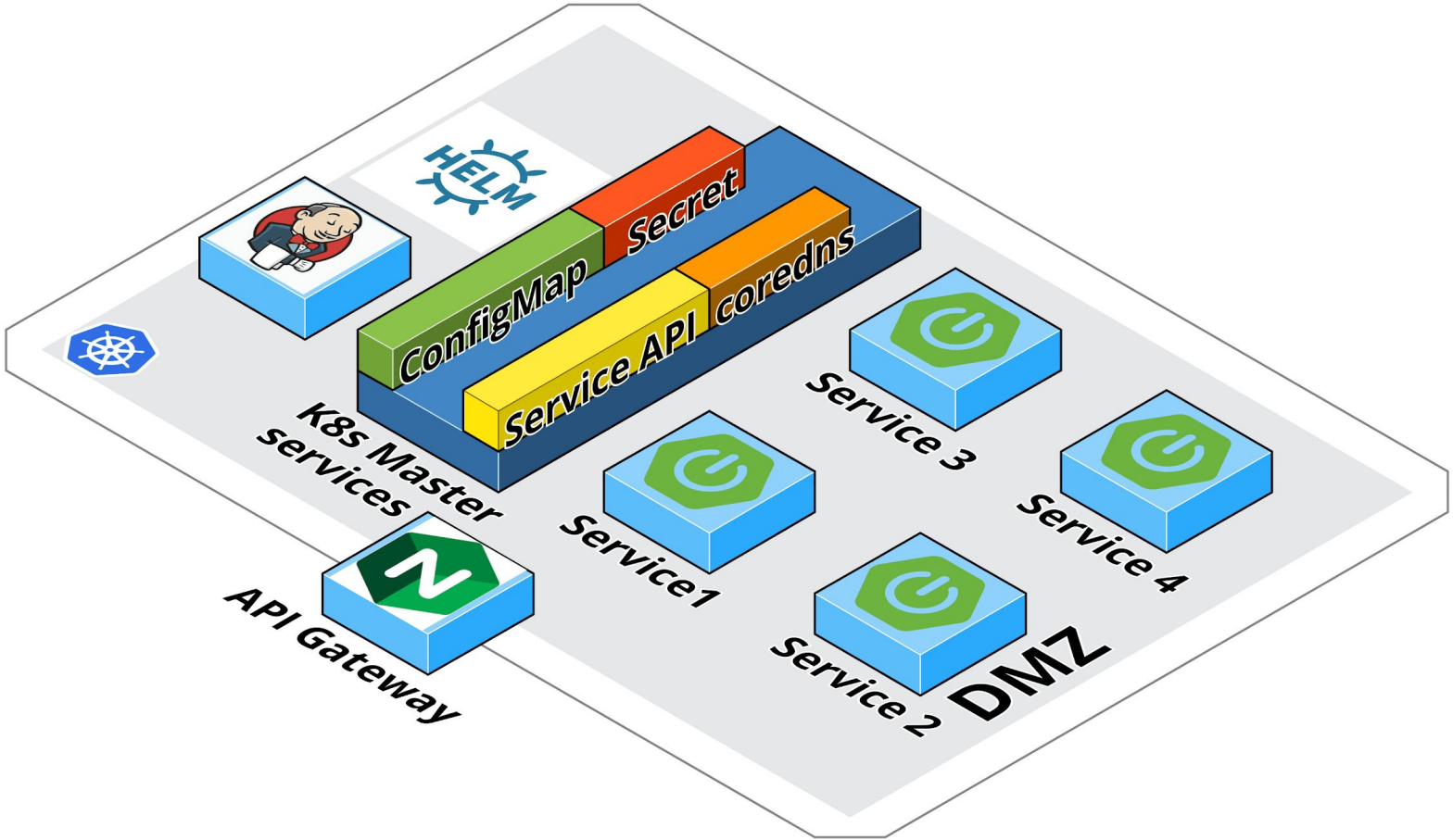
Тесты на Minikube/Docker for Windows

PoC сервиса на Java: грабли

Высокий порог вхождения в k8s:

- незнакомая терминология
- непривычная архитектура k8s компонентов
- новые инструменты (kubectl, helm)


Config Server
Registry/
Discovery
Service 3
Service1
Service 4
DMZ
API Gateway
Service 2


HELM
ConfigMap
Secret
Service API
coredns
K8s Master
services
Service 3
Service1
Service 4
API Gateway
Service 2
DMZ

