Платформенный сервис без хардкода

CEL в конфиге и как его готовить

Артемий Андрианов

Go-разработчик в платформенной команде, YADRO

- Разрабатываю сервисы AAA пользователей
- Последние 2 года пишу на Golang

Common Yadro Platform

Платформенная команда

Мы разрабатываем дистрибутив и сервисы, которые используют продукты компании. А еще у нас есть инженерный портал

Common Yadro Platform

Платформенная команда

Мы разрабатываем дистрибутив и сервисы, которые используют продукты компании. А еще у нас есть инженерный портал

Мы создаем универсальные сервисы

Главная задача – разработать и поддержать требования продуктов и сделать это эффективно и безопасно

Common Yadro Platform

Платформенная команда

Мы разрабатываем дистрибутив и сервисы, которые используют продукты компании. А еще у нас есть инженерный портал

Мы создаем универсальные сервисы

Главная задача – разработать и поддержать требования продуктов и сделать это эффективно и безопасно

Просто CYP

Сокращенно в компании команду называют CYP, а в простонародье ЦЫП, поэтому у нас цыпленок на логотипе

А по каким правилам нужно валидировать пароли?

А по каким правилам нужно валидировать пароли?

А по каким правилам нужно валидировать пароли?

Продукт B

длина пароля не меньше 8 и не больше 16;
пароль может содержать заглавные буквы, строчные буквы, специальные символы `-`, `_`, `.`

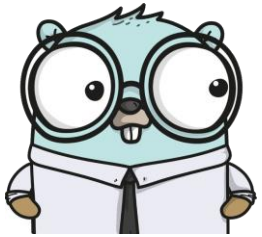
А по каким правилам нужно валидировать пароли?

Продукт В

длина пароля не меньше 8 и не больше 16;
пароль может содержать заглавные буквы, строчные буквы, специальные символы `-`, `_`, `.`

Продукт О

пароль должен содержать не менее одной заглавной буквы;
пароль должен содержать не менее двух строчных букв;
пароль должен содержать не менее трех цифр



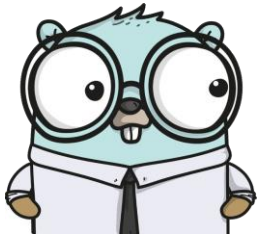
А по каким правилам нужно валидировать пароли?

Продукт В

длина пароля не меньше 8 и не больше 16;
пароль может содержать заглавные буквы, строчные буквы, специальные символы `-`, `_`, `.`

Продукт О

пароль должен содержать не менее одной заглавной буквы;
пароль должен содержать не менее двух строчных букв;
пароль должен содержать не менее трех цифр



Продукт Т

пароль не должен быть одним из списка: `admin`, `root`, `test`;
символы в пароле не должны повторяться более 2 раз

Как удовлетворить требования всех продуктов?

Решить задачу через хардкод проверок для разных продуктов

Решаем задачу через хардкод

```go
type ProductType int

const (
  ProductTypeB ProductType
  ProductTypeO
  ProductTypeT

)
```

types.go

```go
switch productType {
case ProductTypeB:
  // product B validation logic.
case ProductTypeO:
  // product O validation logic.
case ProductTypeT:
  // product T validation logic.


default:
  ...
}
```

validate.go

Решаем задачу через хардкод

```
type ProductType int

const (
  ProductTypeB ProductType
  ProductTypeO
  ProductTypeT
  ProductTypeX
)
```

types.go

```
switch productType {
case ProductTypeB:
  // product B validation logic.
case ProductTypeO:
  // product O validation logic.
case ProductTypeT:
  // product T validation logic.
case ProductTypeX:
  // product X validation logic.
default:
  ...
}
```

validate.go

Введение требований для нового сервиса потребует внесения изменений в метод с логикой проверки и в список сервисов

Решаем задачу через хардкод

```
type ProductImpl interface {
	validate(pass string) error
}

func validate[T ProductImpl](
	impl T,
	pass string,
) error {
	if err := impl.validate(pass); err != nil {
		return err
	}
	return nil
}
```

validate.go

Решаем задачу через хардкод

```go
type ProductImpl interface {
	validate(pass string) error
}

func validate[T ProductImpl](
	impl T,
	pass string,
) error {
	if err := impl.validate(pass); err != nil {
		return err
	}
	return nil
}
```

validate.go

Будем имплементировать интерфейс для каждого продукта

Решаем задачу через хардкод

```
type ProductImpl interface {
	validate(pass string) error
}

func validate[T ProductImpl](
	impl T,
	pass string,
) error {
	if err := impl.validate(pass); err != nil {
		return err
	}
	return nil
}
```

validate.go

Но что, если нам нужно будет изменить что-то в имплементации проверки для продукта?

~~Решить задачу через хардкод проверок для разных сервисов~~

Ввести динамически изменяемый regexp

Введем динамически изменяемый regexp

```
re := regexp.MustCompile(rule)

if !re.MatchString(password) {
    return errors.New("password validation failed")
}
```

validate.go

Введем динамически изменяемый regexp

```
re := regexp.MustCompile(rule)

if !re.MatchString(password) {
    return errors.New("password validation failed")
}
```

validate.go

```
password: ChangeMe123
rule: ^[A-Za-z1-9]{8,16}$
```

Введем динамически изменяемый regexp

```
re := regexp.MustCompile(rule)

if !re.MatchString(password) {
    return errors.New("password validation failed")
}
```

validate.go

```
password: ChangeMe123
rule: ^[A-Za-z1-9]{8,16}$
```

```
<nil>
```

Введем динамически изменяемый regexp

```
re := regexp.MustCompile(rule)

if !re.MatchString(password) {
    return errors.New("password validation failed")
}
```

validate.go

```
password: Change Me@%123
rule: ^[A-Za-z1-9]{8,16}$
```

Введем динамически изменяемый regexp

```
re := regexp.MustCompile(rule)

if !re.MatchString(password) {
    return errors.New("password validation failed")
}
```

validate.go

```
password: Change Me@%123
rule: ^[A-Za-z1-9]{8,16}$
```

```
Password validation
failed
```

Введем динамически изменяемый regexp

```go
re := regexp.MustCompile(rule)

if !re.MatchString(password) {
    return errors.New("password validation failed")
}
```

validate.go

```
password: Change Me@%123
rule: ^(?=.*[A-Z])(?=.*[a-z]).+$
```

Введем динамически изменяемый regexp

```
re := regexp.MustCompile(rule)

if !re.MatchString(password) {
    return errors.New("password validation failed")
}
```

validate.go

```
password: Change Me@%123
rule: ^(?=.*[A-Z])(?=.*[a-z]).+$
```

```
panic: regexp: Compile(`^(?=.*[A-Z])(?=.*[a-z]).+$`):
error parsing regexp: invalid or unsupported Perl
syntax:`(?=`
```

Пакет regexp в go поддерживает не весь синтаксис

~~Решить задачу через хардкод проверок для разных сервисов~~

~~Ввести динамически изменяемый regexp~~

Предоставить сервисам возможность самим писать правила в конфигурации

Выносим правила в конфигурацию

```
- длина пароля >= 8
- длина пароля <= 16
- пароль может содержать заглавные буквы
- пароль может содержать строчные буквы
- пароль может содержать специальные
символы `-`, `_`, `.`
```

Правила для продукта B

Выносим правила в конфигурацию

```
- длина пароля >= 8
- длина пароля <= 16
- пароль может содержать заглавные буквы
- пароль может содержать строчные буквы
- пароль может содержать специальные
символы `-`, `_`, `.`
```

Правила для продукта В

```
- пароль должен содержать не менее одной
заглавной буквы
- пароль должен содержать не менее двух
строчных букв
- пароль должен содержать не менее трех цифр
```

