K8C

Product description

The product is a platform for deploying, managing, and scaling machine learning models in production. It offers a secure, flexible environment for automating ML tasks like model versioning, routing, and monitoring. With Kubernetes integration and containerization support, it's designed for developers, ML engineers, and enterprises needing scalable, reliable ML infrastructure.

Team K8C: Tsurkan Daniel; Dandamaev Gadji; Tsaturyan Konstantin; Smolkin Mikhail

Project repo: https://github.com/fanglores/Advanced-Software-Design

Promts

Suggest {number_of_frameworks} microservice frameworks from {git_url/list_of_frameworks} that correspond the following project the best. {project_description}. It has features {features_list}. Provide only chosen frameworks with a brief description.

Consider MVP must be developed within {N} days in a small team, having skills only in {programming_lang}. Provide only {M} frameworks which fits team the best for current project. Mind that only one will be used to implement a project. Provide chosen frameworks with a brief description.

Justify your choice.

For these {number_of_frameworks} frameworks create implementation of a microservice program representing custom Kubernetes operator for answering "Hello world!" to any client connected. Write only source code for the program.

Compare implementations for these frameworks.
Which one would you pick for developing in current conditions?

See the full logs in research.log.md files

Main candidates

FastAPI

Supports OpenAPI schema generation out of the box and provides built-in tools for request validation. It's simple to deploy with Kubernetes, making it an excellent choice for rapid prototyping and building an API Gateway with features like SSO and caching.

Kong

Supports OpenAPI schema generation out of the box and provides built-in tools for request validation. It's simple to deploy with Kubernetes, making it an excellent choice for rapid prototyping and building an API Gateway with features like SSO and caching.

Traefik

A Kubernetes-native reverse proxy and load balancer designed for microservices. It has seamless CRD support for Kubernetes and integrates easily with authentication and caching middleware. Traefik’s declarative configuration simplifies deployment, aligning well with operator-focused projects.

Code examples: Traefik

Traefik Implementation (Go)

go

Копировать код

```go
package main

import (
    "context"
    "fmt"
    "log"
    "net/http"
    "os"
    "os/signal"
    "syscall"

    "github.com/labstack/echo/v4"
    "k8s.io/client-go/kubernetes"
    "k8s.io/client-go/rest"
    "k8s.io/apimachinery/pkg/watch"
)

func main() {
    // Start Echo HTTP server
    e := echo.New()
    e.GET("/", func(c echo.Context) error {
        return c.JSON(http.StatusOK, map[string]string{"message": "Hello, world!"})
    })
```

Копировать код

```go
    go func() {
        if err := e.Start(":8000"); err != nil && err != http.ErrServerClosed {
            log.Fatalf("Shutting down the server: %v", err)
        }
    }()

    // Start Kubernetes operator
    go func() {
        config, err := rest.InClusterConfig()
        if err != nil {
            log.Fatalf("Error building Kubernetes config: %v", err)
        }
        clientset, err := kubernetes.NewForConfig(config)
        if err != nil {
            log.Fatalf("Error creating Kubernetes client: %v", err)
        }

        watcher, err := clientset.CoreV1().Pods("").Watch(context.TODO(), rest.ListOptions
        if err != nil {
            log.Fatalf("Error starting watch: %v", err)
        }

        ch := watcher.ResultChan()
        log.Println("Starting operator...")
        for event := range ch {
            fmt.Printf("Event: %v\n", event.Type)
        }
    }()

    // Graceful shutdown
    quit := make(chan os.Signal, 1)
    signal.Notify(quit, os.Interrupt, syscall.SIGTERM)
    <-quit
    log.Println("Shutting down server...")
    e.Shutdown(context.Background())
}
```

Code examples: Kong

Kong Declarative Configuration (YAML)

yaml Копировать код

```yaml
_format_version: "3.0"
services:
  - name: hello-world-service
    url: http://localhost:8080 # Your backend service URL
    routes:
      - name: hello-world-route
        paths:
          - /hello
        strip_path: false
        methods:
          - GET
```