CI/CD: инструменты

Jenkins 2

Kubernetes Jenkins Plugin - агенты Jenkins по запросу

Jenkins BlueOcean Plugin - визуализация пайплайна

Jenkins Kubernetes Credentials Provider plugin - хранение секретов для Jenkins в k8s

Nexus OSS - хранение Docker образов

CI/CD: инструменты

kubectl - утилита командной строки

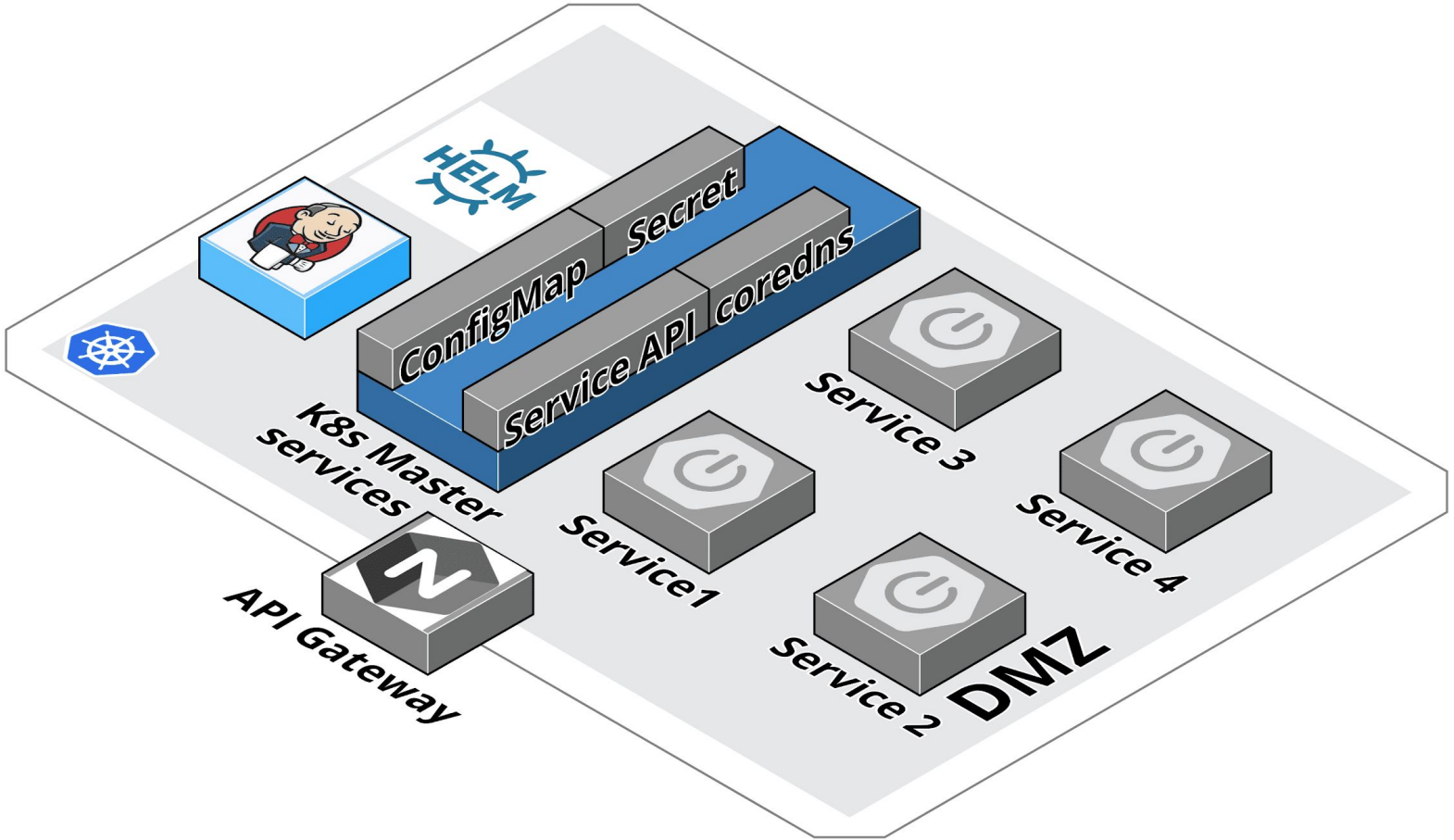
Helm 2 - пакетный менеджер

Google Jib/Kaniko - создание Docker образов без Docker daemon

CI/CD: концепт

Pipeline as Code - пайплайн представлен в виде программного кода (Jenkinsfile)