Правила для продукта О

Выносим правила в конфигурацию

```
- длина пароля >= 8
- длина пароля <= 16
- пароль может содержать заглавные буквы
- пароль может содержать строчные буквы
- пароль может содержать специальные
символы `-`, `_`, `.`
```

Правила для продукта B

```
- пароль должен содержать не менее одной
заглавной буквы
- пароль должен содержать не менее двух
строчных букв
- пароль должен содержать не менее трех цифр
```

Правила для продукта O

```
- пароль не должен быть один списка:
  `admin`, `root`, `test`
- символы в пароле не должны повторяться
более 2 раз
```

Правила для продукта T

Выносим правила в конфигурацию

```
- длина пароля >= 8
- длина пароля <= 16
- пароль может содержать заглавные буквы
- пароль может содержать строчные буквы
- пароль может содержать специальные
символы `-`, `_`, `.`
```

```
- пароль должен содержать не менее одной
заглавной буквы
- пароль должен содержать не менее двух
строчных букв
- пароль должен содержать не менее трех цифр
```

```
- пароль не должен быть один списка:
  `admin`, `root`, `test`
- символы в пароле не должны повторяться
более 2 раз
```

Напрашивается использование domain-specific language

Common Expression Language

Common Expression Language

Простота и читаемость

CEL имеет синтаксис схожий с Golang, Python, C и другими языками программирования

Common Expression Language

Простота и читаемость

CEL имеет синтаксис схожий с Golang, Python, C и другими языками программирования

Строгая типизация

CEL строго-типизированный и содержит необходимый набор типов и методов для них

Common Expression Language

Простота и читаемость

CEL имеет синтаксис схожий с Golang, Python, C и другими языками программирования

Строгая типизация

CEL строго-типизированный и содержит необходимый набор типов и методов для них

Интеграция с Go

Для CEL разработан пакет, позволяющий быстро и безопасно внедрить функционал в уже существующий проект

Common Expression Language

Простота и читаемость

CEL имеет синтаксис схожий с Golang, Python, C и другими языками программирования

Строгая типизация

CEL строго-типизированный и содержит необходимый набор типов и методов для них

Интеграция с Go

Для CEL разработан пакет, позволяющий быстро и безопасно внедрить функционал в уже существующий проект

Расширяемость

Набор методов для типов может быть расширен для решения задач валидации определенных продуктов

Common Expression Language

Простота и читаемость

CEL имеет синтаксис схожий с Golang, Python, C и другими языками программирования

Строгая типизация

CEL строго-типизированный и содержит необходимый набор типов и методов для них

Мы уже используем его

Мы используем CEL при описании API – а значит все, кто работают с нашими сервисами – так или иначе знакомы с CEL

Интеграция с Go

Для CEL разработан пакет, позволяющий быстро и безопасно внедрить функционал в уже существующий проект

Расширяемость

Набор методов для типов может быть расширен для решения задач валидации определенных продуктов

Common Expression Language

Простота и читаемость

CEL имеет синтаксис схожий с Golang, Python, C и другими языками программирования

Строгая типизация

CEL строго-типизированный и содержит необходимый набор типов и методов для них

Мы уже используем его

Мы используем CEL при описании API – а значит все, кто работают с нашими сервисами – так или иначе знакомы с CEL

Интеграция с Go

Для CEL разработан пакет, позволяющий быстро и безопасно внедрить функционал в уже существующий проект

Расширяемость

Набор методов для типов может быть расширен для решения задач валидации определенных продуктов

прим: и да, это можно было сделать на Lua, Rego и тд

Научим Go работать с CEL

```
import "github.com/google/cel-go/cel"
```

Импортируем пакет cel-go

prepare.go

Научим Go работать с CEL

```go
import "github.com/google/cel-go/cel"

func prepare(
  rule string,
) (*cel.Program, error) {
  env, err := cel.NewEnv(
    cel.Variable("password", cel.StringType),
  )
  ...
```

prepare.go

Подготовим окружения для компиляции программы.
В данном случае – введем переменную `password` типа string

Научим Go работать с CEL

```
import "github.com/google/cel-go/cel"

func prepare(
  rule string,
) (*cel.Program, error) {
  env, err := cel.NewEnv(
    cel.Variable("password", cel.StringType),
  )
  ...
  ast, issues := env.Compile(rule)
  ...
```

prepare.go

Подготовим AST представление для компиляции на основе окружения

Научим Go работать с CEL

```go
import "github.com/google/cel-go/cel"

func prepare(
  rule string,
) (*cel.Program, error) {
  env, err := cel.NewEnv(
    cel.Variable("password", cel.StringType),
  )
  ...
  ast, issues := env.Compile(rule)
  ...
  prm, err := env.Program(ast)
  ...
  return &prm, nil
}
```

prepare.go

Скомпилируем CEL программу на основе окружения и AST представления

Научим Go валидировать пароли при помощи CEL

```
func validate(
  password string,
  prm cel.Program,
) error {
  v, _, err := prm.Eval(map[string]any{
    "password": password,
  })
  ...
```

Вызовем программу, передав значения CEL-переменных

validate.go

Научим Go валидировать пароли при помощи CEL

```
func validate(
  password string,
  prm cel.Program,
) error {
  v, _, err := prm.Eval(map[string]any{
    "password": password,
  })
  ...
  if !v.Value().(bool) {
    return errors.New("password validation
failed")
  }
  return nil
}
```

validate.go

Проверим полученный результат, если правило не выполняется – вернем ошибку

Перепишем правила на CEL

```
password_validation:
  rules:
    - длина пароля >= 8
    - длина пароля <= 16
    - пароль может содержать заглавные буквы
    - пароль может содержать строчные буквы
    - пароль может содержать спец. символы `-`, `_`, `.`
```

app_config.yaml

Правила для продукта B

Перепишем правила на CEL

```
password_validation:
  rules:
    - password.size() >= 8
    - password.size() <= 16
    - password.matches('^[A-Za-z0-9_.-]*$')
```

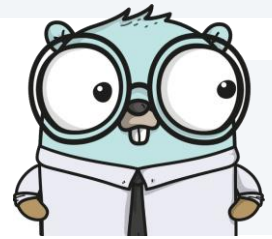
app_config.yaml

Правила для продукта B

Перепишем правила на CEL

```
password_validation:
  rules:
    - пароль должен содержать не менее одной заглавной буквы
    - пароль должен содержать не менее двух строчных букв
    - пароль должен содержать не менее трех цифр
```

app_config.yaml

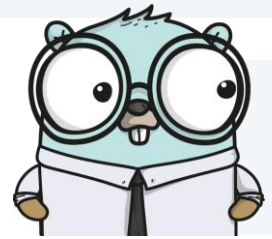


Правила для продукта О

Перепишем правила на CEL

```
password_validation:
  rules:
    - password.matches('^([^A-Z]*[A-Z][^A-Z]*){1,}$')
    - password.matches('^([^a-z]*[a-z][^a-z]*){2,}$')
    - password.matches('^([^0-9]*[0-9][^0-9]*){3,}$')
```

app_config.yaml



Правила для продукта О

Перепишем правила на CEL

```
password_validation:
  rules:
    - пароль не должен быть одним из списка: `admin`, `root`, `test`
    - символы в пароле не должны повторяться более 2 раз
```

app_config.yaml

Правила для продукта Т

Перепишем правила на CEL

```
password_validation:
  rules:
    - !(password in ['admin', 'root', 'dev’])
    - символы в пароле не должны повторяться более 2 раз
```

app_config.yaml

Нет метода, чтобы поддержать данный функционал

Правила для продукта Т

Введем новый CEL метод

prepare.go

```
env, err := cel.NewEnv(cel.Function(
    "maxRepeatCount",
    cel.MemberOverload(
      "string_maxRepeatCount_int",
      []*cel.Type{cel.StringType, cel.IntType},
      cel.BoolType,
      cel.BinaryBinding(func(lhs, rhs ref.Val) ref.Val {
        // Method implementation.
      }),
    ),
  ),
)
```