Backend Service (Optional in Go)

If a simple backend is needed to respond with "Hello, world!":

go Копировать код

```go
package main

import (
    "fmt"
    "net/http"
)

func helloWorldHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, `{"message": "Hello, world!"}`)
}

func main() {
    http.HandleFunc("/", helloWorldHandler)
    http.ListenAndServe(":8080", nil)
}
```

Code examples: FastAPI

FastAPI Implementation (Python)

python

Копировать код

```python
from fastapi import FastAPI
from kubernetes import client, config
from fastapi.responses import JSONResponse

app = FastAPI()

# Load Kubernetes configuration (assumes in-cluster configuration)
config.load_incluster_config()

# Define a Kubernetes CRD for the operator
CRD_GROUP = "example.com"
CRD_VERSION = "v1"
CRD_PLURAL = "hellos"
NAMESPACE = "default"

@app.get("/")
async def hello_world():
    return {"message": "Hello, world!"}

@app.on_event("startup")
async def setup_crd():
    """Ensure the CRD is registered."""
    api = client.ApiextensionsV1Api()
    crd = client.V1CustomResourceDefinit
```

Копировать код

```python
    crd = client.V1CustomResourceDefinition(
        metadata=client.V1ObjectMeta(name="hellos.example.com"),
        spec=client.V1CustomResourceDefinitionSpec(
            group=CRD_GROUP,
            versions=[client.V1CustomResourceDefinitionVersion(
                name=CRD_VERSION,
                served=True,
                storage=True
            )],
            scope="Namespaced",
            names=client.V1CustomResourceDefinitionNames(
                plural=CRD_PLURAL,
                singular="hello",
                kind="Hello",
                shortNames=["hw"]
            )
        )
    )
    try:
        api.create_custom_resource_definition(crd)
    except client.exceptions.ApiException as e:
        if e.status != 409:
            raise

@app.get("/crd")
async def list_custom_hellos():
    """List all custom resources of kind 'Hello'."""
    api = client.CustomObjectsApi()
    hellos = api.list_namespaced_custom_object(
        group=CRD_GROUP,
        version=CRD_VERSION,
        namespace=NAMESPACE,
        plural=CRD_PLURAL
    )
    return JSONResponse(content=hellos)
```

Comparison

Aspect	FastAPI	Kong	Traefik
Ease of Setup	Requires coding the entire operator logic (CRD management).	Declarative configuration makes it simpler and faster.	Moderate: Requires Go setup and dependency management with go mod
Development Speed	Moderate: More custom coding required.	High: Pre-built API Gateway features speed up development.	Moderate: Slower due to boilerplate and strict typing.
Flexibility	Full control over API behavior and customizations.	Limited to what the configuration and plugins allow.	Moderate: Limited by Go's strict typing, but Echo simplifies API creation.
Kubernetes Integration	Deep integration via Python Kubernetes SDK.	Strong with existing ingress and routing capabilities.	High: Native Go Kubernetes client offers seamless integration.
Performance	Lightweight with minimal dependencies.	Slight overhead with Kong as a proxy layer.	High: Compiled language with efficient concurrency
Team Skills	Ideal for a Python-skilled team.	Better if Golang familiarity exists but not critical.	Moderate: Requires Go expertise, which may not be familiar to all
Scalability	Requires scaling the service explicitly.	Highly scalable by design due to Kong's architecture.	High: Go's goroutines support lightweight and scalable concurrency
Extensibility	Can easily add custom logic, validations, and features.	Extensible through plugins but less customizable than code.	Moderate: Smaller library ecosystem, so requires custom coding for advanced features

Conclusion

We pick **FastAPI** because it offers faster development speed due to built-in features like automatic OpenAPI generation and validation, making it ideal for rapid MVPs. Its ease of setup, team's Python familiarity, and flexibility make it more accessible for a small team with limited time.