Zero Downtime Deployment (Rolling update)


HELM
ConfigMap
Secret
Service API
coredns
K8s Master services
Service 3
Service1
Service 4
API Gateway
Service 2
DMZ

CI/CD: грабли

Обновления K8s - отдельный небольшой кластер для CI/CD

Persistent Volume на AWS - AWS EFS

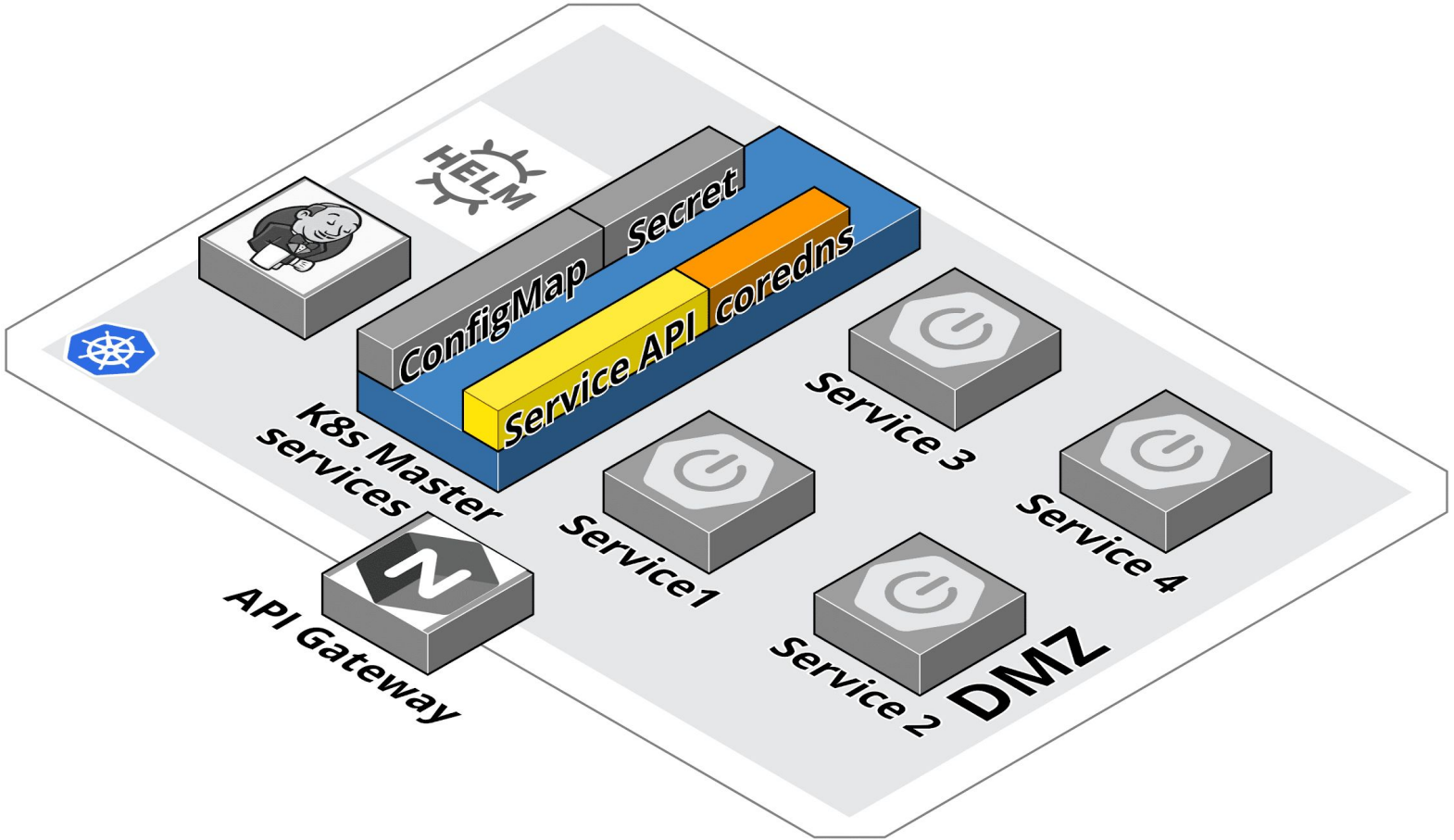
Helm таймаут

Helm опции хранения `[--history-max 20]`

Выбор инструментов

Service Registry

Kubernetes Service API
Kubelet healthcheker
CoreDNS
Spring Boot Actuator
Kubernetes Liveness probe