Введем новый CEL метод

prepare.go

```
env, err := cel.NewEnv(cel.Function(
    "maxRepeatCount",
    cel.MemberOverload(
      "string_maxRepeatCount_int",
      []*cel.Type{cel.StringType, cel.IntType},
      cel.BoolType,
      cel.BinaryBinding(func(lhs, rhs ref.Val) ref.Val {
        // Method implementation.
      }),
    ),
  ),
)
```

Имя метода

Введем новый CEL метод

prepare.go

```
env, err := cel.NewEnv(cel.Function(
    "maxRepeatCount",
    cel.MemberOverload(
      "string_maxRepeatCount_int",
      []*cel.Type{cel.StringType, cel.IntType},
      cel.BoolType,
      cel.BinaryBinding(func(lhs, rhs ref.Val) ref.Val {
        // Method implementation.
      }),
    ),
  ),
)
```

Объявления параметров метода и их порядок

Введем новый CEL метод

prepare.go

```
env, err := cel.NewEnv(cel.Function(
    "maxRepeatCount",
    cel.MemberOverload(
      "string_maxRepeatCount_int",
      []*cel.Type{cel.StringType, cel.IntType},
      cel.BoolType,
      cel.BinaryBinding(func(lhs, rhs ref.Val) ref.Val {
        // Method implementation.
      }),
    ),
  ),
)
```

Типы параметров метода

Введем новый CEL метод

prepare.go

```
env, err := cel.NewEnv(cel.Function(
    "maxRepeatCount",
    cel.MemberOverload(
      "string_maxRepeatCount_int",
      []*cel.Type{cel.StringType, cel.IntType},
      cel.BoolType,
      cel.BinaryBinding(func(lhs, rhs ref.Val) ref.Val {
        // Method implementation.
      }),
    ),
  ),
)
```

Тип возвращаемого значения

Введем новый CEL метод

prepare.go

```
env, err := cel.NewEnv(cel.Function(
    "maxRepeatCount",
    cel.MemberOverload(
      "string_maxRepeatCount_int",
      []*cel.Type{cel.StringType, cel.IntType},
      cel.BoolType,
      cel.BinaryBinding(func(lhs, rhs ref.Val) ref.Val {
        // Method implementation.
      }),
    ),
  ),
)
```

Реализация функции

Перепишем правила на CEL

```
password_validation:
  rules:
    - !(password in ['admin', 'root', 'dev’])
    - символы в пароле не должны повторяться более 2 раз
```

app_config.yaml

Правила для продукта Т

Перепишем правила на CEL

```
password_validation:
  rules:
    - !(password in ['admin', 'root', 'dev’])
    - password.maxRepeatCount(2)
```

app_config.yaml

Используем новый метод

Правила для продукта Т

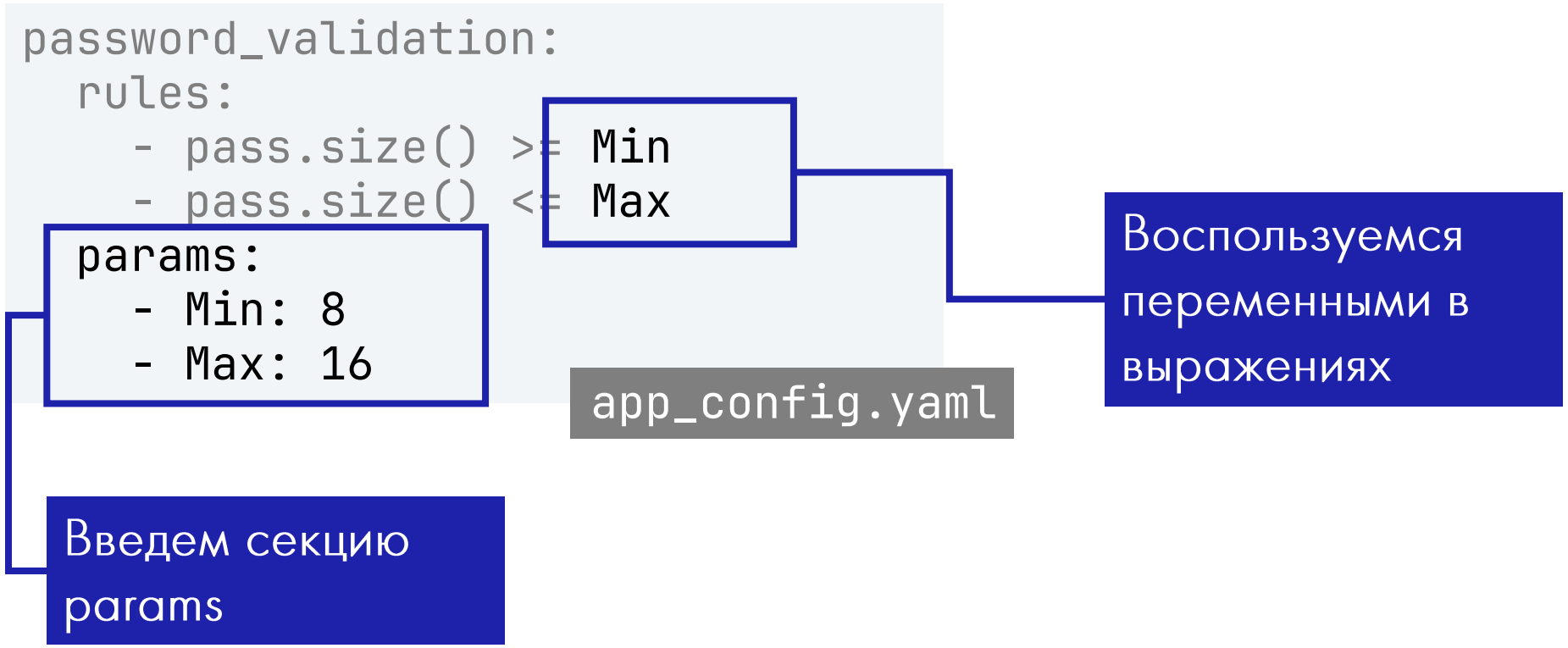
А можно ли сделать систему правил более функциональной?

Дадим возможность вводить переменные

```
password_validation:
  rules:
    - pass.size() >= 8
    - pass.size() <= 16
```

app_config.yaml

Дадим возможность вводить переменные



Дадим возможность вводить переменные

```
env, err := cel.NewEnv(
  cel.Variable(”pass", cel.StringType),
  ...
  cel.Variable(“Min”, ???)
  cel.Variable(“Max”, ???)
)
```

app_config.yaml

Добавим переменные в окружение программы и

Дадим возможность вводить переменные

```
env, err := cel.NewEnv(
  cel.Variable(”pass", cel.StringType),
  ...
  cel.Variable(“Min”, ???)
  cel.Variable(“Max”, ???)
)
```

app_config.yaml

Добавим переменные в окружение программы и

поймем, что CEL строго-типизированный :)

Как решить проблему строгой типизации?

