



KubeCon



CloudNativeCon

Europe 2026

How To Add A New Language Feature To OPA

Charlie Egan — Apple

OPA Introduction

"Can this runner deploy to production?"

```
Evaluate | Debug
allow if {
  "prod" in input.runner.tags
  input.target == "prod"
}
```

"Only admins can delete users"

```
Evaluate | Debug
allow if {
  input.method == "DELETE"
  "admin" in input.user.roles
}
```

"Changes to deploy/*.yaml files must be approved by the prod-eng team"

```
Evaluate | Debug
approvals contains "prod-eng" if {
  some file in input.changed
  startswith(search: file, base: "deploy/")
}
```

OPA Introduction

```
Evaluate | Debug
└─ allow if {
  │ "prod" in input.runner.tags
  │ input.target == "prod"
  └─ }

Evaluate | Debug
└─ allow if {
  │ input.method == "DELETE"
  │ "admin" in input.user.roles
  └─ }

Evaluate | Debug
└─ approvals contains "prod-eng" if {
  │ some file in input.changed
  │ startswith(search; file, base: "deploy/")
  └─ }
```



String Interpolation in Rego

Evaluate | Debug

```
deny contains sprintf(format: "%s's role %q is not allowed", values: [username, role]) if {  
  not role in allowed_roles  
}
```

Evaluate

```
default username := ""
```

Evaluate | Debug

```
username := input.username
```

Evaluate

```
default role := "guest"
```

Evaluate | Debug

```
role := input.role
```

Evaluate

```
allowed_roles := ["admin", "employee"]
```



KubeCon



CloudNativeCon

Europe 2026

String Interpolation in Rego

Evaluate | Debug

```
deny contains "${input.username}'s role '{role}' is not allowed" if {  
  |   not role in allowed_roles  
}
```

Evaluate

```
default role := "guest"
```

Evaluate | Debug

```
role := input.role
```

Evaluate

```
allowed_roles := ["admin", "employee"]
```



KubeCon



CloudNativeCon

Europe 2026

Step 1: Update the Parser

The parser scans source code and breaks it into tokens
(keywords, identifiers, numbers, strings, operators, etc.).

Evaluate | Debug

```
package example
```

Evaluate

```
kubecon_location := "Amsterdam"
```

Evaluate | Debug

```
message := $"KubeCon is in {kubecon_location}!"
```

Step 1: Update the Parser

New tokens:

TemplateStringPart, TemplateStringEnd

Evaluate | Debug

```
package example
```

Evaluate

```
kubecon_location := "Amsterdam"
```

Evaluate | Debug

```
message := $"KubeCon is in {kubecon_location}!"
```



KubeCon



CloudNativeCon

Europe 2026

Step 2: Update the AST

The parser takes tokens and builds an Abstract Syntax Tree (AST)
(Modules, Packages, Rules & Value types)

Evaluate | Debug

```
package example
```

Evaluate

```
kubecon_location := "Amsterdam"
```

Evaluate | Debug

```
message := $"KubeCon is in {kubecon_location}!"
```



KubeCon



CloudNativeCon

Europe 2026

Step 2: Update the AST

New TemplateString AST Node

Contains the static strings and the dynamic expressions

Evaluate | Debug

```
package example
```

Evaluate

```
kubecon_location := "Amsterdam"
```

Evaluate | Debug

```
message := $"KubeCon is in {kubecon_location}!"
```

Step 3: Update Compiler

The compiler takes the AST and makes it executable

Rego code is transformed through a number of steps

Evaluate | Debug

```
package example
```

Evaluate

```
kubecon_location := "Amsterdam"
```

Evaluate | Debug

```
message := $"KubeCon is in {kubecon_location}!"
```

Step 3: Update Compiler

Some parts of the AST are adapted for evaluation

We added a new RewriteTemplateStrings step

```
4 - message := $"KubeCon is in {data.example.kubecon_location}!"  
5+ message := internal.template_string(["KubeCon is in ", {data.example.kubecon_location}, "])  
6
```

Step 4: Update Interpreter

Rego's topdown interpreter is responsible for running compiled Rego

A new function: `internal.template_string` builds the final string

```
4 - message := $"KubeCon is in {data.example.kubecon_location}!"  
5+ message := internal.template_string("KubeCon is in ", {data.example.kubecon_location}, "}")  
6
```

Step 4: Update Interpreter

`internal.template_string` processes each part recursively
And then concatenates the results

```
func builtinTemplateString(bctx BuiltinContext, operands []*ast.Term, iter func(*ast.Term) error) error {
    arr, err := builtins.ArrayOperand(operands[0].Value, 1)
    if err != nil {
        return err
    }

    buf := make([]string, arr.Len())

    var count int
    err = builtinPrintCrossProductOperands(bctx.Location, buf, arr, 0, func(buf []string) error {
        count += 1
        // Precautionary run-time assertion that template-strings can't produce multiple outputs; e.g. for
        // custom relation type built-ins not known at compile-time.
        if count > 1 {
            return Halt{Err: &Error{
                Code:    ConflictErr,
                Location: bctx.Location,
                Message: "template-strings must not produce multiple outputs",
            }}
        }
        return nil
    })

    if err != nil {
        return err
    }

    return iter(ast.StringTerm(strings.Join(buf, "")))
}
```



KubeCon



CloudNativeCon

Europe 2026

Step 5: Everything Else!

TextMate
Grammar

Regal Linter
Parsing

New Regal Linter
Rules

Treesitter
Grammar

GitHub Linguist

Neovim

OPA Go-To-
Definition

Helix Editor

Documentation

VS Code Extension

Codemirror

Rego Playground

Social Posts

Lots of tests!

fmt command

WASM
Implementation

OPA at KubeCon

Come to our talk!

OPA Maintainer Track Session
Wednesday 14:15 - 14:45
E103-105

*More cool
language updates!*

Come to our kiosk!

Project Pavillion
All Week!





KubeCon



CloudNativeCon

Europe 2026