HELM
ConfigMap
Secret
Service API
coredns
K8s Master
services
Service 3
Service1
Service 4
API Gateway
Service 2
DMZ

Service API

```
apiVersion: v1
kind: Service
metadata:
  name: allocation-service
spec:
  selector:
    api: allocation
    tier: backend
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
    type: ClusterIP
```

Service API

```
apiVersion: v1
kind: Service
metadata:
  name: allocation-service
spec:
  selector:
    api: allocation
    tier: backend
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

К сервису внутри кластера можно обращаться

http://allocation-service:8080

Liveness Probe

```
apiVersion: apps/v1
kind: Deployment
...
spec:
...
  template:
    spec:
      containers:
      - livenessProbe:
          httpGet:
            path: /allocations/health
            port: 8080
```

Liveness Probe

```
apiVersion: apps/v1
kind: Deployment
...
spec:
...
  template:
    spec:
      containers:
      - livenessProbe:
          httpGet:
            path: /allocations/health
            port: 8080
```

Kubelet healthchecker делает запрос в контейнер:

`http://<ip>:8080/allocations/health`

Liveness Probe: грабли

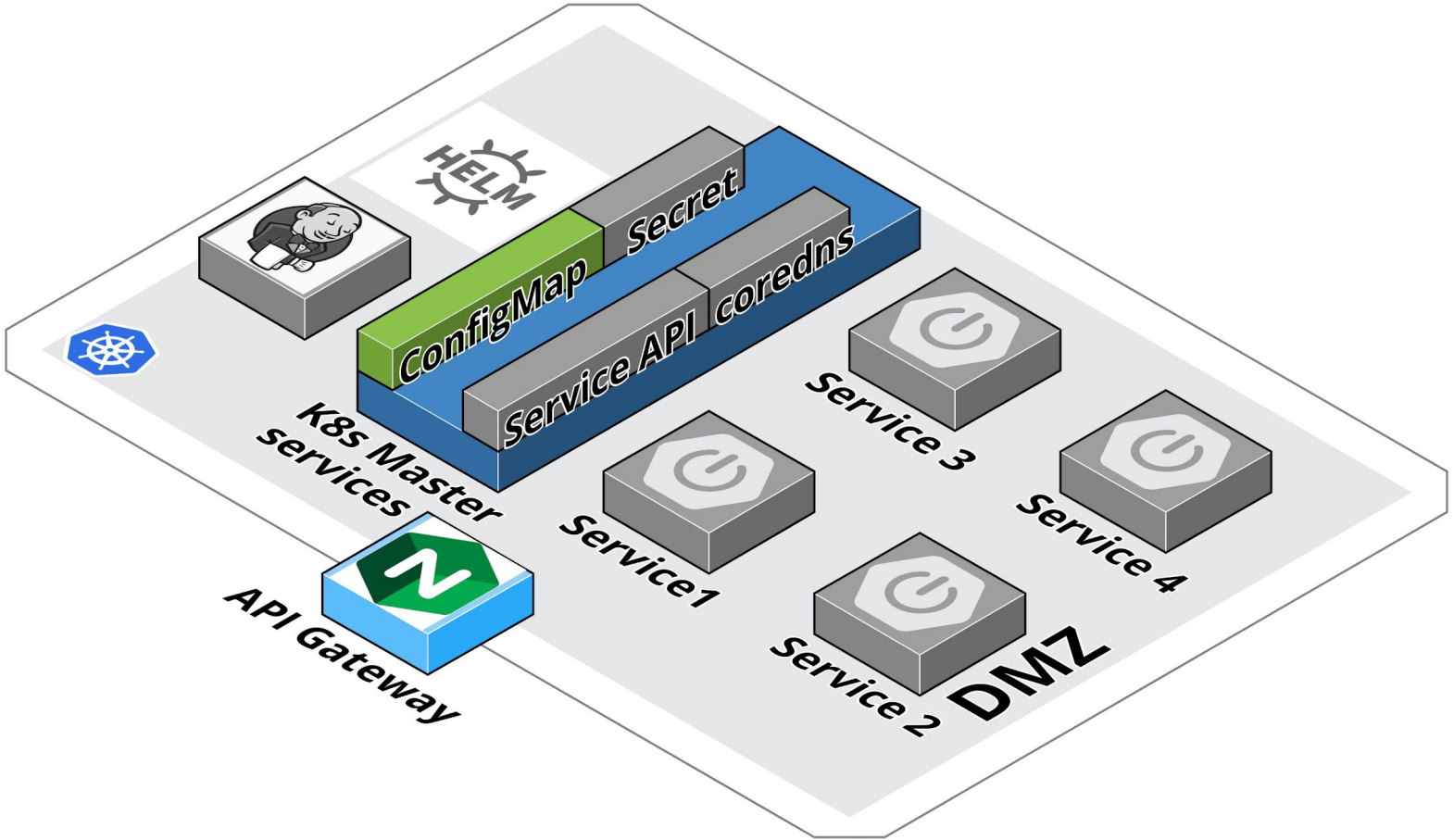
Liveness Probe задержка > Readiness probe задержка

- Liquibase
- K8s считал что контейнер не отвечает и надо создать новый
- процесс в контейнере прерывался
- в БД оставалась метка о том, что идет процесс обновления

API Gateway

Kubernetes Ingress API

Nginx Ingress Controller - использование nginx как реверс прокси


HELM
ConfigMap
Secret
Service API
coredns
K8s Master
services
Service 3
Service1
Service 4
API Gateway
Service 2
DMZ

Ingress API