Использовать маппинг типов

Маппинг типов?

```yaml
password_validation:
  params:
    - name: MinLen
      value: 8
      type: int
    - name: MaxLen
      value: 16
      type: int
```

app_config.yaml

```
func getType(varType string)
*cel.Type {
  switch varType {
  case “int”:
    return cel.IntType
  default:
    return cel.AnyType
  }
}
```

types.go

Ввести метод для маппинга типов и указывать типы в конфигурации переменной

Маппинг типов?

```yaml
password_validation:
  params:
    - name: MinLen
      value: 8
      type: int
    - name: MaxLen
      value: 16
      type: int
```

app_config.yaml

```go
func getType(varType string)
*cel.Type {
  switch varType {
  case “int”:
    return cel.IntType
  default:
    return cel.AnyType
  }
}
```

types.go

Ввести метод для маппинга типов и указывать типы в конфигурации переменной

А что, если появятся новые типы?

Маппинг типов?

```yaml
password_validation:
  params:
    - name: MinLen
      value: 8
      type: int
    - name: MaxLen
      value: 16
      type: int
```

app_config.yaml

```go
func getType(varType string)
*cel.Type {
  switch varType {
  case “int”:
    return cel.IntType
  default:
    return cel.AnyType
  }
}
```

types.go

Ввести метод для маппинга типов и указывать типы в конфигурации переменной

А что, если появятся новые типы?

А что, если ошибиться в конфигурации?

Маппинг типов?

```yaml
password_validation:
  params:
    - name: MinLen
      value: 8
      type: int
    - name: MaxLen
      value: 16
      type: int
```

app_config.yaml

```go
func getType(varType string)
*cel.Type {
  switch varType {
  case “int”:
    return cel.IntType
  default:
    return cel.AnyType
  }
}
```

types.go

Ввести метод для маппинга типов и указывать типы в конфигурации переменной

А что, если появятся новые типы?

А что, если ошибиться в конфигурации?

Но ведь это харкод!

~~Использовать маппинг типов~~

Использовать явное приведение типов

Явное приведение типов

```yaml
password_validation:
  rules:
    - pass.size() >= int(Min)
    - pass.size() <= int(Max)
  params:
    - name: Min
      value: 8
    - name: Max
      value: 16
```

app_config.yaml

```
env, err := cel.NewEnv(
  cel.Variable(”pass", cel.StringType),
  ...
  cel.Variable("Min", cel.AnyType),
  cel.Variable(”Max", cel.AnyType),
)
```

prepare.go

Использовать тип cel.AnyType и явно приводить типы в правилах

Явное приведение типов

```
password_validation:
  rules:
    - pass.size() >= int(Min)
    - pass.size() <= int(Max)
  params:
    - name: Min
      value: 8
    - name: Max
      value: 16
```

app_config.yaml

```
env, err := cel.NewEnv(
  cel.Variable(”pass", cel.StringType),
  ...
  cel.Variable(”Min", cel.AnyType),
  cel.Variable(”Max", cel.AnyType),
)
```

prepare.go

```
rules:
  - pass.matches(‘^[Charset]{Min,}$’)
params:
  - name: Min
    value: 8
  - name: Charset
    value: “A-Za-z0-9_.-”
```

А вот так уже сделать не получится...

~~Использовать маппинг типов~~

~~Использовать явное приведение типов~~

Использовать шаблоны языка Go

Шаблоны text/template

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.matches(‘^[ {{- .Charset -}} ]{ {{- .MinLen -}} ,}$’)
  params:
    - MinLen: 8
    - Charset: “A-Za-z0-9_.-”
```

app_config.yaml

Шаблоны text/template

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.matches('^[ {{- .Charset -}} ]{ {{- .MinLen -}} ,}$')
  params:
    - MinLen: 8
    - Charset: "A-Za-z0-9_.-"
```

app_config.yaml

Опишем CEL правила с шаблонами text/template

Шаблоны text/template

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.matches('^[ {{- .Charset -}} ]{ {{- .MinLen -}} ,}$')
  params:
    - MinLen: 8
    - Charset: "A-Za-z0-9_.-"
```

app_config.yaml

Теперь это параметры шаблона

Шаблоны text/template

prepare.go

```
import "github.com/google/cel-go/cel"

func prepare(
  rule string,
  params map[string]any,
) (*cel.Program, error) {
  tmpl, err := template.New(”exp").Parse(rule)
  ...
  var parsedRule bytes.Buffer
  err = tmpl.Execute(&parsedRule, tmplParams)
  ...
  ast, issues := env.Compile(parsedRule.String())
  ...
}
```

Шаблоны text/template

prepare.go

```
import "github.com/google/cel-go/cel"

func prepare(
  rule string,
  params map[string]any,
) (*cel.Program, error) {
  tmpl, err := template.New(”exp").Parse(rule)
  ...
  var parsedRule bytes.Buffer
  err = tmpl.Execute(&parsedRule, tmplParams)
  ...
  ast, issues := env.Compile(parsedRule.String())
  ...
}
```

Произведем предварительную компиляцию правила с использованием шаблона

Шаблоны text/template

prepare.go

```
import "github.com/google/cel-go/cel"

func prepare(
  rule string,
  params map[string]any,
) (*cel.Program, error) {
  tmpl, err := template.New(”exp").Parse(rule)
  ...
  var parsedRule bytes.Buffer
  err = tmpl.Execute(&parsedRule, tmplParams)
  ...
  ast, issues := env.Compile(parsedRule.String())
  ...
}
```

Подставим обработанное правило

Делаем ошибки user-friendly

User-friendly ошибки

```
rules:
  - “password.size() >= {{ .MinLen }}“


params:
  - MinLen: 8
```

app_config.yaml

```
Password validation failed:
password.size() >= 8
```

Такую ошибку нельзя отдавать пользователю!

User-friendly ошибки

```
rules:
  - expression: “password.size() >= {{ .MinLen }}“
    message: “password len is less than required”