```
apiVersion:
extensions/v1beta1
kind: Ingress
metadata:
  name: allocation
spec:
  rules:
  - host: api.company.com
    http:
     paths:
     - path: /allocations
       backend:
       serviceName: allocation-service
       servicePort: 8080
```

Ingress API

```
apiVersion:
extensions/v1beta1
kind: Ingress
metadata:
  name: allocation
spec:
  rules:
  - host: api.company.com
    http:
     paths:
     - path: /allocations
       backend:
         serviceName: allocation-service
         servicePort: 8080
```

К сервису вне кластера можно обращаться
http://**api.company.com/allocations**

Ingress API: грабли

❌ Использует контейнер (под) с nginx для проксирования трафика. Если контейнер упадет, то доступа через Ingress не будет

Ingress API: грабли

Должен быть включен режим высокой доступности - несколько контейнеров (подов) в разных дата цетрах

Ingress API: грабли

❌ Разделение общедоступных и внутренних приложений

K8s Ingress: грабли

Использование 2 Ingress Controller со своими политиками безопасности

K8s Ingress: грабли

❌ Верификация SSL сертификата

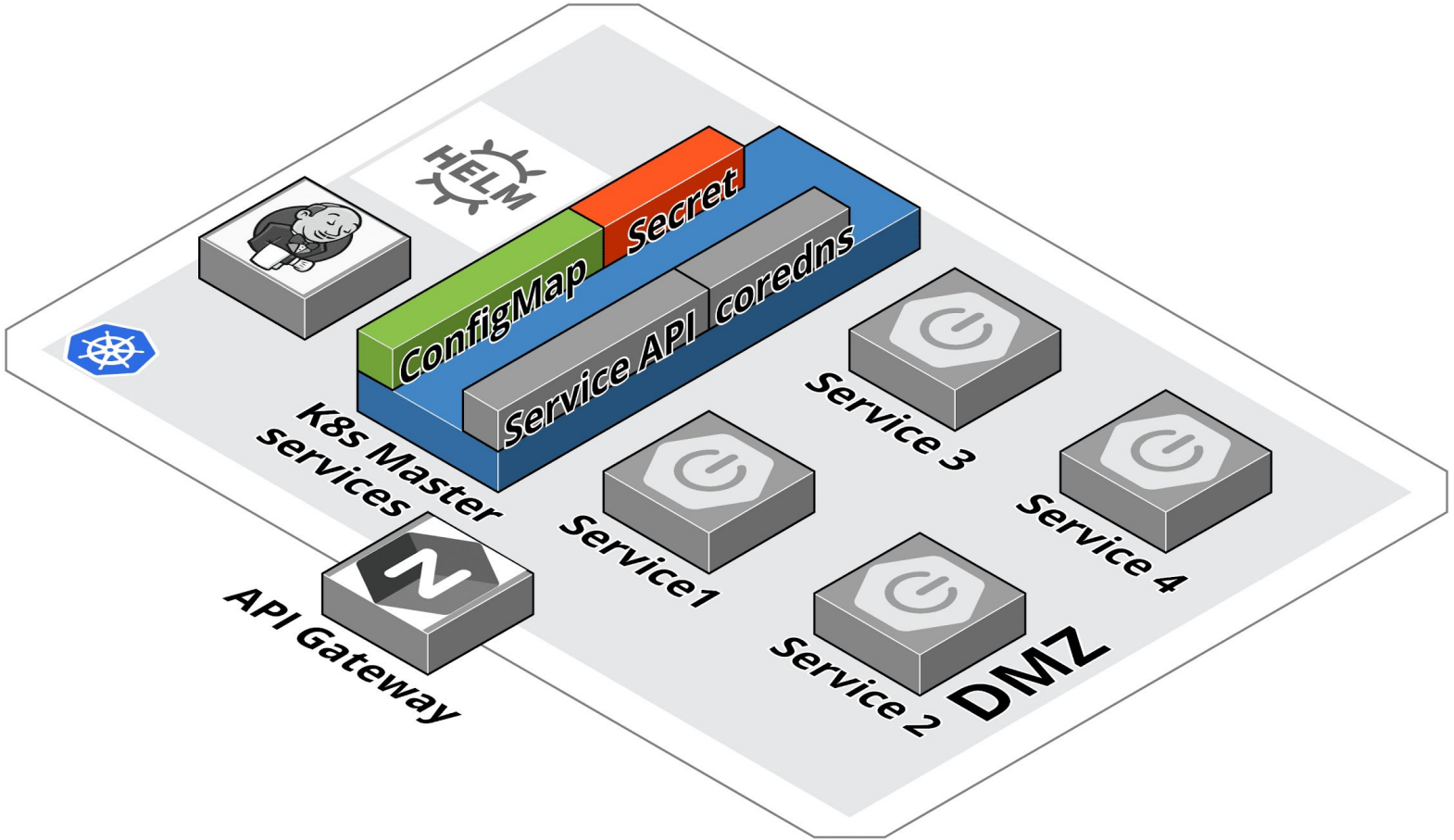
K8s Ingress: грабли

Использование AWS Certificate Manager

Централизованная конфигурация

Kubernetes ConfigMap хранение Spring конфигурации

Kubernetes Secret - хранение секретов


HELM
Secret
ConfigMap
Service API
coredns
K8s Master
services
Service 3
Service 4
Service1
Service 2
DMZ
API Gateway

ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: allocation-config
data:
  config.yaml: |-
     spring:
       jpa:
          database: mysql
     rates:
          url: http://rates-service
```

```
apiVersion: apps/v1
kind: Deployment
spec:
  ...
      - volumeMounts:
        - name: config-volume
          mountPath: /opt/config/
      volumes:
      - name: config-volume
        configMap:
          name: allocation-config
```

ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: allocation-config
data:
  config.yaml: |-
     spring:
       jpa:
          database: mysql
     rates:
          url: http://rates-service
```

```
apiVersion: apps/v1
kind: Deployment
spec:
  ...
       - volumeMounts:
         - name: config-volume
           mountPath: /opt/config/
       volumes:
       - name: config-volume
         configMap:
           name: allocation-config
```

Secret

```
apiVersion: v1
kind: Secret
metadata:
  name: allocation-secret
type: Opaque
stringData:
  secret.yaml: |-
     spring:
       datasource:
          username: allocationdb
          password: password
```

```
apiVersion: apps/v1
kind: Deployment
spec:
  ...
      - volumeMounts:
        - name: secret-volume
          mountPath: /opt/secret/
      volumes:
      - name: secret-volume
        secret:
          secretName: allocation-secret
```

Secret

```
apiVersion: v1
kind: Secret
metadata:
  name: allocation-secret
type: Opaque
stringData:
  secret.yaml: |-
     spring:
       datasource:
          username: allocationdb
          password: password
```

```
apiVersion: apps/v1
kind: Deployment
spec:
  ...
       - volumeMounts:
         - name: secret-volume
           mountPath: /opt/secret/
       volumes:
       - name: secret-volume
         secret:
           secretName: allocation-secret
```

Централизованная конфигурация: грабли

Ограничения etcd на размер ConfigMap 1Mb

K8s решил наши проблемы?

Единая точка отказа

❌ Kubernetes API - если мастер ноды упадут - все не работает

Единая точка отказа