params:
  - MinLen: 8
```

app_config.yaml

```
Password validation failed:
password len is less than
required
```

Такую ошибку уже можно
отдавать пользователю,
но не понятно в чем он не прав...

User-friendly ошибки

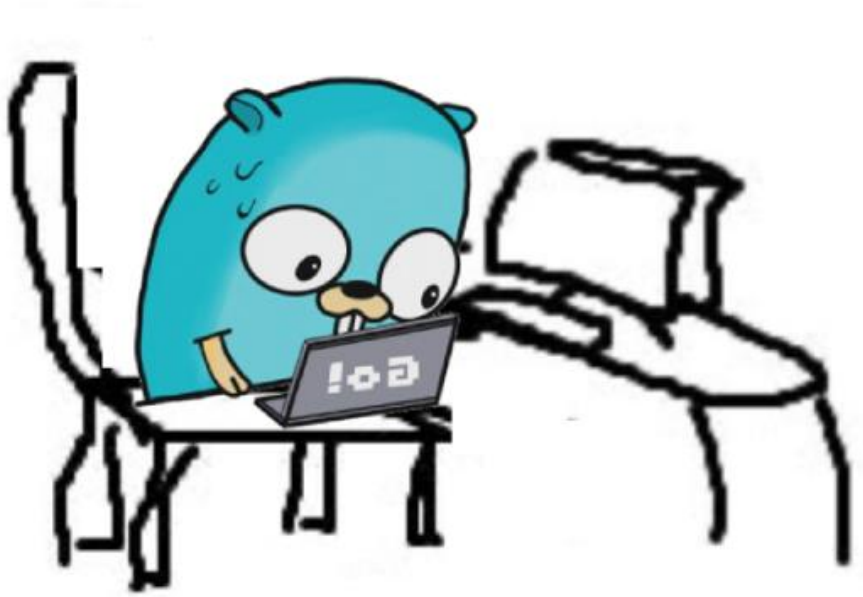
```
rules:
  - expression: “password.size() >= {{ .MinLen }}“
    message: “password len is less than required,
                required: {{ .MinLen }}”
params:
  - MinLen: 8
```

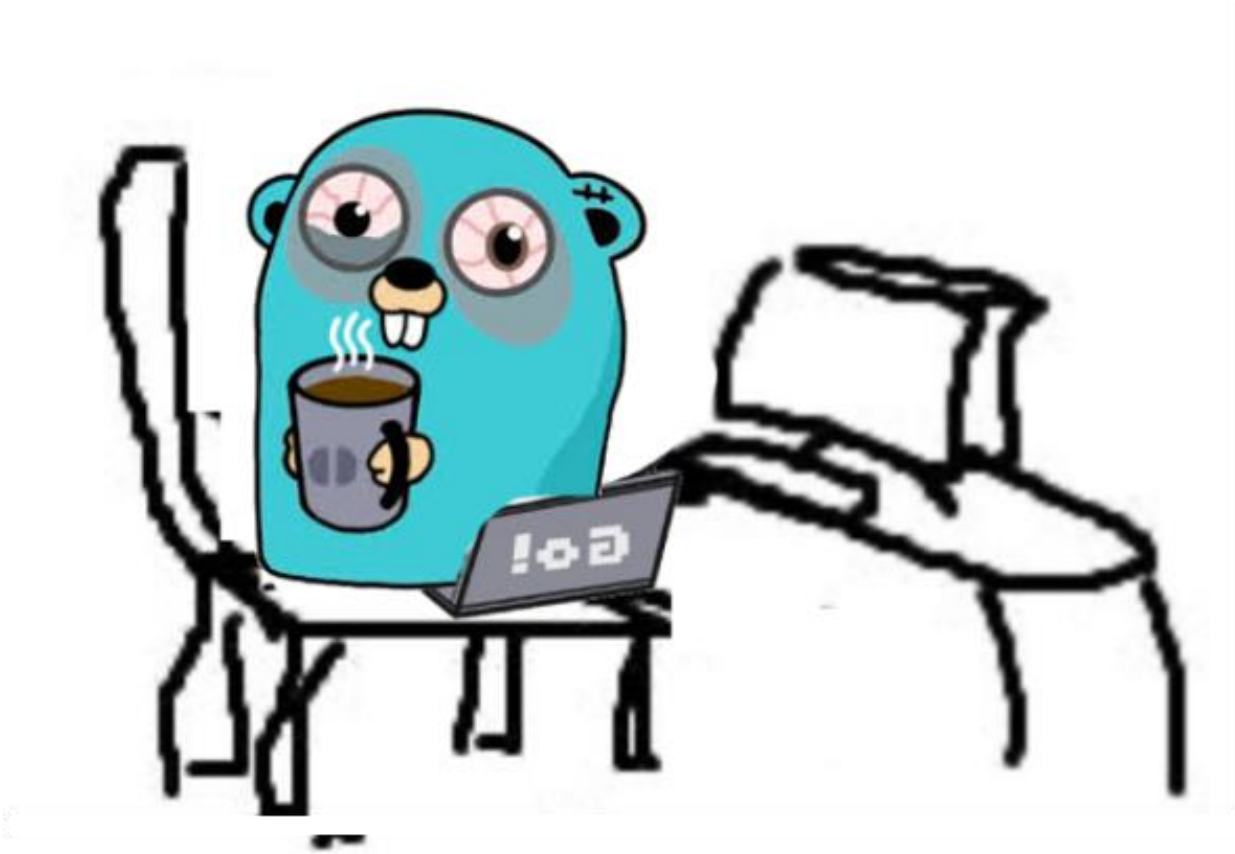
app_config.yaml

```
Password validation failed:
password len is less than
required 8
```

Все понятно и есть численные показатели для пользователя!


ARE YA WINNING,
SON?





“

With Great Power Comes Great Responsibility

Ben Parker
Spider-Man Movie, 2002

Свобода приводит к хаосу

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.size() <= {{ .MaxLen }}
    - password.matches({{ .Regexp }})
    - password.maxRepeatCount({{ .RepCnt }})
  params:
    - MinLen: 8
    - MaxLen: 16
    - Regexp: “^[A-Za-z0-9_.-]{8,16}$”
    - RepCnt: 2
```

app_config.yaml

Свобода приводит к хаосу

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.size() <= {{ .MaxLen }}
    - password.matches({{ .Regexp }})
    - password.maxRepeatCount({{ .RepCnt }})
  params:
    - MinLen: 8
    - MaxLen: 16
    - Regexp: “^[A-Za-z0-9!@#_.-]{8,16}$”
    - RepCnt: 2
```

app_config.yaml

Введение новых значений в regexp делает менее жесткой, что может привести к использованию символов, на которых может быть завязана дополнительная логика продукта

Свобода приводит к хаосу

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.size() <= {{ .MaxLen }}
    - password.matches({{ .Regexp }})
    - password.maxRepeatCount({{ .RepCnt }})
  params:
    - MinLen: 8
    - MaxLen: 16
    - Regexp: “^[A-Za-z0-9_.-]{8,16}$”
    - RepCnt: 2
```

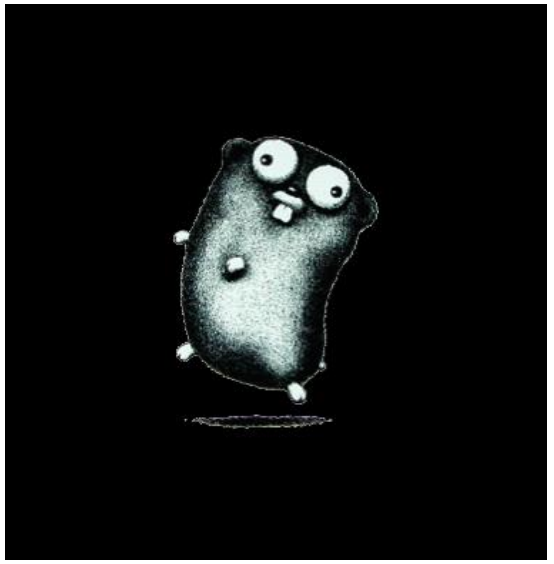
app_config.yaml

Отключение данного правила при переконфигурации позволяет не задавать пароль

Свобода приводит к хаосу

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.size() <= {{ .MaxLen }}
    - password.matches({{ .Regexp }})
    - password.maxRepeatCount({{ .RepCnt }})
  params:
    - MinLen: 16
    - MaxLen: 8
    - Regexp: “^[A-Za-z0-9_.-]{8,16}$”
    - RepCnt: 2
```

app_config.yaml



Указав такие границы длин паролей произойдет поломка системы проверки и никто не сможет установить пароль

Ограничим абсолютную свободу

Что хотим ограничить?

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.size() <= {{ .MaxLen }}
    - password.matches({{ .Regexp }})
    - password.maxRepeatCount({{ .RepCnt }})
  params:
    - MinLen: 8
    - MaxLen: 16
    - Regexp: “^[A-Za-z0-9_.-]{8,16}$”
    - RepCnt: 2
```

app_config.yaml

Что хотим ограничить?

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.size() <= {{ .MaxLen }}
    - password.matches({{ .Regexp }})
    - password.maxRepeatCount({{ .RepCnt }})
  params:
    - MinLen: 8
    - MaxLen: 16
    - Regexp: “^[A-Za-z0-9_.-]{8,16}$”
    - RepCnt: 2
```

app_config.yaml

Эти правила должны быть спрятаны в app_config.yaml от пользователя

Что хотим ограничить?

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.size() <= {{ .MaxLen }}
    - password.matches({{ .Regexp }})
    - password.maxRepeatCount({{ .RepCnt }})
  params:
    - MinLen: 8
    - MaxLen: 16
    - Regexp: “^[A-Za-z0-9_.-]{8,16}$”
    - RepCnt: 2
```

app_config.yaml

Эти правила должны быть спрятаны в app_config.yaml от пользователя

Значения этих переменных должна быть возможность валидировать

Что хотим ограничить?

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.size() <= {{ .MaxLen }}
    - password.matches({{ .Regexp }})
    - password.maxRepeatCount({{ .RepCnt }})
  params:
    - MinLen: 8
    - MaxLen: 16
    - Regexp: “^[A-Za-z0-9_.-]{8,16}$”
    - RepCnt: 2
```

app_config.yaml

Эти правила должны быть спрятаны в app_config.yaml от пользователя

Значения этих переменных должна быть возможность валидировать

Продукт хочет, чтобы данный regexp нельзя было менять

Что хотим ограничить?

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.size() <= {{ .MaxLen }}
    - password.matches({{ .Regexp }})
    - password.maxRepeatCount({{ .RepCnt }})
  params:
    - MinLen: 8
    - MaxLen: 16
    - Regexp: "^[A-Za-z0-9_.-]{8,16}$"
    - RepCnt: 2
```

app_config.yaml

Эти правила должны быть спрятаны в app_config.yaml от пользователя

Значения этих переменных должна быть возможность валидировать

Продукт хочет, чтобы данный regexp нельзя было менять

А вот это правило и переменные должна быть возможность изменять по желанию пользователя

Что хотим ограничить?

```
password_validation:
  rules:
    - password.size() >= {{ .MinLen }}
    - password.size() <= {{ .MaxLen }}
    - password.matches({{ .Regexp }})
    - password.maxRepeatCount({{ .RepCnt }})
  params:
    - MinLen: 8
    - MaxLen: 16
    - Regexp: "^[A-Za-z0-9_.-]{8,16}$"
    - RepCnt: 2
```

app_config.yaml

Эти правила должны быть спрятаны в app_config.yaml от пользователя

Значения этих переменных должна быть возможность валидировать

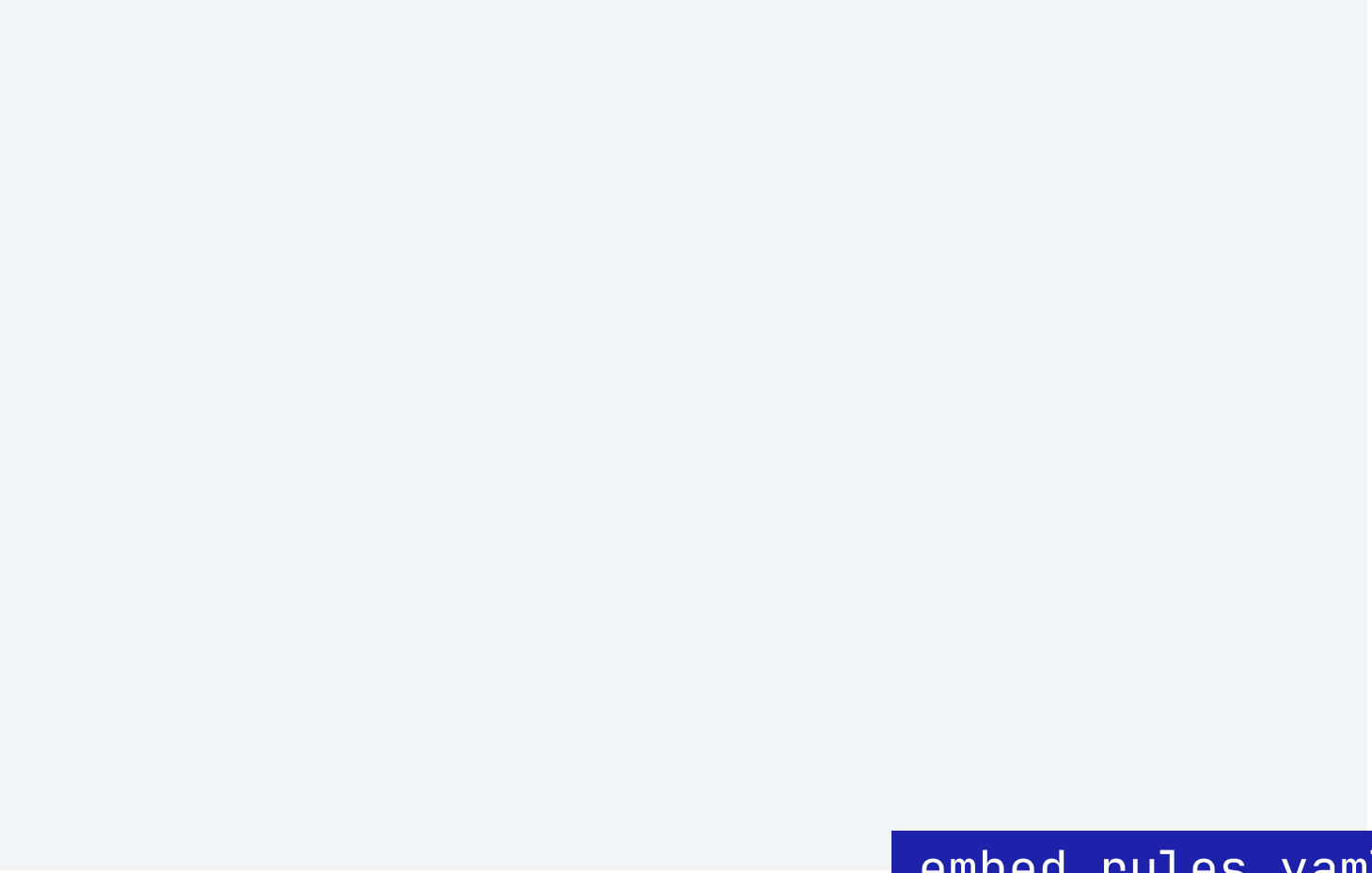
Продукт хочет, чтобы данный regexp нельзя было менять

А вот это правило и переменные должна быть возможность изменять по желанию пользователя

Но при этом продукт должен уметь задавать начальные значения

Будем использовать embedded файлы и buildtime конфигурацию

Использование embedded файла конфигурации



Введем файл конфигурации embed_rules.yaml, который будем подключать в проект в качестве embedded файла

`embed_rules.yaml`

Использование embedded файла конфигурации

```
rules:
  - password.size() >= {{ .MinLen }}
  - password.size() <= {{ .MaxLen }}
  - password.matches( {{ .Regexp }} )
```