Мастер машины должны быть развернуты в режиме высокой доступности (High Availability)

✔ Размер пакета

Пакет содержит только код бизнес логики

Размер пакета ~ 50 MB

Быстрое время старта сервиса:

- ~30-40 секунд со скриптами миграции
- ~10-20 секунд на актуальную БД

Баги в Kubernetes

Существуют баги, в том числе и критические

Время исправления критических уязвимостей мало

Огромное сообщество разработчиков

Открытый исходный код

Документация

“Хайповая” технология

Смена фокуса разработки

Концентрация на решении бизнес задач

Сложность локальной разработки

Развернуть приложение для локальной разработки:

- Локальный кластер (Minikube, Docker for Windows)
- одинаковый принцип развертывания при локальной разработке и на окружении

✔ Простой CI/CD

Jenkins

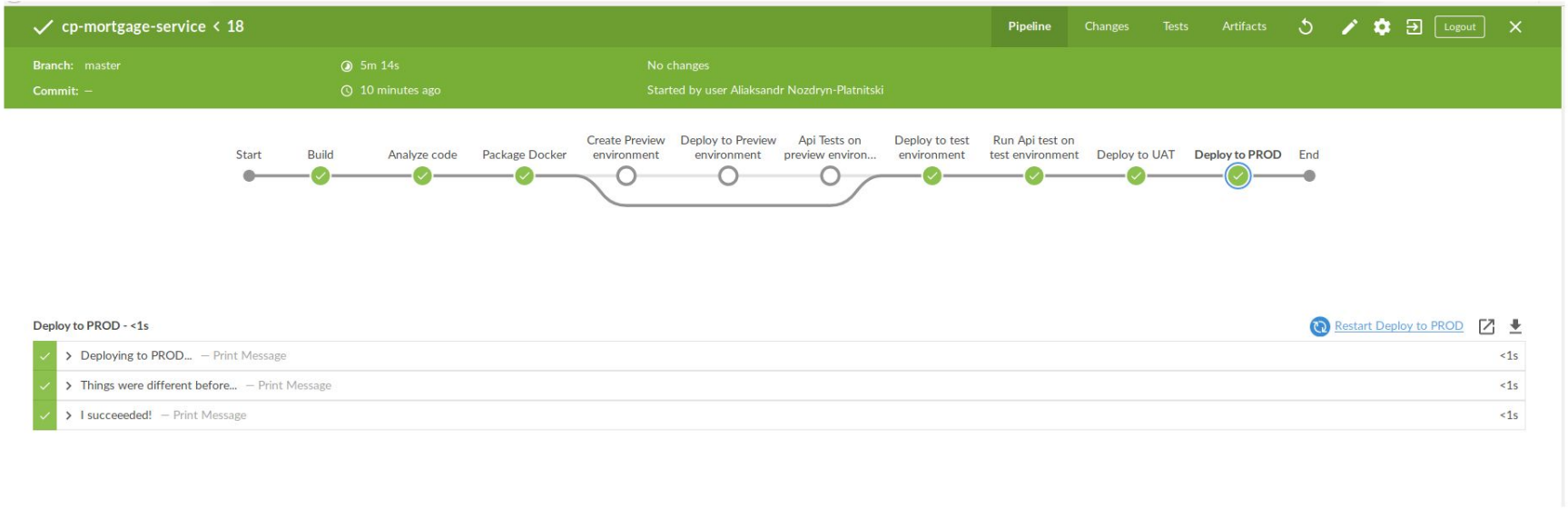
Полностью автоматизированный процесс

Интегрирован с системой контроля кода GIT

Continuous Delivery подход

Бесперебойное развертывание (Zero Downtime deployment)

Минимум сторонних инструментов (kubectl, helm)


cp-mortgage-service < 18
Pipeline
Changes
Tests
Artifacts
Logout
Branch: master
Commit: –
5m 14s
10 minutes ago
No changes
Started by user Aliaksandr Nozdryn-Platnitski
Start
Build
Analyze code
Package Docker
Create Preview environment
Deploy to Preview environment
Api Tests on preview environ...
Deploy to test environment
Run Api test on test environment
Deploy to UAT
Deploy to PROD
End
Deploy to PROD - <1s
Restart Deploy to PROD
Deploying to PROD... — Print Message
<1s
Things were different before... — Print Message
<1s
I succeeeded! — Print Message
<1s

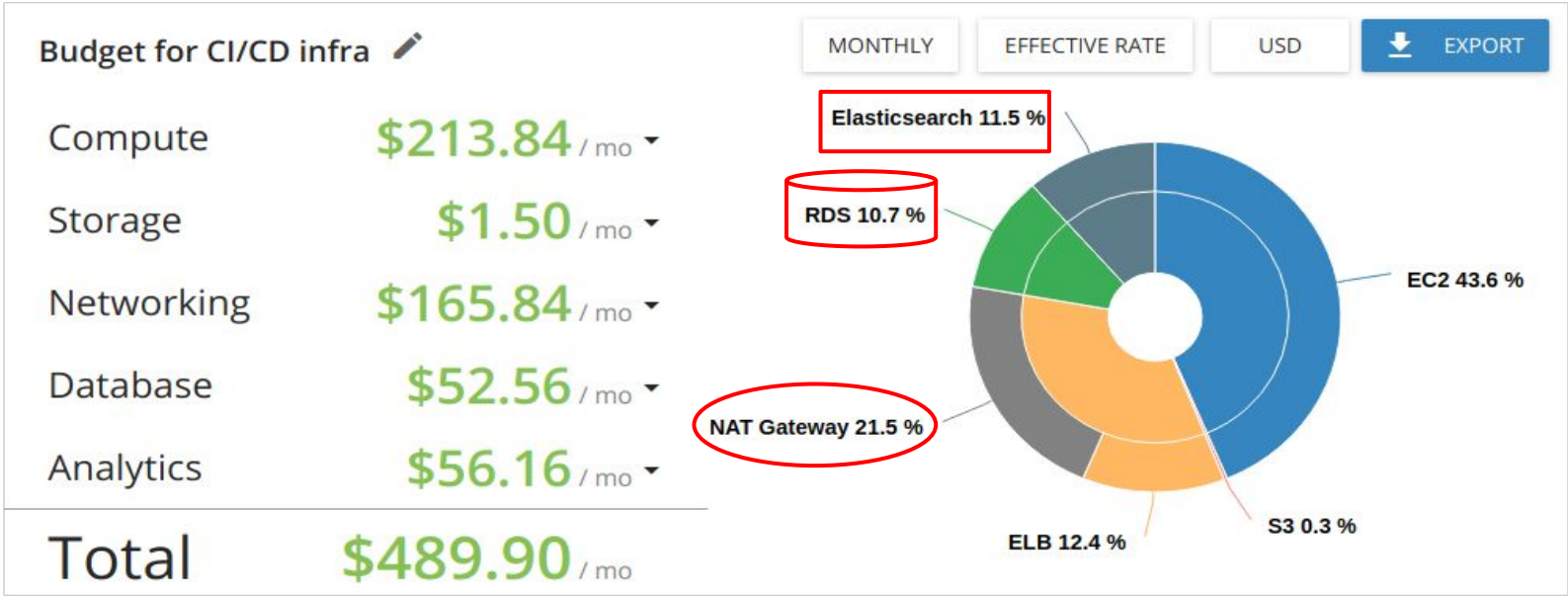