Вынесем те правила, которые нельзя изменять в отдельную секцию

`embed_rules.yaml`

Использование embedded файла конфигурации

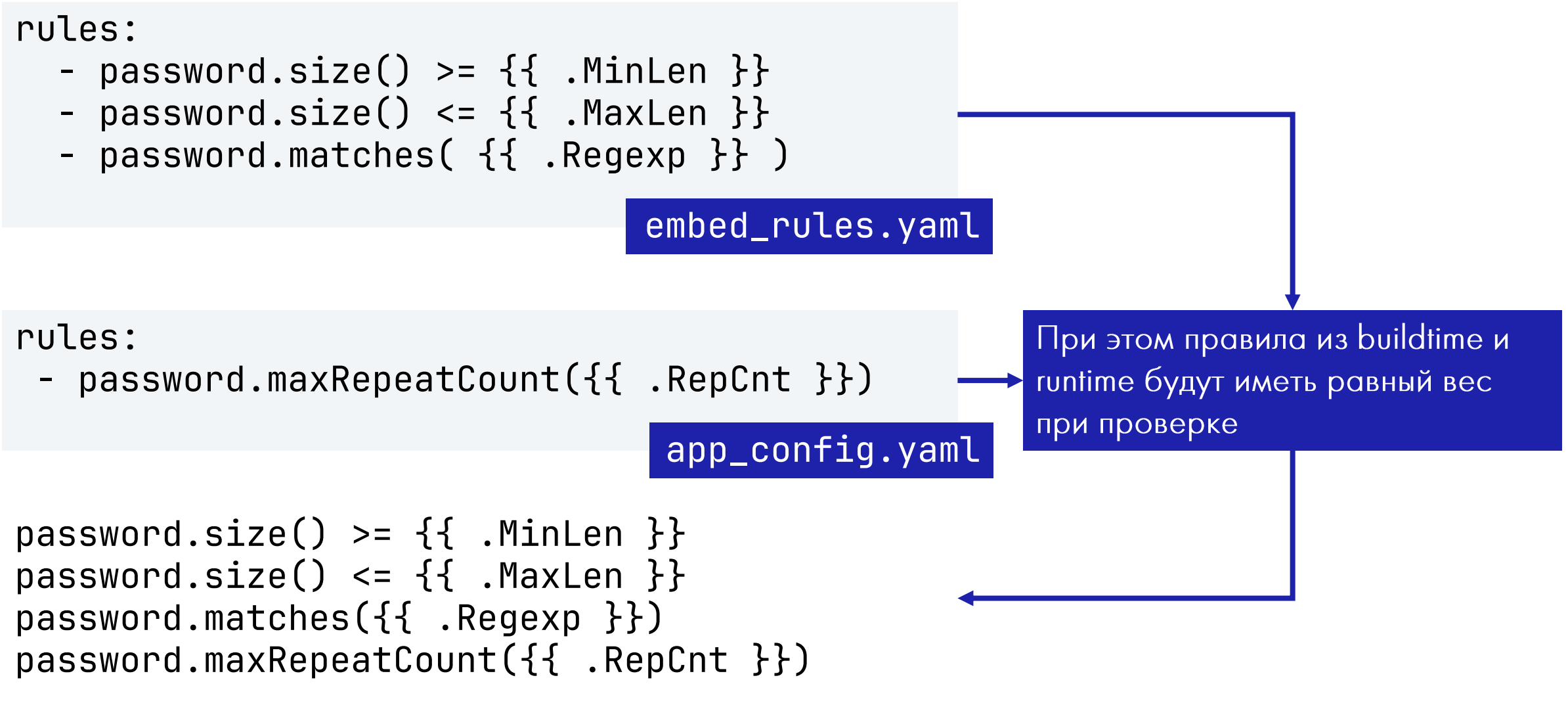
```
rules:
  - password.size() >= {{ .MinLen }}
  - password.size() <= {{ .MaxLen }}
  - password.matches( {{ .Regexp }} )
```

embed_rules.yaml

```
rules:
 - password.maxRepeatCount({{ .RepCnt }})
```

app_config.yaml

Использование embedded файла конфигурации



Использование embedded файла конфигурации

```
rules:
  - password.size() >= {{ .MinLen }}
  - password.size() <= {{ .MaxLen }}
  - password.matches( {{ .Regexp }} )
const:
  - Regexp: “^[A-Za-z0-9_.-]{8,16}$”
```

Выделим переменные, которые нельзя изменять в app_config.yaml

embed_rules.yaml

Использование embedded файла конфигурации

```
rules:
  - password.size() >= {{ .Params.MinLen }}
  - password.size() <= {{ .Params.MaxLen }}
  - password.matches( {{ .Const.Regexp }} )
const:
  - Regexp: “^[A-Za-z0-9_.-]{8,16}$”
```

embed_rules.yaml

Тем самым разделим переменные в правилах на params и const

```
params:
    - MinLen: 8
    - MaxLen: 16
    - Regexp: “^[A-Za-z0-9_.-]{8,16}$”
    - RepCnt: 2
```

app_config.yaml

Использование embedded файла конфигурации

```
rules:
  ...
const:
  ...
params:
  - MinLen: 8
  - MaxLen: 16
  - RepCnt: 1
```

embed_rules.yaml

Введем начальные значения для переменных, которые могут быть изменены в app_config.yaml

Использование embedded файла конфигурации

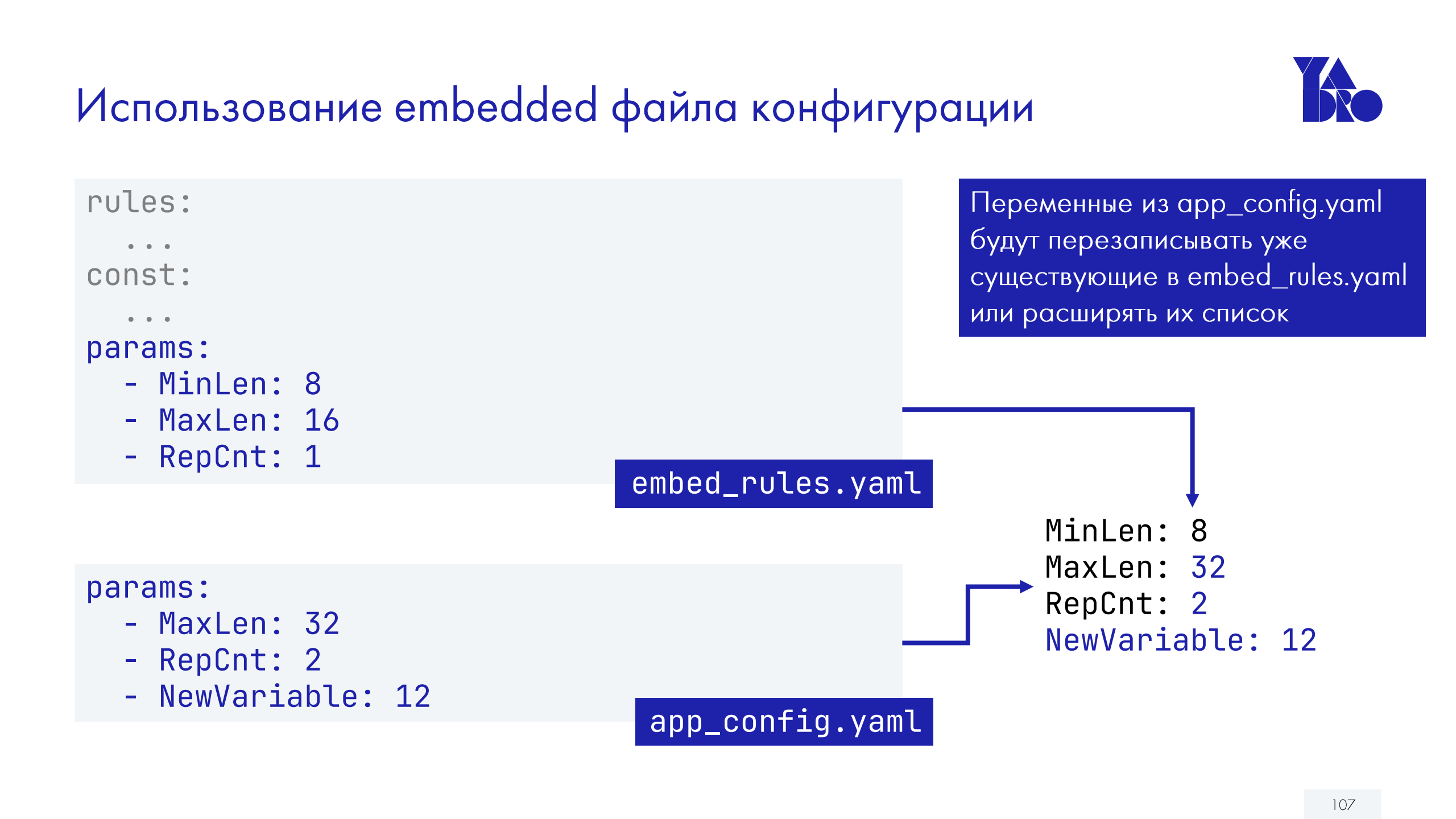
```
rules:
  ...
const:
  ...
params:
  - MinLen: 8
  - MaxLen: 16
  - RepCnt: 1
```