✔ Утилизация ресурсов

Каждый сервис разворачивался как контейнер на узле кластера.

Утилизация ~ 85-90%

Значительное снижение стоимости инфраструктуры в AWS

- все тестовые контуры ~ 500 USD (~31 775 RUR)


Budget for CI/CD infra
MONTHLY
EFFECTIVE RATE
USD
EXPORT
Compute $213.84 / mo
Storage $1.50 / mo
Networking $165.84 / mo
Database $52.56 / mo
Analytics $56.16 / mo
Total $489.90 / mo
Elasticsearch 11.5 %
RDS 10.7 %
EC2 43.6 %
NAT Gateway 21.5 %
S3 0.3 %
ELB 12.4 %

Kubernetes поддержка

Один из самых активных проектов на GitHub

Разнообразие инструментов

Выводы

6 месяцев в продакшене

Первый сервис был реализован за две недели

Новый микросервис создается из шаблона за 1 час

❌ Кривая обучения

Сложная система

Команда должна понимать принципы работы Docker контейнеров, java в Docker контейнерах

Знание терминологии обязательно, все должны “говорить на одном языке”

Единая ответственность

Разделение инфраструктурной и бизнес логики

Код чище и понятнее

Помогает сконцентрировать фокус команды на разработке бизнес задач

Более частые фича релизы - счастливый клиент

✔ Воспроизводимость

Прод окружение легко воспроизвести со своими настройками даже на локальной машине (Minikube, Docker for Windows)

Проще воспроизводить ошибки, получать доступ к логам, метрикам, статистике

✔ Стандарт

Фундамент микросервисной архитектуры

Поддержка всеми ведущими облачными вендорами

Работает одинаково (?) на AWS, GKE, AKS, Bare metal

Не привязан к языку программирования

Клиенты любят гибкость в выборе

Клиент выбрал как фундамент для своей инфраструктуры

Спасибо!

Вопросы