embed_rules.yaml

```
params:
  - MaxLen: 32
  - RepCnt: 2
  - NewVariable: 12
```

app_config.yaml

Использование embedded файла конфигурации



```
rules:
  ...
const:
  ...
params:
  - MinLen: 8
  - MaxLen: 16
  - RepCnt: 1
```

embed_rules.yaml

Переменные из app_config.yaml будут перезаписывать уже существующие в embed_rules.yaml или расширять их список

```
params:
  - MaxLen: 32
  - RepCnt: 2
  - NewVariable: 12
```

app_config.yaml

```
MinLen: 8
MaxLen: 32
RepCnt: 2
NewVariable: 12
```

Использование embedded файла конфигурации

```
rules:
  ...
const:
  ...
params:
  ...
limits:
  - {{ .Params.MinLen }} <= {{ .Params.MinLen }}
  - {{ .Params.MinLen }} >= {{ .Const.Min }}
  - {{ .Params.MinLen }} <= {{ .Const.Max}}
  - {{ .Params.MaxLen }} >= {{ .Const.Min }}
  - {{ .Params.MaxLen }} <= {{ .Const.Max }}
```

embed_rules.yaml

Введем дополнительный набор правил для валидации значения

Использование embedded файла конфигурации

```
rules:
  ...
const:
  ...
  - MinLenRange: 1
  - MaxLenRange: 64
  ...
params:
  ...
limits:
  - {{ .Params.MinLen }} <= {{ .Params.MinLen }}
  - {{ .Params.MinLen }} >= {{ .Const.Min }}
  - {{ .Params.MinLen }} <= {{ .Const.Max}}
  - {{ .Params.MaxLen }} >= {{ .Const.Min }}
  - {{ .Params.MaxLen }} <= {{ .Const.Max }}
```

embed_rules.yaml

И набор констант, используемых в валидации значений переменных

Итоговый пайплайн

Итоговый пайплайн

До сборки сервиса

Описание неизменяемых правил, констант, переменных и правил валидации значений переменных в embedded файле

Итоговый пайплайн

До сборки сервиса	Описание неизменяемых правил, констант, переменных и правил валидации значений переменных в embedded файле
Сборка сервиса	**Добавление embedded файлов в бинарный файл проекта**

Итоговый пайплайн

До сборки сервиса	Описание неизменяемых правил, констант, переменных и правил валидации значений переменных в embedded файле
Сборка сервиса	Добавление embedded файлов в бинарный файл проекта
Настройка сервиса	Описание правил и переменных в конфиге сервиса

Итоговый пайплайн

До сборки сервиса	Описание неизменяемых правил, констант, переменных и правил валидации значений переменных в embedded файле
Сборка сервиса	Добавление embedded файлов в бинарный файл проекта
Настройка сервиса	Описание правил и переменных в конфиге сервиса
Запуск сервиса	Объединение правил, перезапись и расширение набора переменных

Итоговый пайплайн

До сборки сервиса	Описание неизменяемых правил, констант, переменных и правил валидации значений переменных в embedded файле
Сборка сервиса	Добавление embedded файлов в бинарный файл проекта
Настройка сервиса	Описание правил и переменных в конфиге сервиса
Запуск сервиса	Объединение правил, перезапись и расширение набора переменных Валидация переменных с использованием правил-лимитов

Итоговый пайплайн

До сборки сервиса	Описание неизменяемых правил, констант, переменных и правил валидации значений переменных в embedded файле		
Сборка сервиса	Добавление embedded файлов в бинарный файл проекта		
Настройка сервиса	Описание правил и переменных в конфиге сервиса		
Запуск сервиса	Объединение правил, перезапись и расширение набора переменных	Валидация переменных с использованием правил-лимитов	Сборка правил в CEL программы для их дальнейшего использования

Итоговый пайплайн

До сборки сервиса	Описание неизменяемых правил, констант, переменных и правил валидации значений переменных в embedded файле		
Сборка сервиса	Добавление embedded файлов в бинарный файл проекта		
Настройка сервиса	Описание правил и переменных в конфиге сервиса		
Запуск сервиса	Объединение правил, перезапись и расширение набора переменных	Валидация переменных с использованием правил-лимитов	Сборка правил в CEL программы для их дальнейшего использования
Валидация	Использование уже скомпилированных правил для валидации паролей		

Подведем итоги

Выводы

Выводы

Хардкод – зло

Выводы

Хардкод – зло

Конфигурация в файле – хорошо

Выводы

Хардкод – зло

Конфигурация в файле – хорошо

Сложная конфигурация – простой DSL

Выводы

Хардкод – зло

Конфигурация в файле – хорошо

Сложная конфигурация – простой DSL

Абсолютная свобода всегда ведет к хаосу

Выводы

Хардкод – зло

Конфигурация в файле – хорошо

Сложная конфигурация – простой DSL

Абсолютная свобода всегда ведет к хаосу

embed – это не хардкод

Выводы

Хардкод – зло

Конфигурация в файле – хорошо

Сложная конфигурация – простой DSL

Абсолютная свобода всегда ведет к хаосу

embed – это не хардкод

embed – это путь к безопасности

Больше информации

Больше информации

О языке CEL

Краткая история, зачем нужен, а также ссылки на исходный код.

cel.dev

Больше информации

О языке CEL

Краткая история, зачем нужен, а также ссылки на исходный код.
cel.dev

cel-go

Библиотека-обертка для Go
github.com/google/cel-go

Больше информации

О языке CEL

Краткая история, зачем нужен, а также ссылки на исходный код.
cel.dev

cel-go

Библиотека-обертка для Go
github.com/google/cel-go

Langdef

Описание синтаксиса и примеры использования
github.com/google/cel-spec/langdef.md

Больше информации

О языке CEL

Краткая история, зачем нужен, а также ссылки на исходный код.
cel.dev

cel-go

Библиотека-обертка для Go
github.com/google/cel-go

Langdef

Описание синтаксиса и примеры использования
github.com/google/cel-spec/langdef.md

proto+buf - плагинизация и "декларативизация"

Игорь Цигляр, YADRO

Скоро на Habr компании YADRO

Если остались вопросы

Если остались вопросы

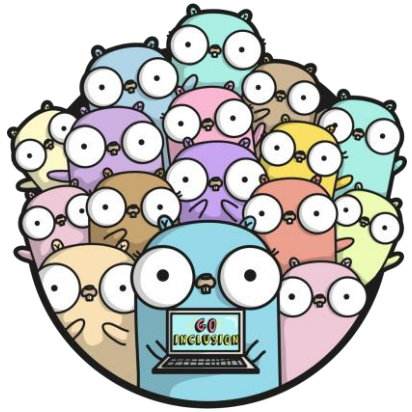
Пишите в Telegram

t.me/andrianovartemii

Если остались вопросы

Пишите в Telegram

t.me/andrianovartemii



Приходите на стенд компании

Чтобы узнать больше о компании и пообщаться со спикерами и инженерами компании

БУДУЩЕЕ
В НАШИХ
РУКАХ